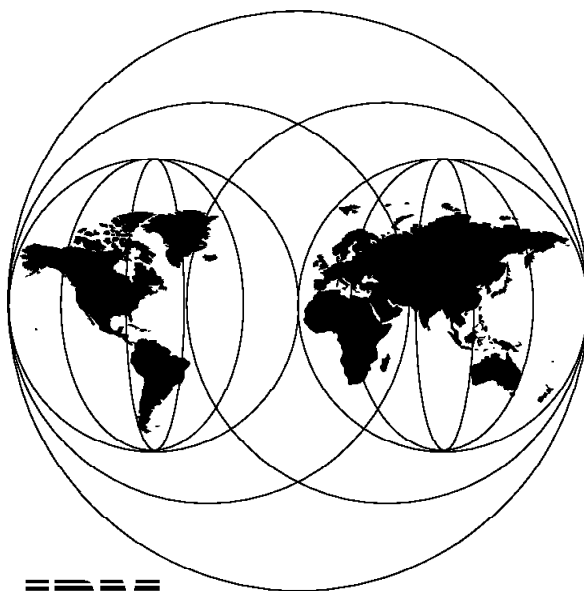


SG24-4668-00

International Technical Support Organization

**OS/2 Security Enabling Services:
A Developer's Guide**

May 1996



IBM

**International Technical Support Organization
Boca Raton Center**



SG24-4668-00

International Technical Support Organization

**OS/2 Security Enabling Services:
A Developer's Guide**

May 1996

Take Note!

Before using this information and the product it supports, be sure to read the general information under "Special Notices" on page xv.

First Edition (May 1996)

This edition applies to OS/2 Security Enabling Services for use with the OS/2 V2.11 or OS/2 Warp V3.0 plus a fixpak.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

An ITSO Technical Bulletin Evaluation Form for reader's feedback appears facing Chapter 1. If the form has been removed, comments may be addressed to:

IBM Corporation, International Technical Support Organization
Dept. JLPC Building 014 Internal Zip 5220
1000 NW 51st Street
Boca Raton, Florida 33431-1328

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© **Copyright International Business Machines Corporation 1996. All rights reserved.**
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Abstract

This document describes the Security Enabling Services for OS/2 Version 2.11 and OS/2 Warp, in respect to its functions that may be utilized by software developers. It provides a discussion and examples of using this new feature and documents the API so that developers can produce security products that interface closely with OS/2 and offer C2 like functionality.

This document was written for developers, and IBM and customer technical personnel. Some knowledge of OS/2 and security issues is assumed.

(331 pages)

Contents

Abstract	iii
Figures	xi
Tables	xiii
Special Notices	xv
Preface	xvii
How This Document is Organized	xvii
International Technical Support Organization Publications	xvii
ITSO Redbooks on the World Wide Web (WWW)	xviii
Acknowledgments	xix

Part 1. Developer's Guide	1
Chapter 1. Introduction to Security Enabling Services	3
1.1 Prerequisite Knowledge	4
1.2 Chapter Breakdown	4
Chapter 2. Security Requirements	7
2.1 Orange Book Security Criteria	10
2.1.1 Continuous Protection	12
2.1.2 Security Policy	13
2.1.3 Accountability	15
2.1.4 Assurance	17
Chapter 3. Security Enabling Services Overview	19
3.1 Installable Security Subsystem (ISS)	21
3.2 Security Enabling Services	21
3.2.1 Security Kernel Services	22
3.2.2 Security Context Services	22
3.2.3 Logon Shell Services	23
3.2.4 Installation, Configuration, Initialization Support	23
3.3 SES and ISS Communication	23
3.3.1 Summary	25
Chapter 4. Security Enabling Services	27
4.1 SES Overview	27

4.2 Security Kernel Services (SKS)	29
4.2.1 Security Relevant Event Interception and Routing (Hooks)	31
4.2.2 Kernel Level Operating System Services	31
4.3 Security Context Services (SCS)	31
4.3.1 Process, Handle and Thread Models	33
4.3.2 Multiple Concurrently Active Security Applications	39
4.3.3 Multiple Concurrently Active Users	41
4.3.4 Trusted Program/Process	42
4.4 Logon Shell Services (LSS)	43
4.4.1 Overview of Key Logon Shell Services Components	46
4.4.2 Overview of Key LSS Operations	51
4.5 Installation, Configuration and Initialization Support (ICIS)	62
Chapter 5. Installable Security Subsystem	65
5.1 What Is an Installable Security Subsystem?	66
5.2 What Are the Typical Components of an ISS?	67
5.3 What Support Does SES Provide for an ISS?	70
5.3.1 Security Context	70
5.3.2 Privileges and Authorities	70
5.3.3 Programming Interfaces	76
5.4 Installable Security Subsystem Summary	77

Part 2. Design Notes 79

Chapter 6. Introduction to SES Development	81
6.1 Chapter Breakdown	82
Chapter 7. Building an Installable Security Subsystem	85
7.1 Logon	86
7.2 Resource Access Control	87
7.3 Audit	88
7.4 Single Signon	89
7.5 Trusted Program Support	89
Chapter 8. Installable Security Subsystem Design Guidelines	91
Chapter 9. SES Architecture Implementation	95
9.1 Security Kernel Services	96
9.1.1 Security Event Router	98
9.1.2 Security Helpers	99
9.2 Security Context Services	100
9.2.1 Process-User Association	102

9.2.2	Process-Status Association	108
9.2.3	Definition of Security Context	110
9.2.4	SCS Programming Interfaces	128
9.3	Logon Shell Services	130
9.3.1	Key Components	131
9.3.2	Definition of Local System Logon	135
9.3.3	Definition of a Guest User	136
9.3.4	Overview of Keyboard/Mouse Support	139
9.3.5	LSS Programming Interfaces	140
9.4	Installation, Configuration, Initialization Support	142
9.4.1	Installation	142
9.4.2	Configuration	142
9.4.3	Initialization	144
Chapter 10.	Interoperation of SES and ISS	147
Chapter 11.	Security Kernel Services (SKS)	151
11.1	Kernel Callouts Imported from the ISS	152
11.1.1	Callouts for Security Relevant OS/2 System Calls	152
11.1.2	Callouts for Callgate Level Support	153
11.1.3	Callouts for Multiple Virtual DOS Machine Support	154
11.1.4	Callouts for Logon Shell Services Trusted Path Support	155
11.1.5	Callouts for Security Enabling Services API Audit Support	156
11.2	Kernel Services Exported to the ISS	156
11.2.1	Security Helpers for File System Services	156
11.2.2	Security Helpers for Security Context Services	156
11.3	Security Kernel Services KPI	156
11.4	Security Kernel Services Scenarios	157
11.4.1	ISS Security Kernel Initialization	158
11.4.2	File System Open Callout	159
Chapter 12.	Security Context Services (SCS)	161
12.1	Security Context Authority Roles	161
12.1.1	Access Control Authority (ACA)	161
12.1.2	Agent Process Authority (APA)	162
12.1.3	Client Logon Authority (CLA)	163
12.1.4	Remote Logon Authority (RLA)	165
12.1.5	System Logon Authority (SLA)	166
12.1.6	Server Process Authority (SPA)	167
12.1.7	User Identification Authority (UIA)	170
12.2	Initialization of Security Context Services	171
12.3	Establishment of Security Context at Process Creation	173
12.4	Association of Security Context with OS/2 IPC	177

12.5 Security Context Services API	178
12.6 SCS Scenarios	180
12.6.1 A User Logs on Remotely to an Application Server	180
12.6.2 An Untrusted Process Creates an Untrusted Child Process	183
12.6.3 A Process Sends Its Security Context to an SCA Process	184
12.6.4 An SPA Process Acts As Proxy for Its Client Processes	186
Chapter 13. Logon Shell Services (LSS)	189
13.1 Overview of LSS Event Flows	190
13.1.1 Logon/Unlock	190
13.1.2 Lock/Logoff/Shutdown	191
13.1.3 Change Password	192
13.2 Overview of Keyboard/Mouse Support	192
13.2.1 Trusted Path Support	192
13.2.2 Keyboard/Mouse Activity Detection	193
13.2.3 Keyboard/Mouse Inactivity Detection	194
13.3 System Logon Driver API	194
13.4 Client Logon Driver API	196
13.5 Password Validation Driver API	196
13.6 Logon Shell Services API	197
13.7 Logon Shell Services Kernel Programming Interface	198
13.7.1 Callouts to the ISS Security Kernel for LSS Functions	198
13.8 Logon Shell Services (LSS) Scenarios	198
13.8.1 Logon	198
13.8.2 Unlock	204
13.8.3 Logoff, Shutdown	208
13.8.4 Lock	210
13.8.5 Change Password	212
13.8.6 Create User Profile, Delete User Profile	215
13.8.7 Identification and Authentication	217
13.8.8 Send Security Context	220
13.8.9 Process Creation	221
13.8.10 Trusted Path	223
Chapter 14. Installation, Configuration, Initialization Support	225
14.1 Installation	225
14.2 Configuration	226
14.2.1 CONFIG.SYS	226
14.2.2 SECURE.SYS	227
14.3 Initialization	230

Part 3. Appendices	235
-------------------------------------	-----

Chapter 15. KPI and API Calls	237
15.1 Chapter Breakdown	237
Appendix A. Security Kernel Services KPI Details	239
Appendix B. Security Context Services (SCS) API Details	241
B.1 SESControlProcessCreation	241
B.2 SESCreateHandleNotify	242
B.3 SESCreateInstanceHandle	243
B.4 SESCreateSubjectHandle	244
B.5 SESDeleteHandleNotify	245
B.6 SESDeleteSubjectHandle	246
B.7 SESKillProcess	247
B.8 SESlogIntegrityViol	248
B.9 SESQueryAuthorityID	248
B.10 SESQueryContextStatus	249
B.11 SESQueryProcessInfo	250
B.12 SESQuerySecurityContext	253
B.13 SESQuerySubjectHandle	254
B.14 SESQuerySubjectHandleInfo	255
B.15 SESQuerySubjectInfo	256
B.16 SESReleaseSubjectHandle	258
B.17 SESReserveSubjectHandle	259
B.18 SESResetThreadContext	260
B.19 SESSendSecurityContext	261
B.20 SESSetContextStatus	262
B.21 SESSetSecurityContext	264
B.22 SESSetSubjectHandle	266
B.23 SESSetSubjectInfo	267
Appendix C. System Logon Driver API Details	271
C.1 SLDInit	271
C.2 SLDQueryUIA	271
Appendix D. Client Logon Driver API Details	275
D.1 CLDInit	275
D.2 CLDQueryCLA	275
Appendix E. Password Validation Driver API Details	279
E.1 PVDValidatePassword	279
Appendix F. Logon Shell Services API Details	281
F.1 SESControlKBDMonitors	281

F.2	SESIactivityNotify	282
F.3	SESRegisterDaemon	283
F.4	SESReturnEventStatus	285
F.5	SESReturnWaitEvent	288
F.6	SESStartEvent	290
F.7	SESWaitEvent	292
Appendix G. Security Enabling Services Error Codes		299
Appendix H. Customer Thoughts on Security Products		301
Glossary		309
List of Abbreviations		317
Index		321

Figures

1.	Components of a Secured OS/2 Workstation	20
2.	SES - Key Components of the OS/2 Enabling Strategy	28
3.	SKS - Hooks and Services for the ISS Security Kernel	30
4.	SCS - Single Process Model	32
5.	SCS - Multiple Process Model	34
6.	SCS - Subject Handle Model	35
7.	SCS - Trusted Process Model	36
8.	SCS - Thread Context Model	38
9.	LSS - Coordination of Logon Session Events	46
10.	LSS - Overview of Logon Session Events	54
11.	ISS - Overview of a Secured OS/2 System	66
12.	ISS - Components	67
13.	ISS - APA and SPA	73
14.	ISS - Interoperation of Security Context Authorities	75
15.	ISS - Kernel Programming Interface	77
16.	SES Architecture - Overview of Key Components	96
17.	SKS Architecture - Interaction between SKS and ISS Security Kernel	97
18.	SCS Architecture - User/Group/Process Security Credentials	101
19.	SCS Architecture - Subject Handles	104
20.	SCS Architecture - Instance Handles	107
21.	SCS Architecture - Process Authority	109
22.	LSS Architecture - Interaction between LSS and Security Applications	132
23.	LSS Architecture - Overview of Keyboard/Mouse Support	139
24.	SES System Design - Interoperation of Key Components	148
25.	SKS System Design - OS/2 System Call Hooks	153
26.	SKS System Design - OS/2 Callgate Level Hooks	154
27.	SKS System Design - ISS Initialization Scenario	158
28.	SKS System Design - File Open Callout Scenario	159
29.	SCS System Design - Trusted Agent/Server Programs	169
30.	SCS System Design - Remote Logon Scenario	181
31.	SCS System Design - Process Creation Scenario	183
32.	SCS System Design - Inter-Process Communication Scenario	184
33.	SCS System Design - Trusted Server Process Scenario	186
34.	ICIS System Design - Example of SES and ISS Initialization	233

Tables

1.	Suggested Mapping of SES Privileges to ISS Functions	68
2.	Applicable SCA, API, KPI, and DLL for Logon	87
3.	Applicable SCA, API, KPI, and DLL for DAC	88
4.	Applicable SCA, API, KPI, and DLL for Audit	88
5.	Applicable SCA, API, KPI, and DLL for Single Signon	89
6.	Applicable SCA, API, KPI, and DLL for Trusted Program Support	90
7.	Logon Shell Services Functions	197
8.	Address of Subject Information	244
9.	Effective Security Context Structure	264
10.	SES Status on Input	273
11.	SES Status on Output	273
12.	Event Status and Sources	294

Special Notices

This publication is intended to help developers, IBM and customer support personnel as well as dealers, understand the features of Security Enabling Services. It provides the means for people to develop security solutions for their environment or customers. A developer must understand the background information provided before developing a secure OS/2 solution.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 208 Harbor Drive, Stamford, CT 06904 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (VENDOR) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

DB2/2

IBM

OS/2
Workplace Shell

RACF

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Microsoft, Windows, and the Windows 95 logo are trademarks of Microsoft Corporation.

Other trademarks are trademarks of their respective companies.

Preface

This document is intended to provide an overview of OS/2 Security Enabling Services. It contains information about the security enabling services which may be installed on top of OS/2 as part of a customer's security strategy. This facility is available for OS/2 V2.11 and OS/2 Warp.

This document is intended for developers, IBM customers and employees requiring an overview and API guide for security enabling services.

How This Document is Organized

The document is organized as follows:

- Part 1, "Developer's Guide"

This provides an overview of Security Enabling Services and the requirements for workstation security and the functional requirements for C2.

- Part 2, "Design Notes"

This provides details of the Security Enabling Services architecture and information required to allow a developer to build an ISS.

- Part 3, "Appendices"

This provides detailed information on the KPI and API that is available in Security Enabling Services.

International Technical Support Organization Publications

- *OS/2 Security Enabling Services*, SG24-4568
- *Security Enhancement Solutions for Workstations*, SG24-4569

A complete list of International Technical Support Organization publications, known as redbooks, with a brief description of each, may be found in:

International Technical Support Organization Bibliography of Redbooks, GG24-3070.

To get a catalog of ITSO redbooks, VNET users may type:

TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG

A listing of all redbooks, sorted by category, may also be found on MKTTOOLS as ITSOCAT TXT. This package is updated monthly.

How to Order ITSO Redbooks

IBM employees in the USA may order ITSO books and CD-ROMs using PUBORDER. Customers in the USA may order by calling 1-800-879-2755 or by faxing 1-800-445-9269. Most major credit cards are accepted. Outside the USA, customers should contact their local IBM office. For guidance on ordering, send a PROFS note to BOOKSHOP at DKIBMVM1 or E-mail to bookshop@dk.ibm.com.

Customers may order hardcopy ITSO books individually or in customized sets, called BOFs, which relate to specific functions of interest. IBM employees and customers may also order ITSO books in online format on CD-ROM collections, which contain redbooks on a variety of products.

ITSO Redbooks on the World Wide Web (WWW)

Internet users may find information about redbooks on the ITSO World Wide Web home page. To access the ITSO Web pages, point your Web browser to the following URL:

<http://www.redbooks.ibm.com/redbooks>

IBM employees may access LIST3820s of redbooks as well. The internal Redbooks home page may be found at the following URL:

<http://w3.itsc.pok.ibm.com/redbooks/redbooks.html>

Acknowledgments

This project was designed and managed by:

Lajos Damen

International Technical Support Organization, Boca Raton Center

The author of this document is:

John Divers

IBM UK

This publication is the result of a residency conducted at the International Technical Support Organization, Boca Raton Center.

Thanks to the following people for the invaluable advice and guidance provided in the production of this document:

Harry Benas

Security Development, IBM Boca Raton

Graham Cogle

IBM UK

Bill Coltin

Security Development, IBM Boca Raton

Tony DiDaniele

Security Development, IBM Boca Raton

Mickey Galper

Security Development, IBM Boca Raton

Andy Low

IBM UK

Linda Paulat

Security Development, IBM Boca Raton

Bruce Scheer

Security Development, IBM Boca Raton

Doc Shankar

Security Development, IBM Boca Raton

Part 1. Developer's Guide

Chapter 1. Introduction to Security Enabling Services

About This Part

The OS/2 Security Enabling Services (SES) enables an Installable Security Subsystem (ISS) to provide C2-level operating system security services such as user identification and authentication, file/directory level access control and audit, and process level security context, etc.

This document is divided into three parts. This part deals mainly with the following:

- Background and overview information on Security Enabling Services (SES).
Discusses the concept of operating system security with an emphasis on customer requirements and the OS/2 security enabling strategy.
- Definition of operating system security
Looks at what operating system security means in an OS/2 environment and the role of Security Enabling Services (SES) in a secured OS/2 workstation.
- Brief details of C2 security requirements
Discusses the functional requirements needed to satisfy C2 and how Security Enabling Services (SES) is intended to assist an Installable Security Subsystem (ISS) meet these requirements.
- Architecture overview
Describes the contents of the OS/2 security enabling architecture, including Installable Security Subsystem (ISS) development and specific OS/2 security enabling services.
- Overview of an Installable Security Subsystem (ISS)
Discusses the type of development that would be required to provide an installable security subsystem that will make use of the security enabling services functions.

1.1 Prerequisite Knowledge

This document is intended for use by security application developers who are familiar with the following:

- Workstation security.
- Workstation programming.
- Basic AIX and OS/2 concepts (for example, shells, processes, and threads). These concepts may be discussed in this document but will not be defined here. Definitions of these concepts may be found in the appropriate product documentation.

A basic understanding of security enabling services may be obtained from the *(OS/2 Security Enabling Services redbook SG24-4568)*. An understanding of information security concepts, the need for workstation security, and the reason why certain security features are desirable, may be found in the *(Security Enhancements Solutions for Workstations redbook SG24-4569)*

For technical details about the OS/2 Security Enabling Services, including specifics about programming interfaces, refer to Part 2, “Design Notes” on page 79 of this document.

The appendices provide detailed information on the API calls available in security enabling services, and a summary of Security Enabling Services error codes referenced in this document. A glossary of the terminology used and an abbreviations list are also provided for convenience.

1.2 Chapter Breakdown

This developer’s guide section contains general information and points to the sections in the security enabling services document that provide more technical information.

The chapters in this part are:

- Chapter 1, “Introduction to Security Enabling Services”

This chapter provides an overview of this part of the document and some information about prerequisite knowledge that is required by developers.

- Chapter 2, “Security Requirements”

This chapter provides an overview of security requirements and looks at the criteria to satisfy C2 functional requirements.

- Chapter 3, “Security Enabling Services Overview”

This chapter provides an overview of security enabling services and what each of the components provides.

- Chapter 4, “Security Enabling Services”

This chapter provides more details of each of the security enabling services components and looks at the main points of each of the components.

- Chapter 5, “Installable Security Subsystem”

This chapter reviews what an installable security subsystem is and how security enabling services provides support for the installable security subsystem.

Chapter 2. Security Requirements

Protection of workstation resources is a primary concern of customers. Information that can be accessed through the workstations may be critical and vital; its loss, destruction or exposure may be devastating. Customers depend upon their workstations to provide support for the security policies that protect this vital information. This leads to customers requesting some basic security functions. Among the functions requested are the following:

- **Identification and Authentication (I&A)** Identification and authentication is the means by which a user is granted approved access to the system. There are the following three ways this requirement could be implemented:
 1. **Something the user knows.** An example of this type of implementation is a password. Passwords are the most commonly used identification and authentication mechanisms. They are the defaults for identification and authentication security for a large number of computer systems. A set of rules for password creation and utilization can be built into the system. Some systems even include password generators as a means of controlling the selection of passwords. The use of passwords has become prevalent in the security community. U.S. Department of Defense (DoD) guidelines for password usage have been published as part of the Rainbow Series (see the Department of Defense Password Management Guideline, also known as the Green book, for more information).

It's easy to design a password checking mechanism for a computer system. But there are problems associated with the use of passwords. In order to prevent someone from guessing a user's password, the password must be fairly complex. But if the password is too complex, the user may not be able to remember it. If a password is written down or stored in clear text, it may be read by the wrong person.
 2. **Something physical.** Tokens and smart cards are typical examples of this type of security implementation. These are physical devices, which may be inserted into a device reader, or may generate a special code that the user types into the computer system. An assumption is made that the user who physically controls these devices is the person who should have them. But since they are physical devices, it is possible that they may be stolen and used by the wrong person.

These devices are sometimes used as secondary authentication devices. For systems that use them as such, a password would probably be the first means of authentication. A user would have to enter a password, then use the device, in order to gain access to the system resources.

3. **Something unique.** This type of implementation is performed through the use of biometrics. Biometrics includes fingerprint analysis, retinal scans, voice signature analysis and keystroke analysis. An on-the-spot analysis of a biological characteristic of the user is performed when a request for system access is made. The new data is compared against a previously obtained biological pattern. This is a very effective means of identifying a user, but it is sometimes difficult to gain customer acceptance.

It's possible to blend any of the basic implementations mentioned above into different secure system combinations, depending upon the customer's needs.

- **Discretionary Access Control (DAC)**

System access based upon the identity of users or groups is the fundamental concept behind discretionary access control. Discretionary access control is typically implemented by access control lists or file permission bits, and can be applied at the discretion of the owner of the object being controlled. A file is an example of an object that can be controlled under discretionary access control.

- **Audit**

Audit is the ability to record significant system events. One of the most difficult tasks in designing an audit system is determining just exactly what needs to be audited. The minimum set of events that must be audited for system security include logon/logoff, file access and security policy violations. Other events may be audited at the discretion of the system designer, per specific customer requirements. Guidelines for audit collection may be found in a Rainbow series book called (*"A Guide to Understanding Audit in Trusted Systems"*) (known as the Tan book) released by the U.S. Department of Defense.

In addition to these requirements, there are other frequent customer requests. These requests are normally for facilities that will aid user productivity or counter a specific problem. The most common requests are for single signon and trusted program capabilities.

- **Single Signon**

Users who run multiple applications or who have access to multiple user accounts across networks frequently request a single signon capability. This allows the users to enter their identification and authentication data once and have access to all the system and network resources that they are allowed, rather than having to enter the identification and authentication data multiple times.

- **Trusted Program Support**

Programs that need privileged access to the security services or secured information controlled by the identification and authentication, discretionary access control, or audit mechanisms are called trusted programs.

The customer requirements for trusted programs may be driven by the following:

- The need to counter a specific perceived threat.
- The need to meet a predefined set of security requirements, such as those defined in the United States Department of Defense (*"Trusted Computer System Evaluation Criteria"*) (TCSEC or orange book). For example, the orange book describes identification and authentication, discretionary access control, audit, and object reuse protection requirements for C2 security.

Note: In addition to the trusted computer system evaluation criteria, customers may be concerned with the European Information Technology Security Evaluation Criteria (ITSEC), the Canadian Trusted Computer Product Evaluation Criteria (CTCPEC), or customer specific certification criteria. However, identification and authentication, discretionary access control and audit are basic requirements that are applicable to all the criteria.

Workstation operating systems can be designed to accommodate the customer's security requirements. A key concept in secure system design is the notion of a trusted computing base. The trusted computing base consists of all the code that must be trusted to enforce the security policies of the operating system. In general, the trusted computing base contains all the code that runs in the privileged state of the hardware, or any user level code that runs with special privileges. An application that runs with these special privileges is called a trusted (secure) program or application and is considered to be part of the trusted computing base. An example of a trusted application would be a secure Database Management System (DBMS).

The OS/2 base operating system doesn't supply specific functions which directly address the security requirements discussed above; it does, however provide support for what is called an installable security subsystem. The installable security subsystem, typically developed by an independent software vendor, contains specific security functions, which, when combined with the base OS/2 system, provide a secure OS/2 system.

2.1 Orange Book Security Criteria

Please note: This entire section was copied (with liberal paraphrasing and interpretation) from various sections of the Department of Defense ("*Trusted Computer Security Evaluation Criteria*") (TCSEC or orange book). The intent is to provide background information on C2 functional requirements, not to provide a precise definition of C2 security criteria. The focus is on C2 function identification and authentication, discretionary access control, audit, object reuse protection, but the criteria includes descriptions of related function from all classes/divisions (C2, B1). For example, a description of trusted path is included in this section because it's an important aspect of identification and authentication (C2 requirement), but trusted path is specified as a criteria for B2 in the trusted computer security evaluation criteria.

Any discussion of computer security necessarily starts from a statement of requirements, for example what it really means to call a computer system secure. In general, secure systems will control, through the use of specific features, access to information such that only properly authorized individuals, or processes operating on their behalf, will have access to read, write, create, or delete information. The following fundamental requirements are derived from this basic statement of objective:

Continuous Protection The trusted mechanisms that enforce these basic requirements must be continuously protected against tampering and/or unauthorized changes. No computer system can be considered truly secure if the basic hardware and software mechanisms that enforce the security policy are themselves subject to unauthorized modification or subversion.

Security Policy There must be an explicit and well-defined security policy enforced by the system. Given identified subjects and objects, there must be a set of rules that are used by the system to determine whether a given subject can be permitted to gain access to a

specific object. Discretionary security controls are required to ensure that only selected users or groups of users may obtain access to data (such as based on a need-to-know).

Identification

Individual subjects must be identified. Each access to information must be mediated based on the identity and authority of the person accessing the information. This identification and authorization information must be securely maintained by the computer system and be associated with every active element that performs some security-relevant action in the system.

Accountability

Audit information must be selectively kept and protected so that actions affecting security can be traced to the responsible party. A trusted system must be able to record the occurrences of security-relevant events in an audit log. The capability to select the audit events to be recorded is necessary to minimize the expense of auditing and to allow efficient analysis. Audit data must be protected from modification and unauthorized destruction to permit detection and after-the-fact investigations of security violations.

Assurance

The computer system must contain hardware/software mechanisms that can be independently evaluated to provide sufficient assurance that the system enforces the above requirements. In order to assure that the above requirements are enforced by a computer system, there must be some identified and unified collection of hardware and/or software controls that perform those functions. These mechanisms are typically embedded in the operating system and are designed to carry out the assigned tasks in a secure manner. The basis for trusting such system mechanisms in their operational setting must be clearly documented so that it is possible to independently examine the evidence to evaluate their sufficiency.

2.1.1 Continuous Protection

This section describes the basis for establishing and maintaining the integrity of the system.

2.1.1.1 Trusted Computing Base

The trusted computing base is the totality of protection mechanisms within a computer system, including hardware, firmware, and software; the combination of which is responsible for enforcing a security policy. A trusted computing base consists of one or more components that together enforce a unified security policy over a product or system. The ability of a trusted computing base to correctly enforce a security policy depends solely on the mechanisms within the trusted computing base and on the correct input by system administrative personnel of parameters (such as a user's credentials) related to the security policy.

In the interest of understandable and maintainable protection, the trusted computing base should be as simple as possible consistent with the functions it has to perform. Thus, the trusted computing base must be designed and implemented so that system elements excluded from it need not be trusted to maintain protection. The trusted computing base will necessarily include all those elements of the operating system and application software essential to the support of the security policies. Note that, as the amount of code in the trusted computing base increases, it becomes harder to be confident that the trusted computing base enforces the security policies under all circumstances.

2.1.1.2 Reference Monitor and Security Kernel

A reference monitor is an access control concept that refers to an abstract machine that mediates all accesses to objects by subjects. The reference monitor validates each reference to data or programs by any user (program) against a list of authorized types of reference for that user. The reference monitor must satisfy three design criteria:

1. It must be tamper proof.
2. It must be always be invoked.
3. It must be small enough to be subject to analysis and test, the completeness of which can be assured.

A security kernel is the hardware, firmware, and software elements of a trusted computing base that implement the reference monitor concept. It must mediate all accesses, be protected from modification, and be verifiable as correct.

2.1.2 Security Policy

In the most general sense, computer security is concerned with controlling the way in which a computer can be used, for example controlling how information processed by it can be accessed and manipulated. A trusted computer system that processes, stores, uses, or produces sensitive data must, with reasonable dependability, prevent the following:

- Deliberate or inadvertent access to protected data by unauthorized persons
- Unauthorized manipulation of the computer and its associated peripheral devices

The computer system protection requirements of an organization must be defined in terms of the perceived threats, risks, and goals of the organization. An organization's computer system security policy is a set of laws, rules, and practices that regulate how the organization manages, protects, and distributes sensitive information in its computer system(s).

A trusted computer system must satisfy the following security policy control objective:

A statement of the intent with regard to control over access to and dissemination of information, to be known as the security policy, must be precisely defined and implemented for each system that is used to protect sensitive information. The security policy must accurately reflect the laws, regulations, and general policies from which it is derived.

2.1.2.1 Discretionary Access Control

The term discretionary access control refers to a computer system's ability to control information on an individual basis. It stems from the requirement that each individual's access to information must be based on a demonstrated need-to-know.

A trusted computer system must satisfy the following discretionary security policy control objective:

Security policies defined for systems that are used to process classified or other sensitive information must include provisions for the enforcement of discretionary access control rules. That is, they must include a consistent set of rules for controlling and limiting access based on identified individuals who have been determined to have a need-to-know for the information.

Discretionary access control is a means of restricting access to objects based on the identity of subjects and/or groups to which they belong. Controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject.

Subject A subject is an active entity, generally in the form of a person, process, or device that causes information to flow among objects or changes the system state. Technically, a process/domain pair, where a domain is defined as the set of objects that a subject has the ability to access.

Object An object is a passive entity that contains or receives information. Access to an object potentially implies access to the information it contains. Examples of objects are: records, blocks, pages, segments, files, directories, directory trees, and programs, as well as bits, bytes, words, fields, processors, video displays, keyboard, clocks, printers, network nodes, etc.

The trusted computing base defines and controls access between named users and named objects (for example, files and programs) in the system. The enforcement mechanism (for example, access control lists) shall allow users to specify and control sharing of those objects, and shall provide controls to limit propagation of access rights. The discretionary access control mechanism, either by explicit user action or by default, provides that objects are protected from unauthorized access. These access controls are capable of specifying, for each named object, a list of named individuals and a list of groups of named individuals with their respective modes of access to that object. Furthermore, for each such named object, it is possible to specify a list of named individuals and a list of groups of named individuals for which no access to the object is given. Access permission to an object by users not already possessing access permission is only assigned by authorized users.

2.1.2.2 Object Reuse Protection

Object reuse is the reassignment of some subject to a storage medium (for example, page frame, disk sector, magnetic tape, etc.) that contains one or more objects. To be securely reassigned, such media must contain no residual data from the previously contained objects. The object reuse protection provided by the trusted computing base must remove residual data from media when objects on the media are reused.

All authorizations to the information that is contained within a storage object shall be revoked prior to initial assignment, allocation, or reallocation to a

subject from the trusted computing bases pool of unused storage objects. No information, including encrypted representations of information, produced by a prior subject's actions is to be available to any subject that obtains access to an object that has been released back to the system.

2.1.2.3 Least Privilege Operation

Least privilege operation is the principle that requires each subject in a system to be granted the most restrictive set of privileges needed for the performance of authorized tasks. The application of this principle limits the damage that can result from accident, error, or unauthorized use.

2.1.3 Accountability

Individual accountability is the key to securing and controlling any system that processes information on behalf of individuals or groups of individuals. The following requirements must be met in order to satisfy this objective:

1. The first requirement is for individual user identification.
2. Second, there is a need for authentication of the identification. Identification is functionally dependent on authentication. Without authentication, user identification has no credibility. Without a credible identity, discretionary security policies cannot be properly invoked because there is no assurance that proper authorizations can be made.
3. The third requirement is for dependable audit capabilities. That is, a trusted computer system must provide authorized personnel with the ability to audit any action that can potentially cause access to, generation of, or effect the release of sensitive information. The audit data will be selectively acquired based on the auditing needs of a particular installation and/or application. However, there must be sufficient granularity in the audit data to support tracing the auditable events to a specific individual who has taken the actions or on whose behalf the actions were taken.

A trusted computer system must satisfy the following accountability control objective:

Systems that are used to process or handle classified or other sensitive information must assure individual accountability whenever either a mandatory or discretionary security policy is invoked. Furthermore, to assure accountability the capability must exist for an authorized and competent agent to access and evaluate accountability information by a secure means, within a reasonable amount of time, and without undue difficulty.

2.1.3.1 Identification and Authentication

The trusted computing base requires users to identify themselves to it before beginning to perform any other actions that the trusted computing base is expected to mediate. Furthermore, the trusted computing base maintains authentication data that includes information for verifying the identity of individual users (for example, passwords) as well as information for determining the security credentials of individual users (for example, group membership, administrative roles). This data is used by the trusted computing base to authenticate the user's identity and to ensure that the security credentials of subjects external to the trusted computing base that may be created act on behalf of the individual user are dominated by the security to credentials of that user.

The trusted computing base must guarantee that its identification and authentication mechanisms cannot be circumvented by a user. The trusted computing base protects authentication data so that it cannot be accessed by any unauthorized user. The trusted computing base is able to enforce individual accountability by providing the capability to uniquely identify each individual system user. The trusted computing base shall also provide the capability of associating this identity with all auditable actions taken by that individual.

Trusted Path: A trusted path is a mechanism by which a person at a terminal can communicate directly with the trusted computing base. This mechanism can only be activated by the person or the trusted computing base and cannot be imitated by untrusted software.

The trusted computing base supports a trusted communication path between itself and users for use when a positive TCB-to-user connection is required (for example, logon). Communications via this trusted path are activated exclusively by a user or the trusted computing base and are logically isolated and unmistakably distinguishable from other paths.

2.1.3.2 Audit

The trusted computing base is able to create, maintain, and protect from modification or unauthorized access or destruction an audit trail of accesses to the objects it protects. The audit data is presented by the trusted computing base so that read access to it is limited to those who are authorized for audit data. The trusted computing base is able to record the following types of events: use of identification and authentication mechanisms, introduction of objects into an user's address space (such as file open, program initiation), deletion of objects, actions taken by computer

operators and system administrators and/or system security officers, and other security relevant events.

For each recorded event, the audit record identifies: date and time of the event, user, type of event, and success or failure of the event. For identification/authentication events the origin of request (for example, terminal ID) is included in the audit record. For events that introduce an object into a user's address space and for object deletion events the audit record includes the name of the object. The system administrator is able to selectively audit the actions of any one or more users based on individual identity. The trusted computing base contains a mechanism that is able to monitor the occurrence or accumulation of security auditable events that may indicate an imminent violation of security policy. This mechanism is able to immediately notify the security administrator when thresholds are exceeded and, if the occurrence or accumulation of these security relevant events continues, the system takes the least disruptive action to terminate the event.

2.1.4 Assurance

Assurance is concerned with guaranteeing or providing confidence that the security policy has been implemented correctly and that the protection-relevant elements of the system do, indeed, accurately mediate and enforce the intent of that policy. By extension, assurance must include a guarantee that the trusted elements of the system work only as intended.

A trusted computer system must satisfy the following assurance control objective:

Systems that are used to process or handle classified or other sensitive information must be designed to guarantee correct and accurate interpretation of the security policy and must not distort the intent of that policy. Assurance must be provided that correct implementation and operation of the policy exists throughout the system's life-cycle.

2.1.4.1 Operational Assurance

Operational assurance focuses on features and system architecture used to ensure that the security policy is uncircumventably enforced during system operation. That is, the security policy must be integrated into the hardware and software protection features of the system. Examples of steps taken to provide this kind of confidence include: methods for testing the operational hardware and software for correct operation, isolation of protection-critical code, and the use of hardware and software to provide distinct domains.

System Architecture: The trusted computing base maintains a domain for its own execution that protects it from external interference or tampering (for example, by modification of its code or data structures). The trusted computing base maintains process isolation through the provision of distinct address spaces under its control. The trusted computing base is internally structured into well-defined modules. It makes effective use of available hardware to separate those elements that are protection-critical from those that are not. Trusted computing base modules are designed such that the principle of least privilege operation is enforced. Features in hardware, such as segmentation, are used to support logically distinct storage objects with separate attributes (namely: readable, writeable). The user interface to the trusted computing base is completely defined and all elements of the trusted computing base identified. The trusted computing base is designed and structured to use a complete, conceptually simple protection mechanism with precisely defined semantics. This mechanism plays a central role in enforcing the internal structuring of the trusted computing base and the system. The trusted computing base incorporates significant use of layering, abstraction and data hiding. Significant system engineering is directed toward minimizing the complexity of the trusted computing base and excluding from the trusted computing base modules that are not protection-critical.

System Integrity: Hardware and/or software features are provided that can be used to periodically validate the correct operation of the on-site hardware and firmware elements of the trusted computing base.

Trusted Facility Management: The trusted computing base supports separate operator and administrator functions. The functions performed in the role of a security administrator are identified. The system administrative personnel is only able to perform security administrator functions after taking a distinct auditable action to assume the security administrator role on the system. Non-security functions that can be performed in the security administration role are limited strictly to those essential to performing the security role effectively.

Trusted Recovery: Procedures and/or mechanisms are provided to assure that, after a system failure or other discontinuity, recovery without a protection compromise is obtained.

Chapter 3. Security Enabling Services Overview

Before you can develop a security solution for OS/2, you must understand the components of the OS/2 Security Enabling Services (SES). You must understand how to exploit/modify Security Enabling Services components and what kind of support is available for your applications.

The diagram shown in Figure 1 on page 20, provides a broad look at the components of the OS/2 security enabling services. We will review each of the sections in this diagram to determine the function that each component provides in a secured environment and then we will be able to determine how the components relate to each other.

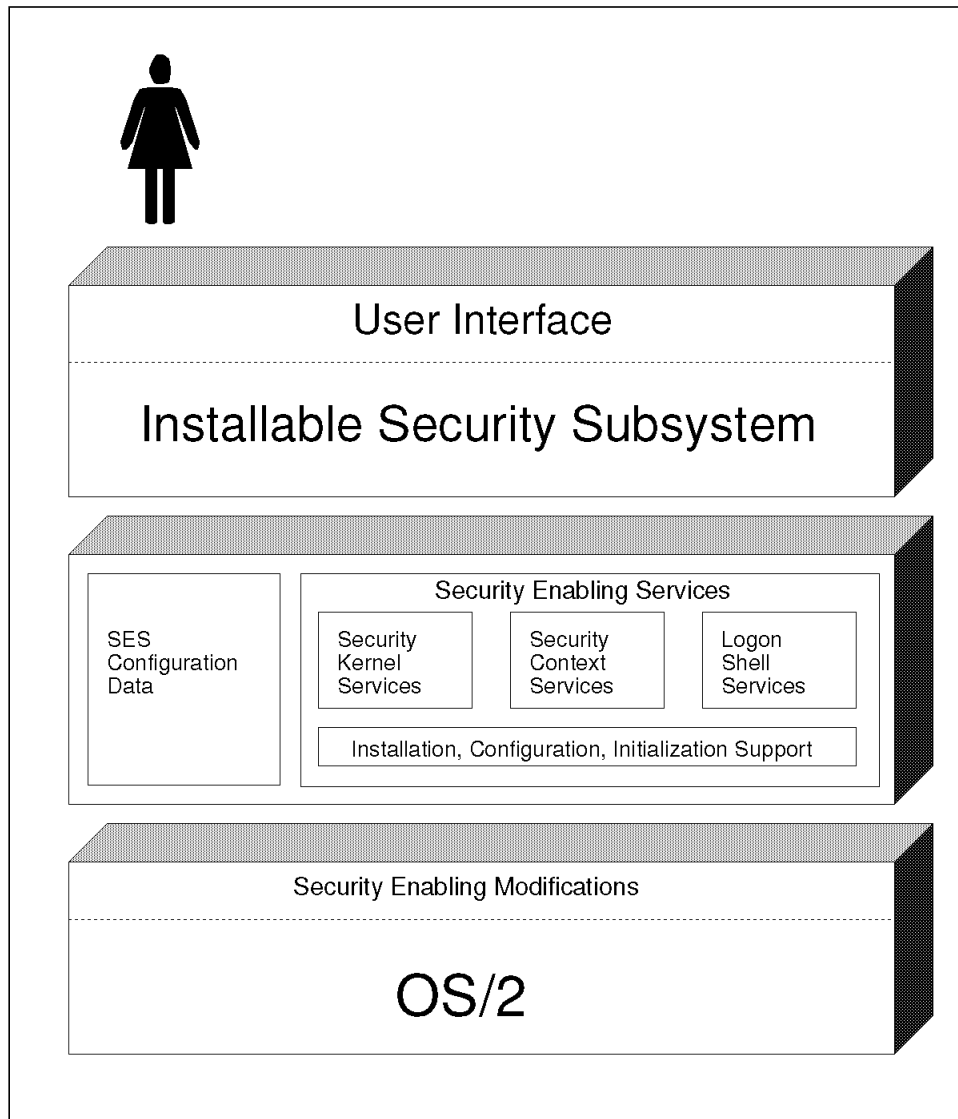
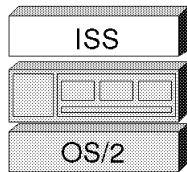


Figure 1. Components of a Secured OS/2 Workstation

3.1 Installable Security Subsystem (ISS)



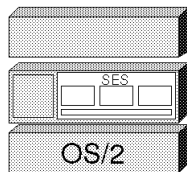
The installable security subsystem is a set of components that provide security features for a secured OS/2 operating system. This set of components will vary depending on the security features required by the customer, and may be developed by an independent software vendor, by IBM, or by the customer themselves.

The following are services that may be included as part of an installable security subsystem:

- User identification and authentication (logon)
- Resource access control
- Audit
- Security context and trusted program support
- Security policy administration tools

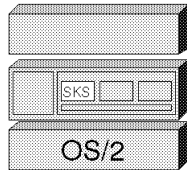
The components included in an installable security subsystem will depend on the security features that are required by the customer.

3.2 Security Enabling Services



Security enabling services enables an installable security subsystem to provide robust (C2 level) operating system security services (like RACF on VM or MVS).

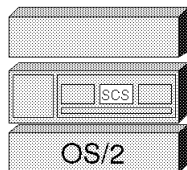
3.2.1 Security Kernel Services



The security kernel services is responsible for the following:

- Routing security relevant operating system events (such as file system access process creation) to an installable security subsystem to enable it to enforce security policies on those events
- Providing operating system services for an installable security subsystem security kernel that aren't normally available to a device driver running at Ring-0 (such as file open, read, write, etc.)

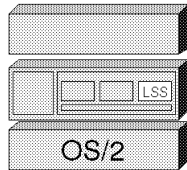
3.2.2 Security Context Services



The security control services allow the following:

- An installable security subsystem to establish an association between a process/thread and a subject handle, which the installable security subsystem can associate with a user's security credentials (such as user identity and group membership)
- An installable security subsystem to provide trusted program support (for example, a database application that can access the database file on behalf of a user even though the user is not authorized to access the database files directly)
- An installable security subsystem to provide concurrent multi-user access to an OS/2 workstation (for example, one local user and multiple remote TCP/IP users)

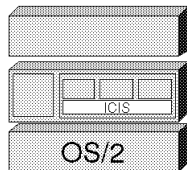
3.2.3 Logon Shell Services



The logon shell services enables the following:

- Alternative authentication mechanisms (such as smart card, finger print, etc.)
- The perception of a single signon by coordinating events that are relevant to a logon session (such as logon, logoff, lock, and unlock) with cooperating local and remote security components

3.2.4 Installation, Configuration, Initialization Support



The installation, configuration, initialization support enables the following:

- An installable security subsystem to provide secure installation, configuration, and system initialization (boot) features
- An installable security subsystem to provide trusted recovery services if the trusted computing base becomes corrupted

3.3 SES and ISS Communication

Communication between security enabling services and the installable security subsystem is accomplished through the following handles and programming interfaces:

Subject Handles: OS/2 is a multi-process user environment. Because of this, it is necessary to have a method to associate users with the process or threads they've started. This is accomplished in OS/2 by the use of subject handles, where a subject may be defined as a user, group, or process.

Subject handles associate user identifiers and other credentials with actively running OS/2 processes. When a user logs onto the system, a unique handle is created for the lifetime of the user's session. This handle, is associated with the user's name and password. Any appropriately privileged program can find the handle for a client process, and can in turn find the user name and password of the client. Furthermore, other security dependent applications running under OS/2 can associate their own notion of credentials with a handle. For example, a LAN file server application could associate LAN credentials with a user's handle.

Security Context Inheritance: With OS/2 security enabling services, the security context inheritance policy between parent and child processes is deliberately flexible. It is up to the Installable Security Subsystem to determine whether the child should inherit its parents authority.

The security enabling services system default inheritance policy is that child processes are untrusted, therefore children do not inherit authority from the parent process. In contrast, with the POSIX-compliant inheritance model, the child process does inherit the security context of the parent process. With OS/2, there is an option which may be specified to enable POSIX-compliant inheritance on a per-process basis. There is not a system-wide definition for this, it must be specified each time it is required otherwise the default inheritance policy will apply, that is no inheritance of authority.

Programming Interfaces: Installable security subsystem applications need to interact with predefined OS/2 services at an application and a kernel level. OS/2 provides programming interfaces for the following:

- Application components (API)
- Kernel level components (KPI)

Through the application programming interface and the kernel programming interface, an installable security subsystem can create, delete, reserve, and examine handles for processes and threads, control processes, wait for events, determine the order of execution for specific authorities, and receive kernel level event information.

3.3.1 Summary

OS/2 security is provided by an installable security subsystem, which takes advantage of the security services provided by security enabling services. The security elements required by an installable security subsystem will inevitably depend on the features required by the market, but the minimum set of components for any security subsystem would have to include the following:

- A security kernel that would enforce security policies at the OS/2 kernel level by interacting directly with the OS/2 kernel and security enabling services security context services.
- A security daemon with user identification and system logon authorities. The user identification authority would allow the identification and authentication of local system users, whereas the system logon authority would establish the security context for a local system logon.

Independent software vendors can create unique applications that provide identification and authentication, discretionary access control, audit, single signon, or trusted program functions for the OS/2 system. If required, these applications can be granted multiple privileges.

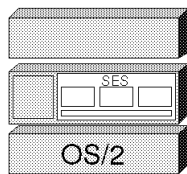
An installable security subsystem can include dynamic link libraries to replace/augment security enabling services policies. The system logon driver and client logon driver dynamic link libraries determine the order in which the user identification and client logon authorities are called by the system. The password validation driver dynamic link libraries validates password composition. The default security enabling services dynamic link libraries (for example, system logon driver, client logon driver, password logon driver) may be replaced by those supplied as part of an installable security subsystem.

User credentials and handles are accessible to the installable security subsystem. Programming interfaces, called APIs and KPIs are provided to enable the installable security subsystem to invoke the OS/2 security enabling services.

Chapter 4. Security Enabling Services

Only by utilizing the security enabling services of OS/2 can a truly robust security system be developed for OS/2. In this chapter, we take a close look at the security enabling services components, and how they interact with both the OS/2 kernel and the installable security subsystem.

4.1 SES Overview



OS/2 security requirements originate from a wide variety of customer environments ranging from the home and small office to large distributed computing environments that include a variety of client and server platforms.

The only truly secure way of dealing with these requirements and that which gives the customer the most choices is resource access control combined with an open security architecture. The resource access control is where security relevant events such as file open, print, connect to a COM port, are intercepted right down within the operating system. It is only when the operating system becomes involved that a true C2 level security system can be developed.

With the realization of an open architecture, the freedom to exploit these security services is given to independent software vendors and customers alike, so robust, consistent (cross platform), security solutions can be developed. With the development of the security enabling services interface for OS/2, IBM has delivered a resource access control level of security interface, combined with an open architecture.

The diagram in Figure 2 on page 28 illustrates the components that together make up a secured OS/2 operating system.

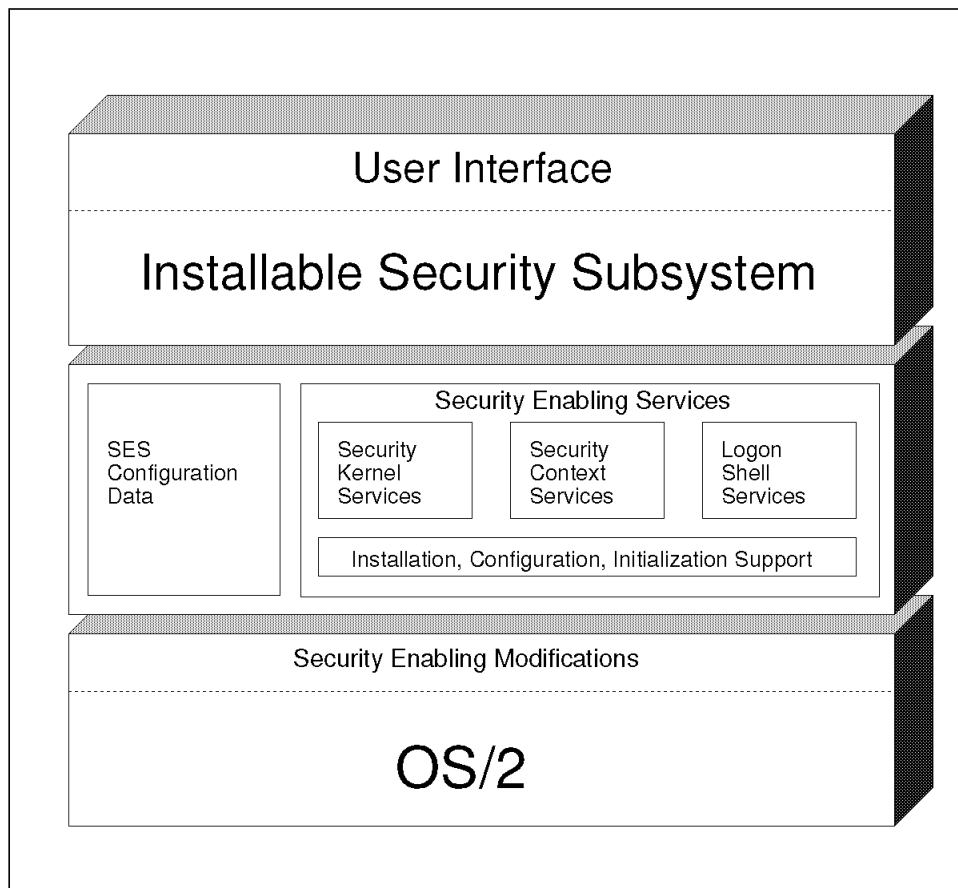


Figure 2. SES - Key Components of the OS/2 Enabling Strategy

- Security Kernel Services (SKS)

This routes security relevant events to the installable security subsystem, like file system access and process creation. It also provides operating system services to the installable security subsystem, which would not otherwise be available at Ring 0, such as file system access.

- Security Context Services (SCS)

This allows the installable security subsystem to associate a user's process, with that user's security credentials (user ID, group membership, trusted program privileges).

- Logon Shell Services (LSS)

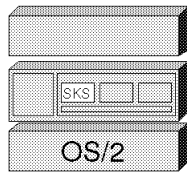
In order to allow the perception of single signon to the user, the logon shell services coordinates a number of components to log the user to the

local security subsystem and to other local/remote services. The authentication process can include the verification of smart cards, tokens, etc.

- **Installation, Configuration, Initialization Support (ICIS)**

Secure installation, configuration, and initialization (boot) of an OS/2 system with security enabling services, an installable security subsystem, and other security-related applications are handled by the installation, configuration, initialization support component. This ensures the integrity of the system at these critical times, while minimizing the impact on the standard OS/2 installation, configuration and installation processes.

4.2 Security Kernel Services (SKS)



The security kernel services supports the installable security subsystem by doing the following:

- Intercepting and routing security relevant OS/2 kernel events to the installable security subsystem security kernel.
- Providing kernel level operating system services for the installable security subsystem security kernel.

The diagram shown in Figure 3 on page 30, illustrates the concept of the hooks and services that security kernel services provides for an installable security subsystem, using the OS/2 file system services as an example.

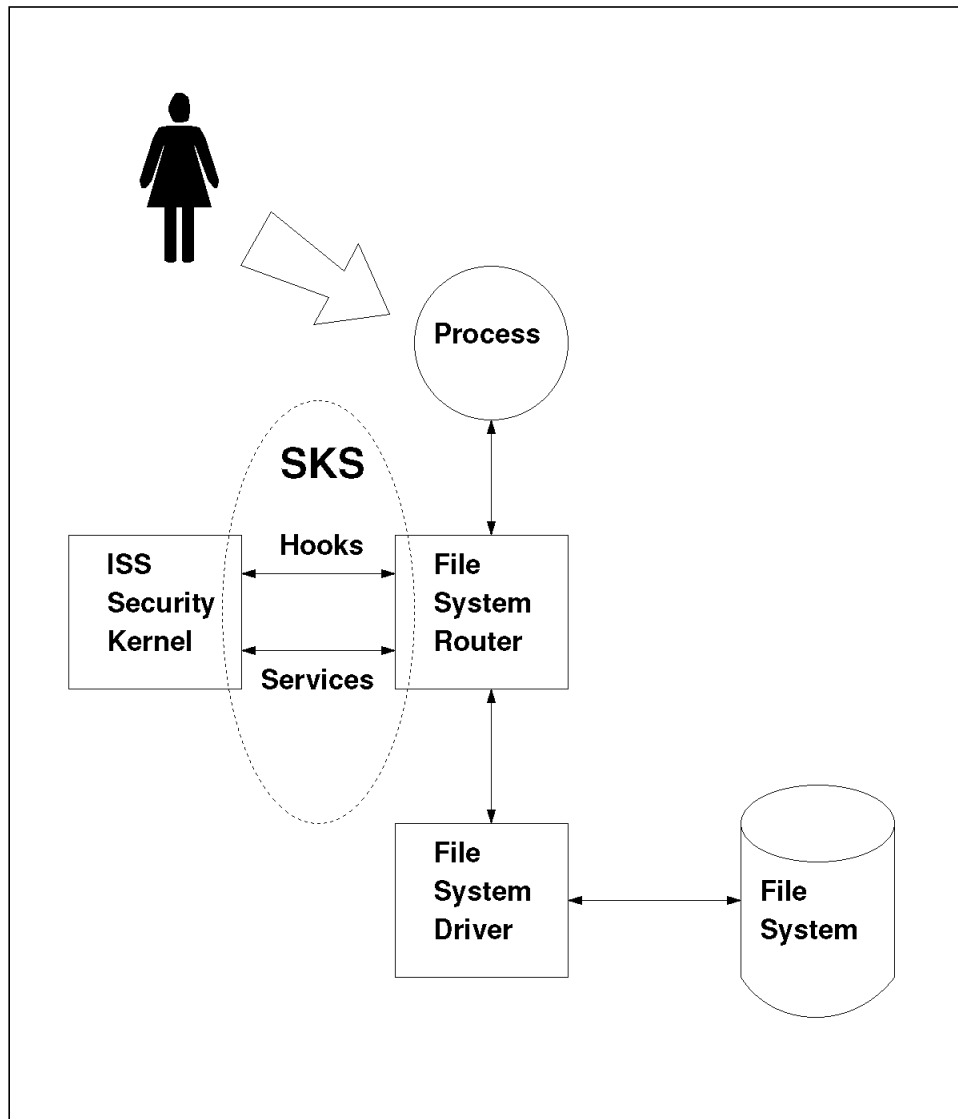


Figure 3. SKS - Hooks and Services for the ISS Security Kernel

4.2.1 Security Relevant Event Interception and Routing (Hooks)

The installable security subsystem security kernel device driver has to be notified of security relevant events so that it can enforce access control and audit policies. This is performed by hooks into the operating system so that these events can be notified to the installable security subsystem and then acted on appropriately. The events that can be hooked directly via the security kernel services are:

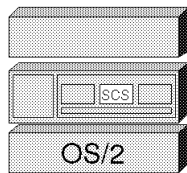
- File access (open, read, write, close, change file pointer, delete, etc.)
- Directory access (make, remove)
- Load DLL module
- Execute program
- Callgate level support
- Multiple virtual DOS machine support
- Logon shell services trusted path support
- Security enabling services API audit support

4.2.2 Kernel Level Operating System Services

The installable security subsystem security kernel device driver runs at Ring-0, but has to be able to invoke some OS/2 services that are normally only available at Ring-3, such as file access for audit log updates. To minimize the performance impact this would otherwise cause, the security kernel services provides the following services to the installable security subsystem security kernel:

- File system access (open, read, write, etc.)
- Security context services

4.3 Security Context Services (SCS)



The objective of security control services is to enable an installable security subsystem (and other security applications) to provide robust security

services, including enforcement of a discretionary access control security policy. A discretionary access control security policy is based on a model for controlling access rights to objects based on the identity of subjects, where:

- Object:** Is defined as a passive entity (for example, file, device) that contains or receives information. Access rights (for example, read, write, execute) to an object implies access rights to the information in the object for used by the subject as specified by the access rights.
- Subject:** Is defined as an active entity (OS/2 process) executing on behalf of a user/group. The subject is associated with user/group/process credentials (for example, user identification, group membership, trusted program privileges, etc.).
- Process:** Is defined as a program in execution, characterized by a single address space, execution state, and associated security context.
- Security Context:** Is defined as the information maintained for each process that enables security applications to associate the subject (for example, process executing on behalf of a user/group) with the appropriate user/group/process credentials.

In an operating system that doesn't support multiple concurrent processes, such as DOS and Windows, the single process is always acting on behalf of the one-and-only local system user. No other users or processes can share the system. However, even in a single process system, multiple concurrently active security applications may need to maintain different definitions of user/group/process credentials.

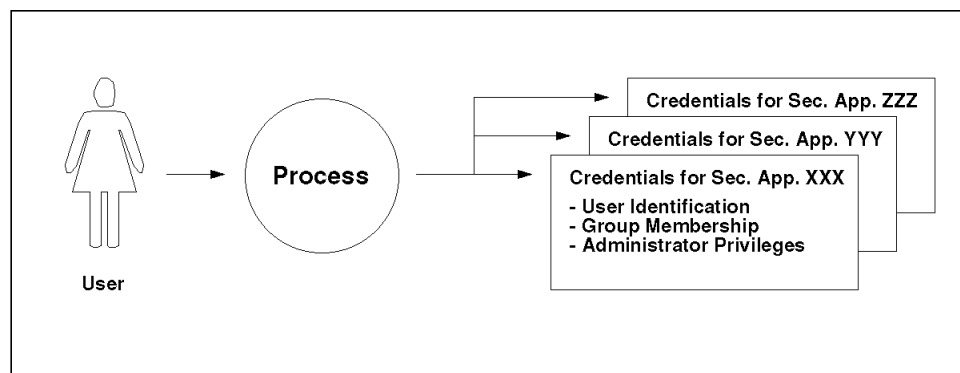


Figure 4. SCS - Single Process Model

4.3.1 Process, Handle and Thread Models

In an operating system that supports multiple concurrent processes, such as OS/2, the potential exists for these processes to be acting on behalf of different concurrently active users (and/or trusted programs with their own user/group/process credentials). And, again, multiple concurrently active security applications may need to maintain different definitions of user/group/process credentials.

In a single process operating system, it is possible for each security application to maintain its own user/group/process credentials for the one-and-only local system user without support from the operating system. However, in a multiple process operating system, where processes (and the associated security credentials) are dynamically created/terminated, the security applications need operating system support to manage the security credentials associated with each process.

To support dynamic management of user/group/process credentials associated with subjects (processes executing on behalf of users/groups) for multiple concurrently active security applications, security control services associates each process with a subject handle (dynamically-generated unique identifier) that each security application can associate with its own definition of the user/group/process credentials.

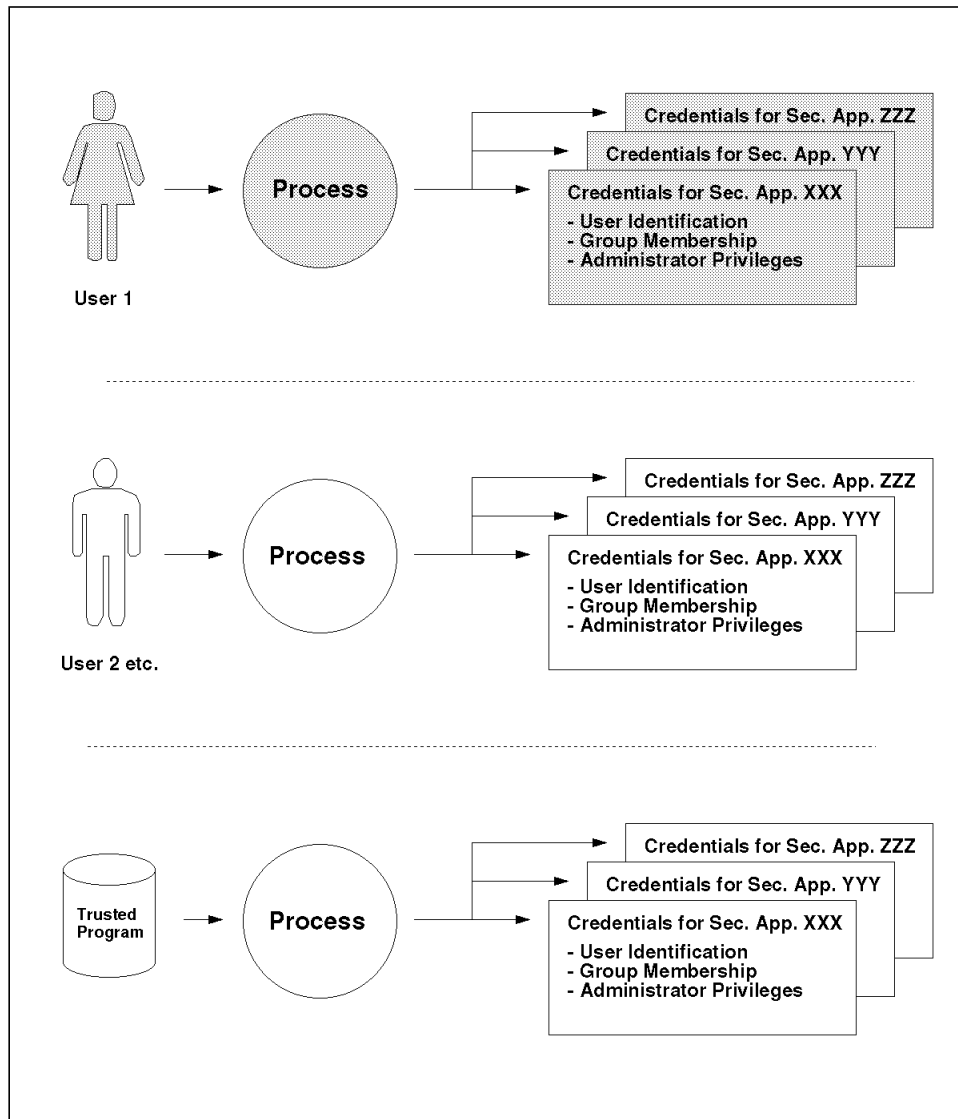


Figure 5. SCS - Multiple Process Model

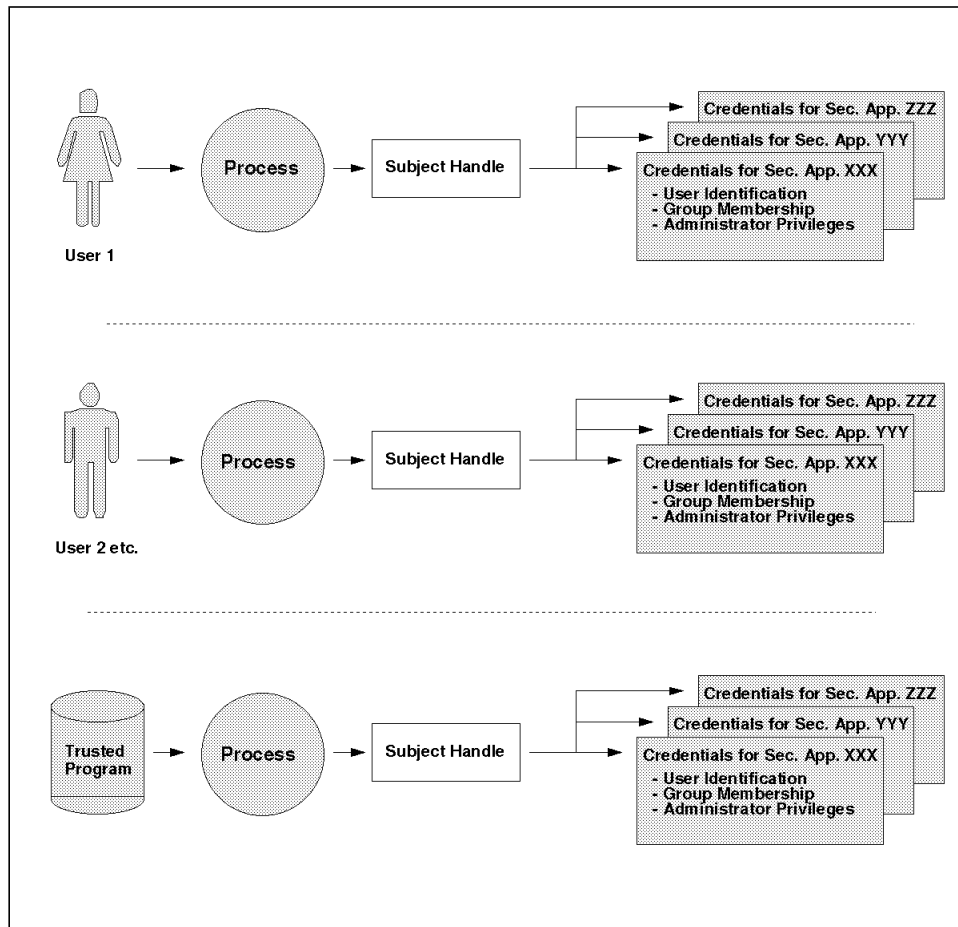


Figure 6. SCS - Subject Handle Model

To make the situation even more complicated, to support various trusted program/process models (for example, POSIX `setuid`, `setgid`, `umask`), each process potentially needs to be associated with the following sets of user/group/process credentials:

- Client and agent user credentials (for example, POSIX `uid`) to enable a trusted process to act on behalf of the real user who requested the process services (client user) and the saved user associated with the corresponding program (agent user).
- Client and agent group credentials (for example, POSIX `gid`) to enable a trusted process to act on behalf of the real group of the user who requested the process services (client group) and the saved group associated with the corresponding program (agent group).

- Client and agent process credentials (for example POSIX umask and distributed computing environment tickets) to enable a trusted server process to act on behalf of a multiple clients (client process) and itself (agent process).

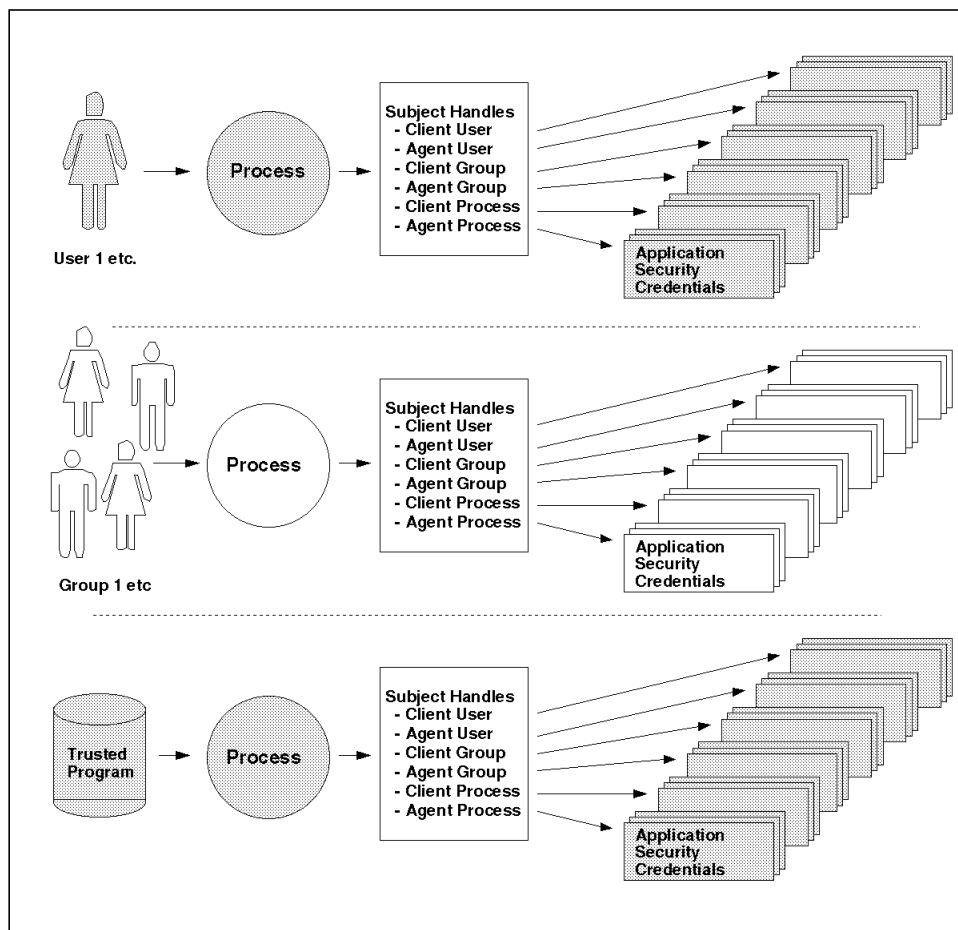


Figure 7. SCS - Trusted Process Model

To support either a process model (where all threads of a process share the same security context) or a thread model (where threads of a trusted process are allowed to maintain different security contexts), each process and each thread of a process potentially needs to be associated with different sets of user/group/process credentials.

However, even for the thread model, all threads of a process are executing the same program, and therefore have the same maximum security context

(although the threads can have different effective security contexts). In addition, an OS/2 process is represented by thread-1 (for example: if thread-1 dies, the process dies), so the effective security context of the process can be represented by the effective security context of thread-1.

Security control services therefore maintains the following:

- One maximum security context for each process
- One effective security context for thread-1 (which represents the process)
- One effective security context for each additional thread if the thread requests its own security context (that is, if it explicitly chooses a thread model instead of the default process model)

Please note that the vast majority of processes are untrusted and, consequently, all threads of the process will have the same effective security context as the maximum security context of the process. Only threads of a trusted process may have effective security contexts that are different from the maximum security context of the process.

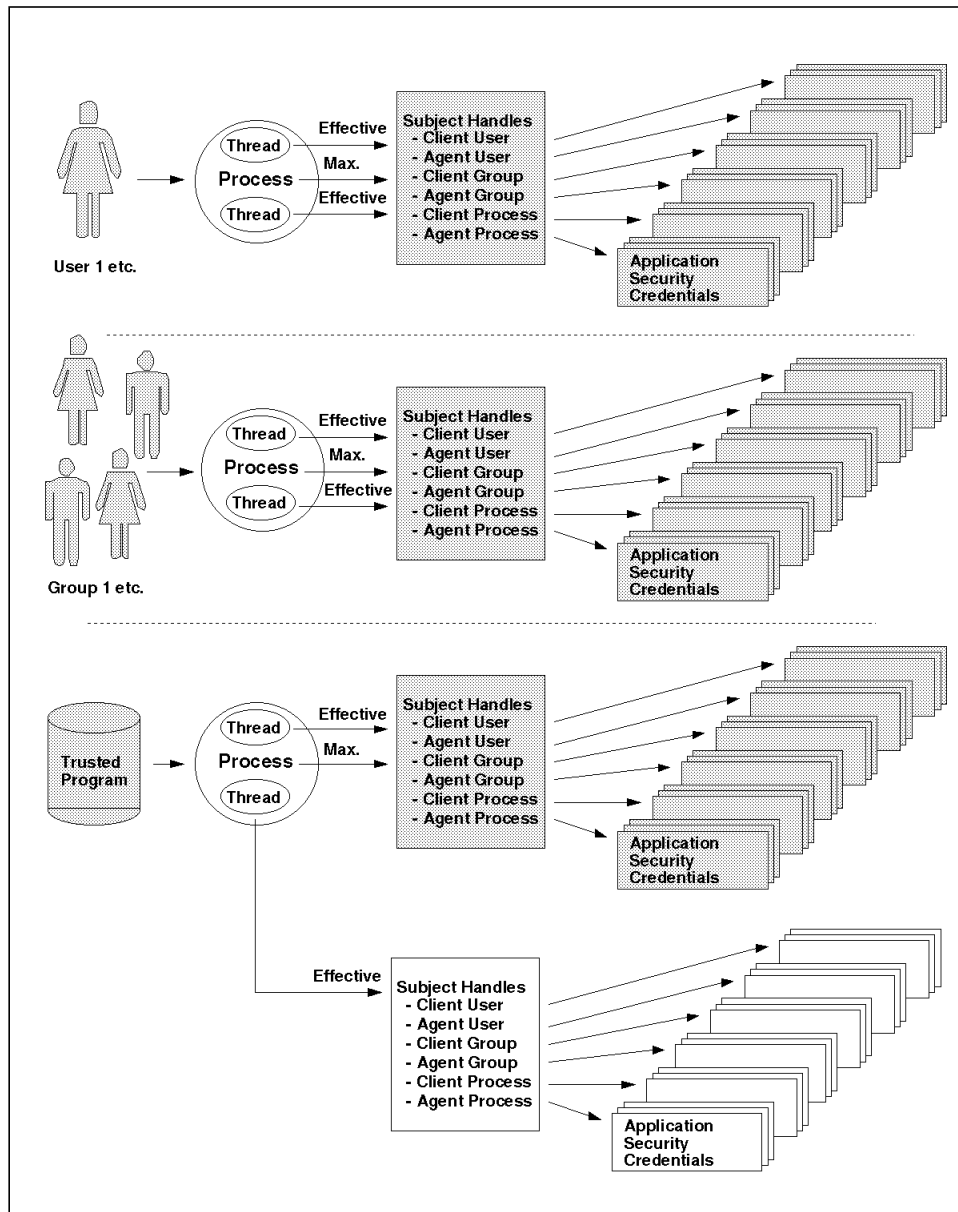


Figure 8. SCS - Thread Context Model

4.3.2 Multiple Concurrently Active Security Applications

Security control services must enable the interoperation of multiple concurrently active security applications. For example, a distributed database manager could be installed on an OS/2 workstation that is secured by an installable security subsystem. The database manager might need to be able to start processes that can access files protected by the installable security subsystem, and the installable security subsystem might need to be able to create processes that can access records in the database protected by the database manager.

To enable the interoperation of the installable security subsystem and other security applications, security control services must do the following:

- Enable each security application to maintain its own definition of the security credentials associated with a process/thread. This is accomplished by associating each process/thread with a set of handles that each security application can then associate with its own definition of security credentials.
- Allow each security application to invoke the security enabling services functions required to provide its security services (some security enabling services functions can only be invoked by trusted security applications). Trusted security applications are recognized by security control services as having the authority to invoke privileged Security Enabling Services functions by maintaining the authority status of the security application in the security context of the trusted process. Each special authority that a security application might need to invoke privileged security enabling services functions is associated with an authority flag in the security context of each process.

Note: Any process that has an authority flag set in its security context is generically referred to as a security context authority. A process with a specific authority flag set in its security context is referred to by that flag name plus Authority, for example, System Logon Authority (SLA), Access Control Authority (ACA), etc.

The services provided by security applications can be conceptually divided into the following groups of functions that require special security enabling services privileges (authority):

1. Establishing the association between a user's security credentials (for example, user identifier, group membership, administrative privileges) and the processes executing on behalf of the user. The first of these requires the identification and authentication of the user, and then the

association of the user's credentials with security context of the user's processes.

- System Logon Authority (SLA):

A process that has the authority to establish the process-user association for the local system user. The local system user is defined as the user who is associated with the OS/2 user interface services, that is Presentation Manager and Workplace Shell (PM/WPS).

- User Identification Authority (UIA):

A process that has the authority to identify and authenticate a user for local system logon (the association of the user's credentials with the OS/2 PM/WPS user interface services is accomplished by the system logon authority as described above).

- Remote Logon Authority (RLA):

A process that has the authority to establish the process-user association for a user who is not the local system user, that is a remote user who is accessing the local system through some interface other than the OS/2 PM/WPS user interface services, for example a user dialing in to the system through a TCP/IP connection.

Note: An Remote Logon Authority is also responsible for identifying and authenticating remote users, that is user identification authorities can only identify/authenticate users for local system logon through the system logon authority.

2. Enforcing resource access control and audit policies for processes executing on behalf of a user. These policies may be enforced for local objects/services and for remote objects/services.

- Access Control Authority (ACA):

A process that controls access to (typically local) resources/services based on the security context established by a system logon authority or remote logon authority.

- Client Logon Authority (CLA):

A process that controls access to (typically remote) resources/services based on its own authentication of a user, for example, Novell's client services that provide access to remote Novell servers. To enable the perception of single signon for the local system user, the client logon authority can access the authentication information (for example, user ID and password) provided by the user during local system logon.

3. Providing services for a client process that require the agent or server process to act on behalf of the client security credentials (in addition to acting on behalf of the agent/server security credentials).
 - Agent Process Authority (APA):

A process that can act on behalf of a client's security credentials and/or its own (typically more privileged) security credentials.
 - Server Process Authority (SPA):

A process with threads that can act on behalf of multiple different clients security credentials and/or its own (typically more privileged) security credentials.

4.3.3 Multiple Concurrently Active Users

Security control services must enable multiple concurrently active users. Although the primary target environment for OS/2 is the serial multi-user workstation with a single local keyboard-mouse-display, support for multiple concurrently active users is required in the following environments:

- Background (Detached) Program/Process

An installable security subsystem may want to allow some processes to continue executing on behalf of one user (in the background) while another user is logged on. For example, a print spooler could run as a background process executing on behalf of one user at the same time that another user is logged on.
- Trusted Program/Process

A trusted program/process can execute on behalf of a trusted user who is not necessarily the user who invoked the program. For example, a change password program can be invoked by an untrusted user but could execute on behalf of the trusted user who has the authority to update the password database.
- Multi-User Application Server

A multi-user application server can execute on behalf of multiple client users through a client/server protocol. For example, a distributed database manager, a file/print/application server, or a multi-user shell that supports multiple user I/O streams (such as an X-Protocol I/O application server).

The key design point for the multi-user support is associating each process with the user on whose behalf the process is executing security control services provides functions to associate a user (name) with a subject handle

and to associate a subject handle with a process. The following scenario describes how security control services functions could be used by an remote logon authority and an access control authority process to associate a user's credentials (for example, user identifier, group membership, administrative privileges) with the user's processes.

- The remote logon authority establishes an association between a user (name) and a unique subject handle. When the user's shell process is initiated, the remote logon authority establishes the association between the user's subject handle and the user's shell process. This association is inherited by all child processes of the user's shell process.
- When the remote logon authority establishes an association between a user (name) and subject handle, the access control authority can be notified of the association. The access control authority can then create the appropriate user credentials to be associated with the handle.
- When the access control authority intercepts a request for access to a protected object, the access control authority can retrieve the subject handle associated with the requesting process and use the associated user credentials to perform the access control check.
- When the last process referencing a subject handle is terminated, the access control authority can be notified. The access control authority can then delete the user credentials associated with the subject handle because it won't be used again during this system boot.

4.3.4 Trusted Program/Process

A trusted process can be defined as a process that has the authority to act on behalf of a user other than the user who invoked the process, that is the process has the authority to transition from executing with the privileges of one user to executing with the privileges of another user.

This privilege transition mechanism satisfies the following key requirements:

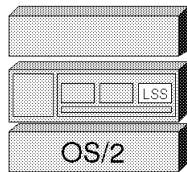
- A trusted process must be able to control access to private data. For example, a database manager could be invoked by any client user to update records in the database. The database manager might need to be able to execute on behalf of the client user, but might also need to execute on behalf of a trusted agent user that has the authority to update the database files (even though the user who invoked the database manager doesn't have the authority to modify the database files directly).
- A trusted server process must be able to impersonate client processes. For example, a multi-user application might need to be able to assume the security context of its client processes so that when it accesses

resources protected by an access control authority, the access control authority will enforce the access control policies for the client's user/group/process credentials (not the server's user/group/process credentials).

Security control services enables an installable security subsystem to implement trusted program support by associating each process/thread with client and agent user handles. The effective user handle can be set equal to either the client or agent user handle. In addition to supporting the association of each process/thread with client/agent user handles, security control services must also support the association of each process/thread with client/agent group handles and with client/agent process handles. The effective group/process handle can be set equal to either the client or agent group/process handle. Two security context authority roles are defined to satisfy the above requirements, without granting the trusted process unlimited super user powers:

- Agent Process Authority (APA):
Enables a trusted process to execute on behalf of an untrusted client (for example, the user who invoked the process) and a trusted agent (for example, the owner of the trusted program).
- Server Process Authority (SPA):
Enables a trusted process to have multiple threads that execute on behalf of different untrusted clients (who are not necessarily the user who invoked the process) and a trusted agent (for example, the owner of the trusted program).

4.4 Logon Shell Services (LSS)



A key requirement for security enabling services is to enable the perception of a single signon in customer environments where user resources may be stored on the user's local workstation and may also be stored on a variety of remote servers.

The local workstation resources may be protected by local workstation security services that require identification and authentication of the local workstation user, and each remote server may be protected by the server's security services that also require identification and authentication of client users. Users want to be able to enter their identification and authentication information (for example, name and password) one time, and have this information accepted by all of the local/remote identification and authentication mechanisms.

Logon shell services enables the perception of a local workstation user logging on to multiple local/remote services through a single signon event. Logon shell services accomplishes this by coordinating the interoperation of the various security components that need to participate in the logon event to perform identification and authentication of the local workstation user for access to the following:

- Local PM/WPS user interface services
- Local workstation resources protected by the ISS
- Other local/remote resources protected by other security services

In addition to enabling the perception of a single signon event, logon shell services enables the use of alternative authentication mechanisms (for example, smart cards, etc.) and coordinates the interoperation of security components for other events related to a logon session:

- Logoff, shutdown
- Lock, unlock
- Identification and authentication
- Change password, create user profile, delete user profile

Please note:

- Logon shell services does not provide generic single signon identification and authentication services itself, logon shell services simply enables cooperating components to work together to provide the perception of single signon.

Security enabling services does not do the following:

- Support distributed computing environment or GSSAPI logon services
- Include any user registry/database
- Include any authentication mechanisms
- Include any services to synchronize user IDs, passwords, etc.

- Include any facility to associate local users with security application credentials, so there is no logon notebook or personal logon facility to associate a local system user name with a remote server's domain/user ID
- Logon shell services does not provide generic logon services for multiple local/remote users. Logon shell services only provides logon services for the one and only local system user.

To emphasize this point:

- LSS functions can only be invoked for the local system user.
- LSS events depend on the state of the local system logon session.
- LSS event flows are integrated with PM/WPS services for the local system user.

Logon shell services functions can only be invoked by a process that can communicate with the local system user through PM/WPS user interface services. Security components participating in logon shell services events assume that they can communicate with the user who invoked a logon shell services function via Presentation Manager services (workplace shell may or may not be active depending on the state of an event).

For example, a security component that needs to authenticate a user for local system logon will most likely communicate with the user through Presentation Manager services (for example, through a dialog box). Consequently, logon shell services functions can only be invoked by a process that can communicate with the user through PM/WPS services. For example, a communications product that supports remote dial-in to an OS/2 workstation cannot invoke logon shell services functions for the remote user.

The following sections describe:

- The requirements of key security components that must cooperate to provide single signon support for the local system user.
- The operational requirements for interaction between these key components to provide an integrated state machine for events related to the local system user's logon session.

4.4.1 Overview of Key Logon Shell Services Components

Figure 9 identifies key security components that participate in LSS local logon session events:

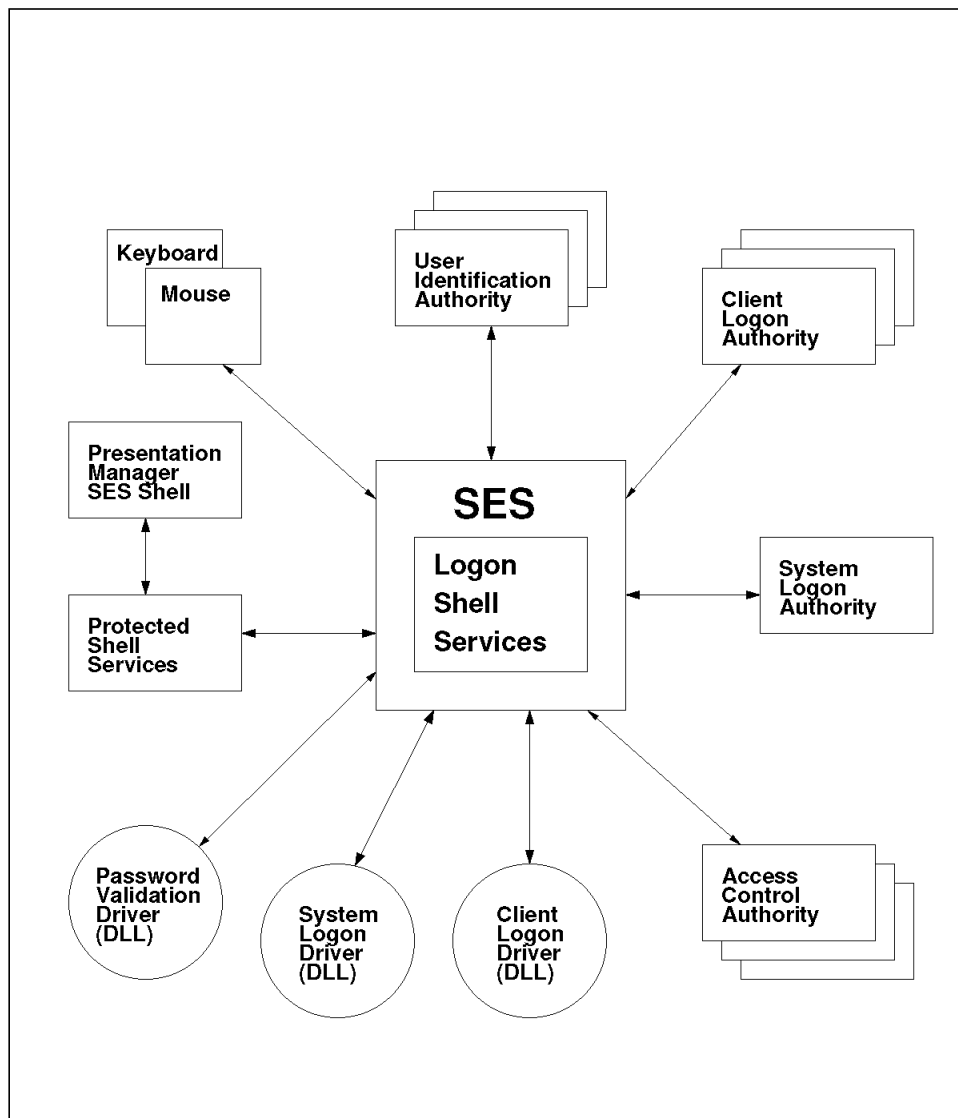


Figure 9. LSS - Coordination of Logon Session Events

4.4.1.1 Logon Shell Services Policy DLLs

To enable the perception of single signon in environments where multiple local/remote identification and authentication is required, these identification and authentication services can be divided into the following major functions:

1. Identification and authentication for access to resources on the local system (local system logon)
2. Identification and authentication for access to remote resources where the server doesn't accept local system identification and authentication (client logon)
3. Administration of user authentication information (for example userid and password)

The logon shell services policies for these functions are encapsulated in DLLs that can be replaced by independent software vendor security products or customers to satisfy specific security requirements.

System Logon Driver: The system logon driver defines/enforces the policy for security enabling services interaction with user identification authorities during a local system logon event (and other events related to a local system logon session), so the system logon driver determines which user identification authorities participate in identifying and authenticating a local system (PM/WPS) user. The system logon driver is called by security enabling services for the logon, unlock, identification and authentication, change password, create user profile and delete user profile operations.

Note: The default policy for the system logon driver supplied with security enabling services is to process the user identification authorities in the order they appear in the SECURE.SYS file, and to stop when the first user identification authority returns either success or failure (it will only go on to the next user identification authority if the current user identification authority returns an error condition).

Client Logon Driver: The client logon driver defines/enforces the policy for security enabling services interaction with client logon authorities during a local system logon event (and other events related to a local system logon session), so the client logon driver determines which client logon authorities participate in events related to a local system logon session. The client logon driver is invoked by security enabling services for the logon, logoff, lock, unlock, change password, create user profile and delete user profile operations.

Note: The default policy for the client logon driver supplied with security enabling services is to process all of the client logon authorities listed

in the SECURE.SYS file in the order they appear in the file. The return code is ignored.

Password Validation Driver: The password validation driver defines/enforces the policy for creating/changing a local system user's password. The password validation driver validates that a new password satisfies specified rules (for example, minimum length, composition rules, dictionary check, etc.). The password validation driver is invoked by security enabling services for the change password and create user profile operations.

Note: The default policy for the password validation driver shipped with security enabling services is to allow any password.

4.4.1.2 Logon Shell Services Event Daemons

An logon shell services event (for example local system logon) requires cooperation between security enabling services (the SES daemon, PSS daemon, SESShell daemon, and SES device driver) and the various security components that must participate in the event. For example, for a logon event, user identification authorities identify and authenticate the local system user; the system logon authority establishes the security context for the local system user; client logon authorities provide single signon services for the local system user; and security enabling services coordinates the whole event. To enable interaction between security enabling services and the cooperating security components, each security component must provide a daemon process that registers with security enabling services and waits to respond to logon shell services events when invoked by logon shell services. These daemon processes require special security enabling services privileges to participate in logon shell services events.

Note: No special privilege is required to start logon shell services events. For example, a smart card device could detect the insertion of a smart card and start a local system logon event without any special privileges. However, it would probably be better to include security applications (such as a smart card device that could provide identification and authentication for local workstation users) in the set of cooperating logon shell services components so that they are aware of the current state of the local system logon session and can act accordingly (for example, logon versus unlock).

User Identification Authority: Logon shell services interacts with user identification authorities to provide identification and authentication for the local workstation user for the logon, unlock, and identification and authentication events. Each user identification authority invoked returns the status of its authentication attempt (for example, success, failure, error). The

status from each user identification authority is processed by the system logon driver and the final user authentication status is determined by the system logon driver, which it returns to logon shell services.

System Logon Authority: Logon shell services interacts with the system logon authority to apply its security policy for logon, logoff, lock, unlock, and shutdown.

For logon and unlock, the system logon authority receives the final status determined by the system logon driver (based on the results of the UIAs that participated in the logon/unlock event). Given the results of the local identification and authentication, the system logon authority can do one of the following:

- Return a status to indicate that the logon/unlock event failed.
- Return a status to indicate that the logon/unlock event succeeded. If the event is a logon, the system logon authority would create the local security context for the user.
- Return a status to indicate a guest logon/unlock event. If the event is a logon, the system logon authority would create the local security context for the guest user.

For lock, logoff and shutdown events, the system logon authority first receives a query event notification. The system logon authority may prompt the user to confirm the event. The system logon authority may then cancel the event or allow lock/logoff/shutdown to continue. The system logon authority will, if the event was not cancelled, then receive a lock/logoff/shutdown event notification. At this point, the system logon authority will perform any processing required for the requested event, for example with logoff, all processes that are running on the users behalf will be terminated.

SLA One reason for the query logoff/shutdown event notification is to give the system logon authority an opportunity to confirm that the user wants to take this action before it's gone too far. Also, for the lock event, the system logon authority may not want a guest user to lock the system.

Client Logon Authority: After local system logon/unlock, client logon authorities are provided the name and password entered by the local user during the local system logon/unlock event.

Note: The password is not permanently stored anywhere. It is maintained in kernel memory during the local system logon session so that client

logon authorities may use it to provide the perception of single signon.

If names/passwords are synchronized between the local and remote systems, the client logon authority may be able to log the user on to the appropriate remote services using the local name/password without any further intervention by the user. If names/passwords are not synchronized, the client logon authority will need to obtain the correct name/password from the user.

Note: To expedite the logon process, client logon authorities should return status to logon shell services as soon as possible (before actually attempting to log the user on to a remote server). The intent is just to notify the client logon authority of the local system logon so that it can prepare to log the user on when needed (hopefully without any further intervention by the user).

4.4.1.3 LSS Keyboard/Mouse Device Driver Support

Logon shell services provides a trusted path service that enables a user to invoke the services of an installable security subsystem through a special key combination (for example, Ctrl-Alt-Del) that cannot be intercepted by applications. When the trusted path service is invoked, the installable security subsystem can take control over keyboard/mouse input and can ensure that the user's input is routed directly to the installable security subsystem.

When no local system user is logged on or the local system user interface services (PM and WPS) are in a locked state, a logon or unlock event must be initiated to start local system logon or to unlock the user interface services.

- If the trusted path service is not configured, logon shell services has control over the user keyboard/mouse input and initiates a logon/unlock event when user activity (keystroke or mouse button) is detected.
- If the trusted path service is configured, logon shell services will not initiate a logon/unlock event when user activity is detected. When a user invokes the trusted path service, the installable security subsystem can take control over keyboard/mouse input and can initiate a logon/unlock event as appropriate.
- Whether the trusted path service is configured or not, user activity other than keyboard/mouse input can be detected and a logon/unlock event can be initiated by security applications. For example, a smart card

reader could detect insertion of a smart card into the reader and initiate a logon/unlock event.

In addition to detecting trusted path invocation or user activity, logon shell services detects keyboard/mouse inactivity (keystroke or mouse button) to automatically lock the user interface services after a specified time period. This facility detects keyboard/mouse inactivity independent of the state of the user interface services (without regard to what screen group is active, etc.).

4.4.2 Overview of Key LSS Operations

Logon shell services supports the integration of multiple cooperating security components to handle local system logon session events (for example, logon, logoff, lock, unlock, etc.). The local system logon session refers to the time period between a successful logon event and a successful logoff event. During a local system logon session, the security context established for the local workstation user is associated with PM, the user shell (for example, WPS), and all untrusted processes created on behalf of the local workstation user. This security context will be referred to as the local system logon session security context.

The requirement to support the integration of multiple cooperating security components to handle local system logon session events involves a fairly complex state machine. In addition, logon shell services supports the following optional modes of operation:

- Trusted Path Support:
 1. Logon shell services monitors keyboard/mouse activity to initiate logon/unlock events.
 2. Installable security subsystem trusted path services initiate logon/unlock events (in lieu of logon shell services).

This optional mode of operation is specified by a CONFIG.SYS environment variable: TRUSTEDPATH=NO|YES (default is NO).

- User shell process handling for logon/logoff:
 1. Let the user shell continue to execute between local system logon sessions.
 2. Terminate/restart the user shell between local system logon sessions.

This optional mode of operation is specified by a CONFIG.SYS environment variable: RESTARTUSERSHELL=NO|YES (default is YES).

- Guest Logon Support:
 1. Requires user to explicitly initiate a guest logon event.
 2. Automatically starts a guest logon event (without explicit user action).

This optional mode of operation is specified by a CONFIG.SYS environment variable: AUTOGUEST=NO|YES (default is NO).

To describe how the requirement to support these optional modes of operation impacts the technical requirements for the logon shell services state machine, we need to loosely define the following four key LSS local system logon session states:

- Explicit Logon State

PM and the user shell process are active. PM and the user shell process are associated with the security context of the user (either an authenticated user or a guest user) for whom a local system logon event was explicitly initiated (although the user doesn't necessarily need to take an overt action to initiate logon).

For example:

- Logon shell services can initiate logon when keyboard/mouse activity is detected.
- The installable security subsystem can initiate logon as part of its trusted path services.
- A smart card device can initiate logon when a smart card is inserted.

Note that in this case, the system logon authority establishes the security context for the local system logon session. The security context associated with the local system logon session in the explicit logon state will be referred to as the explicit logon state security context.

- Auto guest Logon State

PM and the user shell process are active. PM and the user shell process are associated with the security context that is specified for the default unauthenticated user as the result of logon shell services automatically initiating an auto-guest logon event.

Note that in this case, the system logon authority does not establish the security context for the local system logon session. The security context associated with the local system logon session in the auto-guest logon state will be referred to as the auto-guest logon state security context.

- Lock State

PM and the user shell process are active, but the user interface services (for example, keyboard, mouse, display) are not available until an unlock event is initiated, that is keyboard/mouse input to applications is disabled and the display is covered with a customer-specified bitmap.

Note that in this case, the security context associated with the local system logon session is not changed from what was established during the local system logon event. Consequently, the security context associated with the local system logon session in the lock state is either the explicit logon auto security context or the auto-guest logon state security context.

- Logoff State

PM is active, the user shell may or may not be active (depending on the mode of operation), and the user interface services are not available until a logon event is initiated. PM and the user shell (if active) are associated with the security context that is specified for this state.

Note that in this case, the system logon authority may optionally define a security context for PM and the user shell (if active) when no local system user is logged on. The security context associated with PM and the user shell (if active) when no local system user is currently logged on will be referred to as the logoff state security context.

The diagram in Figure 10 on page 54, shows a simplistic view of how the logon shell services state machine transitions between these states as a result of logon shell services events.

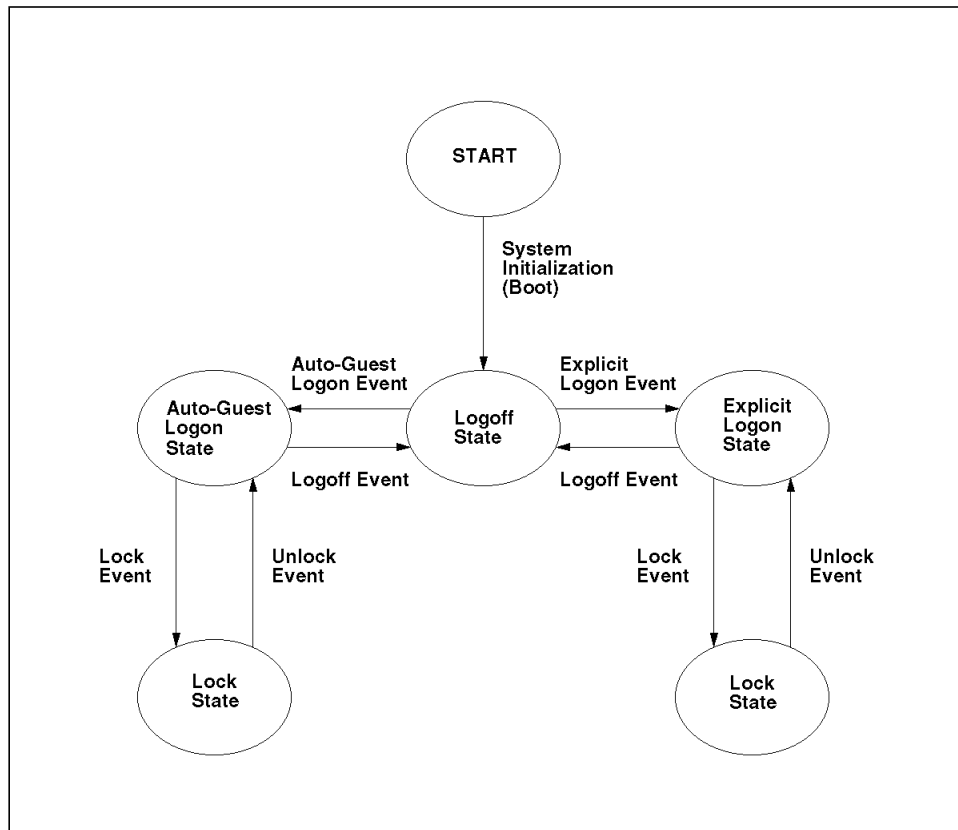


Figure 10. LSS - Overview of Logon Session Events

Please note there is no direct transition from the auto-guest logon state to the explicit logon state (or vice-versa). To transition from one logon state to another requires a logoff event followed by a logon event.

This may seem obvious, but it can cause some operational constraints that are not necessarily obvious (or desirable). The same is true for transitions between the logoff and lock states, that is direct transitions between these two states are not supported.

4.4.2.1 Initialization

System initialization, including initialization of security enabling services and security application components (for example, the installable security subsystem), is discussed in detail in the installation, configuration, initialization support architecture and system design sections.

Here we cover an overview of initialization to show the impact of the optional logon shell services modes of operation on system initialization requirements. In this section we will view the sequence of events that are undertaken for each setting.

- **DEFAULT OPERATION** (TrustedPath=No, RestartUserShell=Yes, AutoGuest=No)
 1. The security enabling services device driver and installable security subsystem security kernel (device driver) are loaded and initialized.
 2. The SES daemon is started.
 3. The SESShell daemon is started, which initializes PM.
Note: The SESShell daemon is the PM process.
 4. The PSS daemon is started.
 5. The SESShell daemon starts the security application daemons (for example: SLA, UIA, CLA) if configured.
 6. The initialization process is suspended until the system logon authority registers with security enabling services (and optionally specifies a logoff state security context). When the system logon authority registers with security enabling services, the initialization process is allowed to continue.
 7. The logon shell services state machine ends up in the logoff state, the user shell is not active and the user interface services are not available until a logon event is initiated.

- **TRUSTEDPATH=YES**

System initialization is not affected by this option.

- **RESTARTUSERSHELL=NO**

The user shell (for example: WPS) will be started during initialization, after security enabling services, installable security subsystem, PM, system logon authority are ready, but before a logon event. Relating this to our default operation listed earlier, we see:

1. No change.
2. No change.
3. No change.
4. No change.
5. No change.
6. No change.

7. Logon shell services starts the user shell.
 8. The logon shell services state machine ends up in the logoff state, the user shell is active, but the user interface services are not available until a logon event is initiated.
- AUTOGUEST=YES
- An auto-guest logon event will be initiated as the last step of initialization. Relating this to our default operation listed earlier, we see:
1. No change.
 2. No change.
 3. No change.
 4. No change.
 5. No change.
 6. No change.
 7. Logon shell services starts the user shell if RESTARTUSERSHELL=NO.
 8. Logon shell services initiates an auto-guest logon event (see next section for overview of logon).
 9. If the logon event is successful, the logon shell services state machine ends up in the auto-guest logon state.
- Note:** If the logon event is not successful, logon shell services ends up in the logoff state.

4.4.2.2 Logon/Logoff

Logon/logoff overview to show the impact of the optional logon shell services modes of operation on LSS state machine requirements.

- DEFAULT OPERATION (TrustedPath=No, RestartUserShell=Yes, AutoGuest=No)

Logon:

1. When the logon shell services state machine is in the logoff state and the user shell is not active, logon shell services detects keyboard/mouse activity and initiates an explicit logon event.
2. The client logon authorities are invoked (as determined by the system logon driver) to authenticate the local system user.
3. The system logon authority is invoked to establish a security context for the local system user.

4. If the logon event is continued by the system logon authority, the client logon authorities are invoked (as determined by the client logon driver) to provide single signon services and the user shell is started.

Note: If the logon event is not continued by the system logon authority, logon shell services returns to the logoff state.

5. The logon shell services state machine ends up in the explicit logon state.

Logoff:

1. When the logon shell services state machine is in the explicit logon state, a logoff event is initiated.
2. The system logon authority is invoked to determine whether the local system user wants to continue with the logoff event or not.
3. If the logoff event is continued by the system logon authority, the client logon authorities are invoked (as determined by the client logon driver) to provide single signon services and the user shell is terminated.

Note: If the logoff event is not continued by the system logon authority, logon shell services returns to the explicit logon state.

4. The logon shell services state machine ends up in the logoff state.

- TRUSTEDPATH=YES

Logon shell services will not initiate an explicit logon event when keyboard/mouse activity is detected while in the logoff state (the assumption being that the installable security subsystem trusted path services will initiate an explicit logon event as a result of trusted path invocation).

Logon:

1. When the logon shell services state machine is in the logoff state and the user shell is not active, logon shell services ignores keyboard/mouse activity. However, a trusted path invocation is detected and the installable security subsystem trusted path services initiate an explicit logon event.
2. No change.
3. No change.
4. No change.

5. No change.

The logoff operation is not affected by this option.

- **RESTARTUSERSHELL=NO**

Logon shell services will not terminate/restart the user shell between logoff/logon events.

Logon:

1. When the logon shell services state machine is in the logoff state and the user shell is active, an explicit logon event is initiated.
2. No change.
3. No change.
4. If the logon event is continued by the system logon authority, the client logon authorities are invoked (as determined by the client logon driver) to provide single signon services. The user shell is already active.

Note: If the logon event is not continued by the system logon authority, logon shell services returns to the logoff state.

5. No change.

Logoff:

1. No change.
2. No change.
3. If the logoff event is continued by the system logon authority, the client logon authorities are invoked (as determined by the client logon driver) to provide single signon services. The user shell is not terminated.

Note: If the logoff event is not continued by the system logon authority, logon shell services returns to the explicit logon state.

4. No change.

- **AUTOGUEST=YES**

Logon shell services will automatically start an auto-guest logon event (without explicit user action) whenever the logon shell services state machine is in a logoff state, with one key exception to allow a user to explicitly logon (either as an authenticated user or as an explicit guest user) when the logon shell services state machine is in the auto-guest logon state.

With this option, the logon shell services state machine is essentially always in a logon state (either auto-guest or explicit), except for the brief time between logon states while logon shell services transitions through the logoff state. Consequently, when a user wants to explicitly logon, a logoff event must be processed first.

To make this as painless as possible for the installable security subsystem, logon shell services handles the auto-guest logon state a little differently than the explicit logon state:

- Logon shell services allows an explicit logon event to be initiated while the logon shell services state machine is in the auto-guest logon state, and automatically initiates an implicit logoff event before proceeding with the explicit logon event.

Note: When the logon shell services state machine is in the auto-guest logon state, the installable security subsystem should provide a convenient user interface for the auto-guest user to initiate an explicit logon (for example when adding a logon icon to the desktop or adding a logon menu item to the desktop context menu).

- Logon shell services doesn't allow a logoff event while the logon shell services state machine is in the auto-guest logon state (since logon shell services would immediately start an auto-guest logon and return to auto-guest logon state), except for the implicit logoff initiated by logon shell services for an explicit logon.

Note: When the logon shell services state machine is in the auto-guest logon state, the installable security subsystem should not provide a user interface for the auto-guest user to initiate a logoff.

Logon:

1. When the logon shell services state machine is in the auto-guest logon state, an explicit logon event is initiated.
 - a. Logon shell services suspends the explicit logon event and initiates a logoff event.
 - b. When the logoff event is completed, logon shell services allows the explicit logon event to continue.
2. No change.
3. No change.
4. If the logon event is continued by the system logon authority, the client logon authorities are invoked (as determined by the client

logon driver) to provide single signon services. The user shell may already be active or may be started at this time, depending on the RESTARTUSERSHELL option.

Note: If the logon event is not continued by the system logon authority, logon shell services goes to the logoff state. However, an auto-guest logon is initiated immediately, so the logon shell services state machine ends up in the auto-guest logon state.

5. The logon shell services state machine ends up in the explicit logon state.

Logoff:

1. No change.
2. No change.
3. If the logoff event is continued by the system logon authority, the client logon authorities are invoked (as determined by the client logon driver) to provide single signon services. The user shell may or may not be terminated, depending on the RESTARTUSERSHELL option.

Note: If the logoff event is not continued by the system logon authority, logon shell services returns to the explicit logon state.

4. The logon shell services state machine goes to the logoff state, however, an auto-guest logon is initiated immediately, so the logon shell services state machine ends up in the auto-guest logon state.

4.4.2.3 Lock/Unlock

Lock/unlock overview to show the impact of the optional logon shell services modes of operation on logon shell services state machine requirements.

- DEFAULT OPERATION (TrustedPath=No, RestartUserShell=Yes, AutoGuest=No)

Lock:

1. When the logon shell services state machine is in a logon state (auto-guest or explicit), a lock event is initiated.
2. The system logon authority is invoked to determine whether the local system user wants to continue with the lock event or not.

3. If the lock event is continued by the system logon authority, the client logon authorities are invoked (as determined by the &cld) to provide single signon services.

Note: If the lock event is not continued by the system logon authority, logon shell services returns to the logon state.

4. The logon shell services state machine ends up in the lock state.

Unlock:

1. When the logon shell services state machine is in the lock state, logon shell services detects keyboard/mouse activity and initiates an unlock event.
2. The user identification authorities are invoked (as determined by the system logon driver) to re-authenticate the local system user.

Note: During the processing of an unlock event for a guest user (either auto-guest logon or explicit guest user logon), the unlock event cannot require authentication; however, the lock event can be used as a screen saver function for guest users.

3. The system logon authority is invoked to re-establish the security context for the local system user.
4. If the unlock event is continued by the system logon authority, the client logon authorities are invoked (as determined by the client logon driver) to provide single signon services.

Note: If the unlock event is not continued by the system logon authority, logon shell services returns to the lock state.

5. The logon shell services state machine ends up in a logon state (auto-guest or explicit).

- **TRUSTEDPATH=YES**

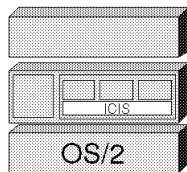
Logon shell services will not initiate an unlock event when keyboard/mouse activity is detected while in the lock state (the assumption being that the installable security subsystem trusted path services will initiate an unlock event as a result of trusted path invocation).

The lock operation is not affected by this option. Unlock:

1. When the logon shell services state machine is in the lock state, logon shell services ignores keyboard/mouse activity. However, a trusted path invocation is detected and the installable security subsystem trusted path services initiate an unlock event.

- 2. No change.
- 3. No change.
- 4. No change.
- 5. No change.
- RESTARTUSERSHELL
The lock/unlock operations are not affected by this option.
- AUTOGUEST The lock/unlock operations are not affected by this option.

4.5 Installation, Configuration and Initialization Support (ICIS)



The installation, configuration, initialization support support for security enabling services with OS/2 Version 2.11 and OS/2 Warp Version 3.00 is very straight forward:

- Installation

The security enabling services component for OS/2 Version 2.11 and OS/2 Warp Version 3.00 is distributed as part of the OS/2 service stream, and is available as follows:

- OS/2 Version 2.11 - fixpak XR_BSES.
- OS/2 Warp Version 3.00 - SECURITY.BBS

Because some of the OS/2 components have been modified to provide enhancements to support security enabling services, a pre-requisite fixpak level will be required for some releases of OS/2. The pre-requisite information is listed below:

- OS/2 Version 2.11 - fixpak XR_B100 or higher
- OS/2 Warp Version 3.00 - fixpak XR_W016 or higher

These fixpaks contain the first level of code to include the necessary security enabling services support modifications to the OS/2 base code. These modification are also included in later levels of the OS/2 code.

The security enabling services installation process simply copies the necessary files to the appropriate drive/directories. No modifications to CONFIG.SYS to enable security enabling services are made during the security enabling services installation process. When a customer installs a security product that requires security enabling services, the installable security subsystem installation process will make the appropriate modifications to CONFIG.SYS (and SECURE.SYS) to enable security enabling services features.

- Configuration

All security enabling services configuration is accomplished by modifying the CONFIG.SYS and SECURE.SYS files. No GUI or APIs are provided for configuration.

- Initialization

Independent software vendor security products typically provide boot protection which ensure continuous protection prior to OS/2 initialization. However, during OS/2 initialization (prior to the installable security subsystem being able to enforce its security policy), OS/2 ensures that the system is protected from unauthorized intervention. There are the following three phases of the initialization process that must be considered:

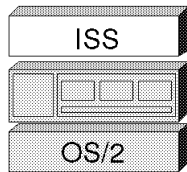
1. Prior to processing of CONFIG.SYS, the user can interrupt initialization by typing ALT-F1, the installable security subsystem must be able to ensure that this interruption cannot be used to circumvent its security policies.
2. During processing of CONFIG.SYS, security enabling services and installable security subsystem components must be loaded and functional prior to allowing a user to logon. If not, the installable security subsystem must be able to cause a default state where only an authorized administrator can intervene.
3. During processing of SECURE.SYS, security enabling services ensures that security applications are assigned authorized privileges and that no applications are assigned unauthorized privileges.

When OS/2 security is installed for Warp, a file called ALTF1SEC.CM is placed in the \OS2\BOOT directory. The independent software vendor product should rename this file to ALTF1SEC.CMD (or provide their own cmd file with the same name).

If ALT-F1 is to be disabled, it is possible to have a one line .CMD file with just an EXIT statement. When security is installed and a user hits ALT-F1, the

system will execute whatever is in ALTF1SEC.CMD, so the independent software vendors should be sure to protect this file.

Chapter 5. Installable Security Subsystem



An installable security subsystem is a set of components that provides the security features for a secured OS/2 system. In this chapter we discuss the relationship between security enabling services, an installable security subsystem and the security dependant applications that must work together in a secured OS/2 workstation. The diagram in Figure 11 on page 66 illustrates this relationship.

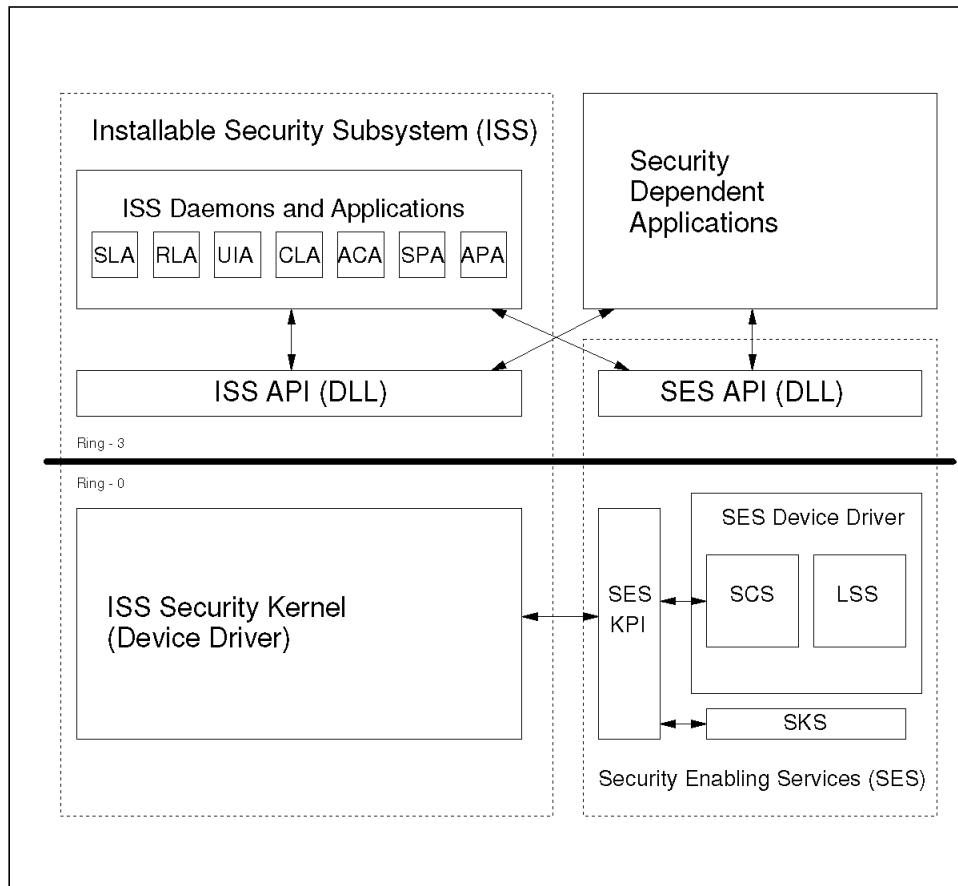


Figure 11. ISS - Overview of a Secured OS/2 System

5.1 What Is an Installable Security Subsystem?

An installable security subsystem is a set of components that provides the security features for a secured OS/2 system. An installable security subsystem may contain components that perform or support identification and authentication (such as password checking), DAC (such as file access control), system audit, single signon, trusted program support.

5.2 What Are the Typical Components of an ISS?

The components of an installable security subsystem will vary, depending upon the security features that the independent software vendor needs to add to the OS/2 system to satisfy the customer set. An installable security subsystem may include the components depicted in the diagram in Figure 12.

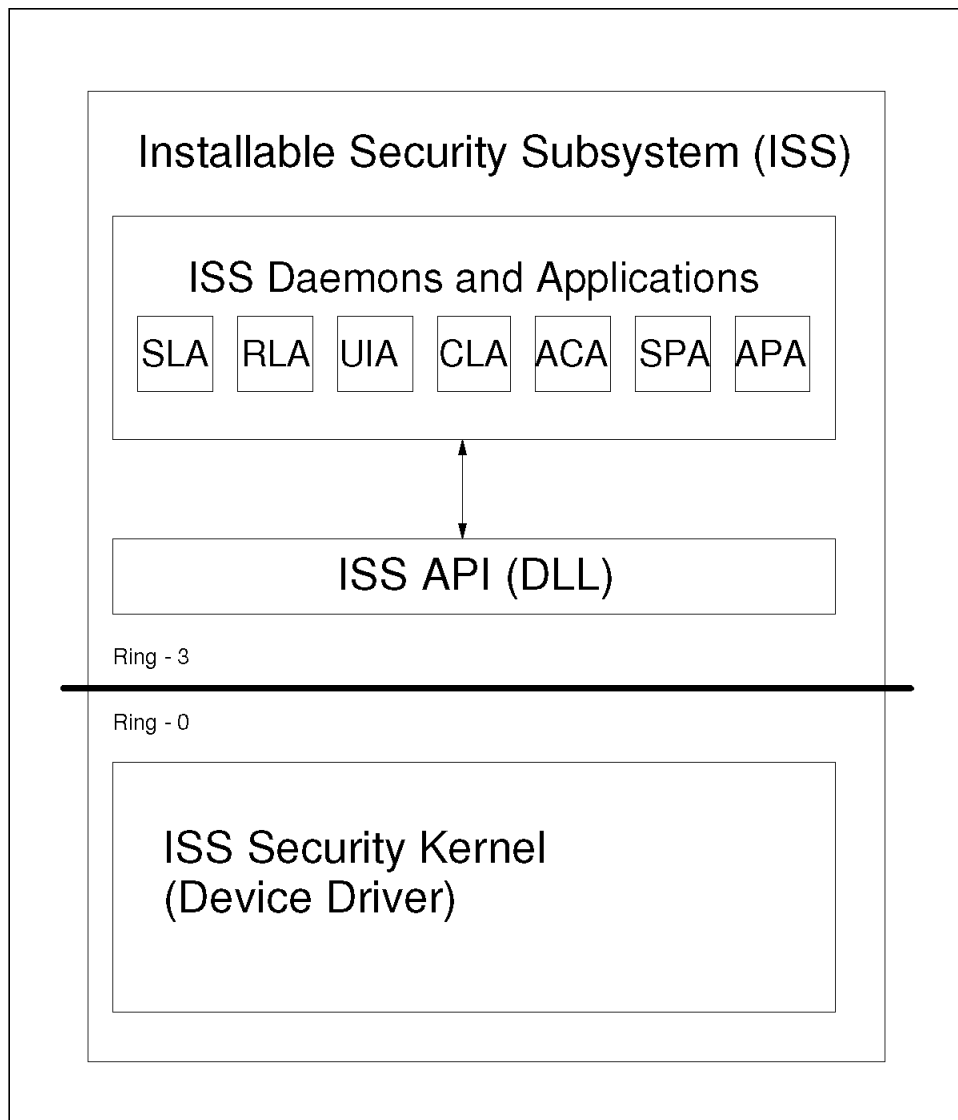


Figure 12. ISS - Components

We should take a closer look at the following components of an installable security subsystem (ISS).

- **Security daemons and applications**

The application level components of an installable security subsystem are what the customers see. This is the user interface and/or application program interface. By exploiting the security enabling services trusted process support, much of the logic required to implement installable security subsystem security features can be built as trusted installable security subsystem applications.

Note: From an OS/2 customer's point of view, there is no distinction between a security daemon and a security application, both are a set of one or more software programs/components that provide a service to the customer.

The security enabling services definition of a security daemon is an application/program that executes as a trusted process with special SES privileges and interacts with security enabling services via a captive thread that is blocked in the security enabling services device driver until security enabling services need the security daemon to respond to a security relevant event.

The installable security subsystem security daemons and applications may need special security enabling services privileges to perform their responsibilities. Table 1 shows which privileges might be applicable to the various services provided by an installable security subsystem.

Privileged ISS Daemon/Application	ISS service
Access Control Authority (ACA)	Resource Access Control
Agent Process Authority (APA)	Trusted Program Support
Client Logon Authority (CLA)	Client/Server Support
Remote Logon Authority (RLA)	Client/Server Support
System Logon Authority (SLA)	Local System Logon
Server Process Authority (SPA)	Trusted Program Support
User Identification Authority (UIA)	Identification and authentication

Table 1. Suggested Mapping of SES Privileges to ISS Functions

- **Security kernel (device driver)**

The function of the security kernel is to permit the installable security subsystem to establish communication with the OS/2 kernel. This enables the installable security subsystem to receive security event

information from the kernel. Without a security kernel (implemented as an OS/2 device driver), an installable security subsystem cannot enforce its security policy on OS/2 kernel operations.

Typical OS/2 kernel operations include file actions such as open, read, write, close, change file pointer, delete, process creation. These actions are referred to as security events. Notification of the occurrence of selected security events is sent from the OS/2 kernel to the ISS security kernel, if the installable security subsystem has indicated that it wishes to receive the notification.

- **Dynamic link libraries**

Three special DLLs are defined in the OS/2 security enabling services. Each has a well defined function in the processing of security events and each provides a default policy for processing security events. These DLLs, and the corresponding policy for processing security events, can be replaced by installable security subsystem DLLs:

- System Logon Driver (SLD)

The system logon driver determines the order in which user identification authorities are invoked during logon.

- Client Logon Driver (CLD)

The client logon driver determines the order in which client logon authorities are invoked during logon.

- Password Validation Driver (PVD)

The password validation driver verifies that a password issued during a change password request satisfies specified rules (composition, history, dictionary, etc.).

In addition, the installable security subsystem can provide its own APIs, with which a security dependent application can invoke the security functions provided by the installable security subsystem. This is an optional component of an installable security subsystem.

A trusted application is an external security component which depends upon the security services provided by the secure OS/2 operating system. It won't necessarily have been developed by the independent software vendor who developed the installable security subsystem, but it may need to use services provided by the installable security subsystem. Access to these services is provided through the APIs supplied with the installable security subsystem.

Why would an installable security subsystem want to provide APIs? By providing an API, a customer can develop security dependent

applications that can invoke the security services provided by the installable security subsystem. This adds significant value to an installable security subsystem for many large OS/2 customers.

5.3 What Support Does SES Provide for an ISS?

Security enabling services provides an operational environment for the installable security subsystem with well defined security services and support.

5.3.1 Security Context

Security enabling services maintains a security context for each OS/2 process/thread. The security context contains subject handles that are associated with user/group/process credentials and information about privileges and status of the process/thread. The important point to note here is that each process/thread is associated with a security context that denotes its privileges, status, credentials, etc.

The following are several ways the security context of a process/thread can be established/changed:

- Inherited from its parent process
- Specified as having special SES privileges
- Established by trusted components with appropriate SES privileges
- Modified through SES APIs
- Established by an ISS during system logon

Note: System logon is defined as associating a user's security context with the OS/2 user interface services (Presentation Manager and Workplace Shell).

5.3.2 Privileges and Authorities

A process that is active under OS/2 may be granted a specified set of security enabling services privileges. These privileges control the execution environment of the process and determine what security enabling services functions the process may access. Each privilege is defined by a separate flag in the process/thread security context, it is referred to as the security context authority (or authority for short). A program/process/thread that has one or more of these privileges is referred to as a security context authority.

Since the processes are identified by the authority roles they may assume, they are often referred to by the name of the authority. For example, a process that has assumed the role of access control may be referred to as

an access control authority. Processes may assume multiple authority roles, depending upon the functions they need to perform. The roles are assigned during system initialization, in response to information the independent software vendor adds to a system file named SECURE.SYS.

5.3.2.1 ACA, SLA and UIA

The ACA/SLA/UIA roles are defined primarily to support local workstation security services.

- **Access Control Authority (ACA)**

An access control authority establishes the rules for access to protected objects. It is invoked each time an initial access request is made. It must have access to the credentials of the requester and the access conditions for each protected object.

This is a means of implementing discretionary access control in OS/2. For example, an installable security subsystem may contain an access control authority which controls access to selected system files. The rules contained within the access control authority would control requester access based upon the user's credentials and the access control established for the file. The access control could be implemented in the form of an access control list and could restrict access to the file for a single user or a group of users.

The specific privilege defined for an access control authority is the right to register for notification of subject handle creation and deletion.

- **System Logon Authority (SLA)**

An system logon authority determines whether or not a user should be allowed to log on to the local system and helps create the credentials that associate the user with the local system logon environment. An installable security subsystem may define/manage the user/group/process credentials that are associated with the process security context (for example, set of subject handles, security context authority flags, and related status information). The system logon authority can associate the user with the processes executing for that user.

The minimum set of authorities that an installable security subsystem should have are user identification authority and system logon authority. With these authorities, an installable security subsystem can grant a user access to the OS/2 system (associate the user's credentials with the user's system logon session), and can grant access to authorized workstation resources. Note that the installable security subsystem

security daemon can be both a user identification authority and a system logon authority.

- **User Identification Authority (UIA)**

A user identification authority performs identification and authentication for users logging onto the system. The user identification authority takes the user logon information as input, and returns a status of user-authenticated or user-not-authenticated (or other error return codes) to security enabling services. An installable security subsystem should provide at least one authentication mechanism and corresponding user identification authority.

Note that there may be multiple user identification authorities in a system, depending upon a customer's needs. For example, a system that uses two forms of identification and authentication, such as a password and a smart card, might use two different user identification authorities to perform the identification and authentication. The order in which the user identification authorities are invoked to authenticate the user is defined in the system logon driver.

5.3.2.2 APA and SPA

The agent process authority and server process authority roles are defined for trusted program support, again primarily for local workstation security services.

- **Agent Process Authority (APA)**

An agent process authority executes on behalf of a client user (who may be untrusted) and a trusted agent. When an agent process authority program is invoked by a user, the agent process authority swaps the user's set of security privileges (called the security context) for the user's process to a set of security privileges which can perform a security sensitive operation. When the requested operation is completed, the agent process authority is terminated.

- **Server Process Authority (SPA)**

A server process authority executes on behalf of multiple client users and a trusted agent. A server process authority maintains one or more threads that run with the user's security context rather than the server process authorities. Each time a different user request is processed, the requester's security context is used to determine access rights.

Agent process authority and server process authority are provided in order to allow a program the capability of being able to transition from executing with the privileges of one user, to executing with the privileges of another

(pseudo) user. This allows a trusted program to execute on behalf of an untrusted client under certain controlled conditions. For example, a DBMS program might need to be accessible and process transactions on behalf of trusted and untrusted clients.

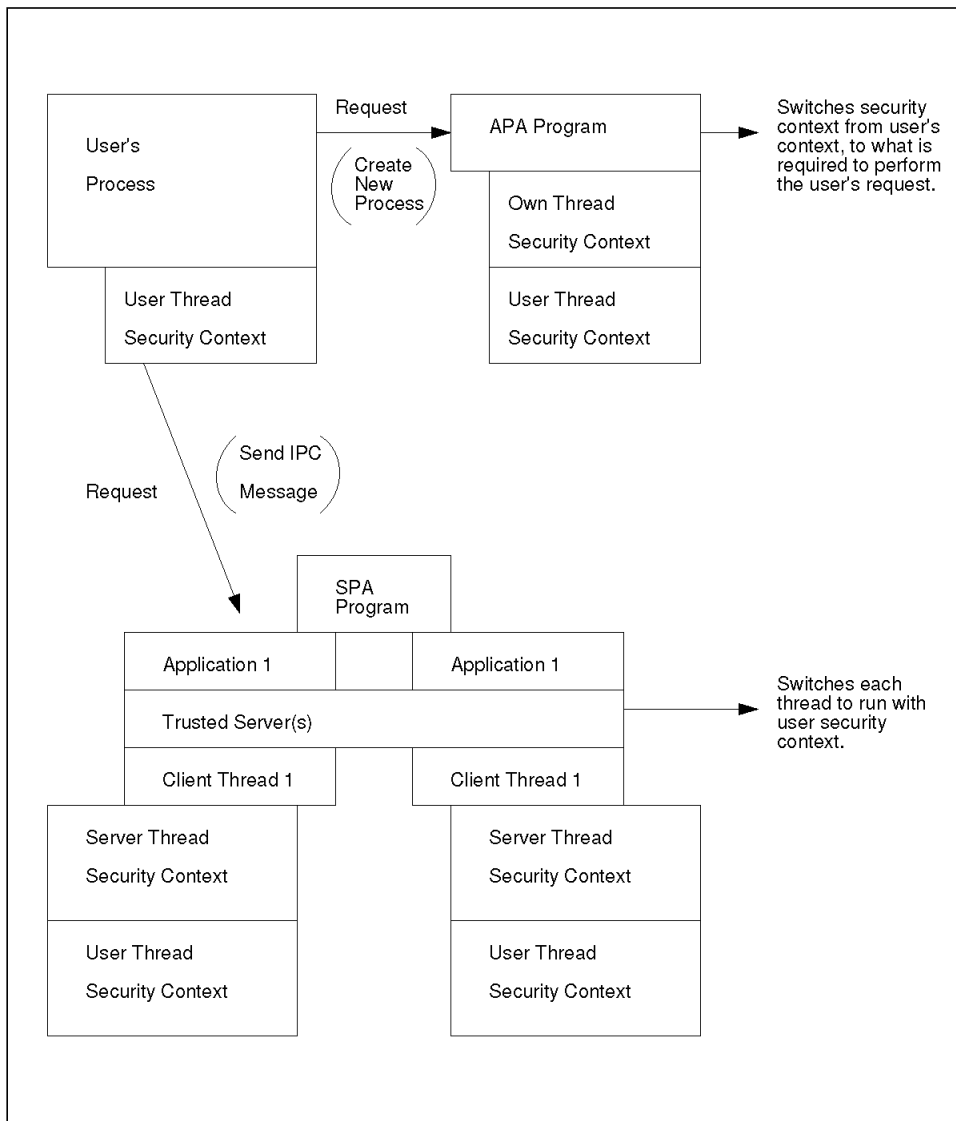


Figure 13. ISS - APA and SPA

5.3.2.3 Remote Logon and Client Logon Authorities

The remote logon authority and client logon authority roles are defined primarily to support distributed (client/server) computing environment security services.

- **Remote Logon Authority (RLA)**

Remote logon authority provides process-user association for remote system users and local system processes. For example, a remote user who wishes to log on to a local machine (perhaps through Telnet), would be authenticated to the local system by a remote logon authority. The functions of a system logon authority and a remote logon authority are similar. The actual association of the user with the processes executing for the user is performed by an remote logon authority for remote users.

- **Client Logon Authority (CLA)**

A client logon authority authenticates the local user onto a remote system. A client logon authority gathers up whatever information is necessary and propagates it to some remote node/server for authentication. If the remote authentication fails, the user may still be locally authenticated. Depending upon the installable security subsystem application, a client logon authority could also have user identification authority authority. There may be multiple client logon authorities in a system; the client logon driver determines the order in which they are called.

A client logon authority is not needed on a stand-alone workstation, but should be present on client workstations in a client/server environment to facilitate the perception of single signon during the local system logon process.

5.3.2.4 Interoperation of Security Context Authorities

How do the security context authorities (UIA, SLA, CLA, RLA, ACA) work together? Consider what happens when a user initiates a logon request.

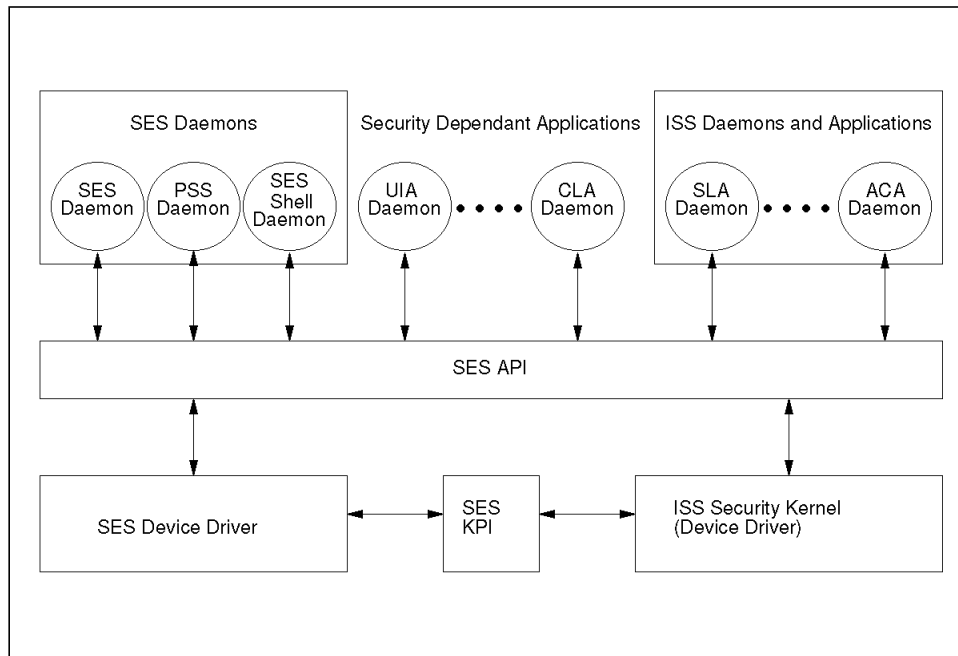


Figure 14. ISS - Interoperation of Security Context Authorities

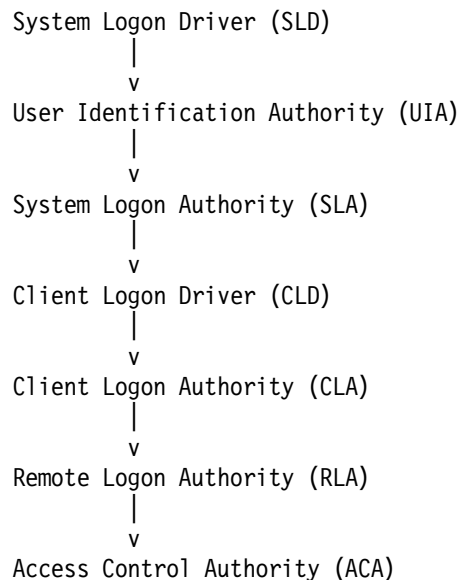
After the logon request is made by the user, the system logon driver is queried to determine the order in which to notify system user identification authorities. The user identification authority obtains the user name and password (any identification and authentication mechanism could be used), and performs authentication of the user. The information is passed back to the system, and the system logon authority is notified of the logon attempt. The system logon authority applies the logon policy that has been defined by the independent software vendor, and determines if system logon is to be performed. If it is, security enabling services creates the security context (as defined by the system logon authority) to associate with the user's credentials for the local system logon environment (typically PM and WPS).

The client logon driver is queried to determine the order in which to call client logon authorities. The appropriate client logon authority obtains the user logon information and propagates it to a remote resource for authentication and subsequent access to the remote resources. If authentication is successful, the user obtains access to the remote services.

The remote logon authority receives a request to logon from a remote server, and performs authentication as needed. If authentication is successful, security enabling services creates the security context (as

specified by the remote logon authority) to associate the user's credentials with the process tree(s) created for the user by the remote logon authority. When an access control authority receives a request for access to protected resources from one of these processes, the access control authority can query the security context for the requesting process and apply the appropriate access control policy.

For example, if we look at a user performing a local logon but also making use of the single signon facility, the logic flow would be similar to this:



5.3.3 Programming Interfaces

Installable security subsystem applications must interact with predefined OS/2 services at an application and a kernel level. OS/2 provides programming interfaces for application (API) and kernel (KPI) level components. The API provides security developers a means to create security applications; the KPI provides security developers a means to create installable security subsystem device drivers. For example, the installable security subsystem security kernel (device driver) communicates through defined KPIs to the kernel level security enabling services:

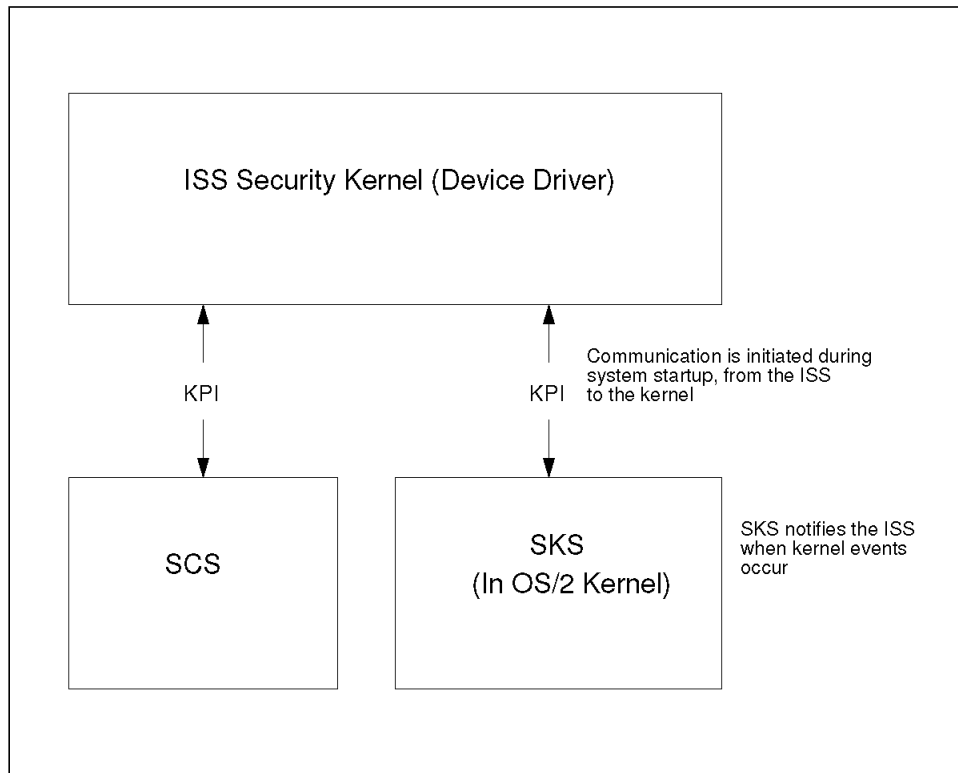


Figure 15. ISS - Kernel Programming Interface

Through the APIs and the KPIs, an installable security subsystem can create, delete, reserve, and examine handles for processes and threads, control processes, wait for events, determine the order of execution for specific authorities, and receive kernel level event information. APIs are made available to independent software vendors as dynamic link libraries, which are loaded into the installable security subsystem through C language calls.

5.4 Installable Security Subsystem Summary

OS/2 security is provided by an installable security subsystem, building on the services provided by OS/2 security enabling services. The minimum set of components necessary for an installable security subsystem depends upon the services provided by the installable security subsystem, but would typically include the following:

- A security kernel (to enforce security policies at the OS/2 kernel level by interacting directly with the OS/2 kernel and security enabling services security context services).
- A security daemon with user identification authority and system logon authority privileges (user identification authority to identify and authenticate local system users, system logon authority to establish the security context for local system logon), perhaps also with access control authority privileges (for notification of subject handle creation/deletion).

Independent software vendors can create unique applications that provide identification and authentication, discretionary access control, audit, single signon, or trusted program functions for the OS/2 system. These applications can be granted a defined set of privileges (ACA, SLA, UIA, RLA, CLA, SPA and APA). An application may be granted multiple privileges.

An installable security subsystem can include DLLs to replace/augment security enabling services policies. The system logon driver and client logon driver DLLs determine the order in which user identification authorities and client logon authorities are called by the system. The password validation driver DLL validates password composition. The default security enabling services DLLs (SLD, CLD and PVD) may be replaced by an installable security subsystem.

User credentials and handles are accessible to the installable security subsystem. Programming interfaces, called APIs and KPIs are provided to enable the installable security subsystem to invoke the OS/2 security enabling services.

Part 2. Design Notes

Chapter 6. Introduction to SES Development

About This Part

Part 1, “Developer’s Guide” on page 1 provides some background information and introduces terminology that would help make reading this part of the document a little easier:

- Overview of the OS/2 security enabling strategy
- Definition of operating system security
- Description of C2 security requirements
- Role of SES in a secured OS/2 workstation

This section describes the requirements that influence the design of security enabling services, discusses the overall security enabling services architecture and key components, and explains the high-level system design for security enabling services features.

- Architecture

Discusses key security enabling services components, the interaction among these components, and key design concepts.

- System Design

Provides an overview of the security enabling services design, including the general flow of control/data for security enabling services functions and scenarios describing the intended use of the functions.

- Building Blocks

Describes the application/kernel programming interfaces, trusted program privileges, and dynamic link libraries that are applicable to various security features.

- Design Guidelines

Contains an action list to use when building an installable security subsystem and information on what a developer should produce.

- Programming Interfaces

Discusses the programming interfaces available in security enabling services.

6.1 Chapter Breakdown

This design notes section contains details of the security enabling services design and architecture, provides more technical information on each of the security enabling services components and discusses how a developer would go about producing an installable security subsystem.

The chapters in this part are:

- Chapter 6, “Introduction to SES Development”
This chapter provides an overview of this part of the document.
- Chapter 7, “Building an Installable Security Subsystem”
This chapter provides detail on how a developer would build an installable security subsystem and what is required knowledge to be able to build an installable security subsystem.
- Chapter 8, “Installable Security Subsystem Design Guidelines”
This chapter contains an action list that can be used when building an installable security subsystem.
- Chapter 9, “SES Architecture Implementation”
This chapter contains in-depth information on the security enabling services architecture and the major components of the current implementation.
- Chapter 10, “Interoperation of SES and ISS”
This chapter describes how the key security enabling services components cooperate to enable an installable security subsystem to provide a secure OS/2 operating system.
- Chapter 11, “Security Kernel Services (SKS)”
This chapter provides in-depth information on the security kernel services.

- Chapter 12, “Security Context Services (SCS)”

This chapter provides in-depth information on the security context services.

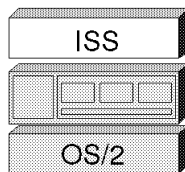
- Chapter 13, “Logon Shell Services (LSS)”

This chapter provides in-depth information on the logon shell services.

- Chapter 14, “Installation, Configuration, Initialization Support”

This chapter provides in-depth information on the installation, configuration, and initialization support.

Chapter 7. Building an Installable Security Subsystem



To build an installable security subsystem (ISS), the developer needs to know which security enabling services privileges will be required by the installable security subsystem programs/processes, which APIs and KPIs will be used by the installable security subsystem (ISS), and what security enabling services security policy driver DLLs (SLD, CLD, PVD) will be provided by the installable security subsystem for the security features that the installable security subsystem supports.

The following is a brief summary of the function of key ISS components:

- UIA** Identifies and authenticates the user for local system logon.
- SLA** Establishes security context for local system logon and enforces installable security subsystem security policy.
- CLA** Acts on behalf of the local system user to logon to remote services.
- RLA** Establishes a local security context for a remote client user.
- ACA** Controls access to local system resources.
- APA** Executes on behalf of a single client and a trusted agent.
- SPA** Executes on behalf of multiple clients and a trusted agent.
- SLD** Determines the order in which to invoke UIAs.
- CLD** Determines the order in which to invoke CLAs.
- PVD** Determines the validity of passwords (composition, history, etc.).

The following sections show which SCAs, APIs, KPIs, and DLLs will typically be applicable for various security features:

- Logon
- Resource Access Control
- Audit
- Single Signon
- Trusted Program Support

7.1 Logon

Local system logon involves first identifying and authenticating the user (UIA), and then establishing of the association between the user's security credentials (user identifier, group membership, administrative privileges) and the processes executing on behalf of the user (SLA).

To enable the perception of single signon in environments where multiple local/remote identification and authentication services are required, these identification and authentication services can be divided into the following functions:

- Identification and authentication for local system logon (UIA) through the SLA.
- Identification and authentication for typically remote services (CLA) that can't accept the local system identification and authentication.

The authorities that will typically be applicable for logon are the user identification authority and system logon authority (for local system logon) and the client logon authority and remote logon authority (for client/server logon). The APIs/KPIs that will typically be used by the installable security subsystem for logon depends on what services the installable security subsystem provides. All three of the security enabling services DLLs could be replaced by the installable security subsystem for logon services.

SCA	API	KPI	DLL
CLA RLA SLA UIA	SESSendSecurityContext SESCreateSubjectHandle SESDeleteSubjectHandle SESSetSubjectHandle SESQuerySubjectHandle SESCreateInstanceHandle SESSetSubjectInfo SESQuerySubjectInfo SESQuerySubjectHandleInfo SESSetContextStatus SESQueryContextStatus SESSetSecurityContext SESResetThreadContext SESQuerySecurityContext SESQueryAuthorityID SESKillProcess SESControlProcessCreation SESControlKBDMonitors SESRegisterDaemon SESReturnEventStatus SESReturnWaitEvent SESWaitEvent SESQueryAuthorityID SESQueryProcessIDs SESWaitEvent: SEND_SECURITY_CONTEXT SET_CONTEXT_STATUS SLDInit SLDQueryUIA CLDInit CLDQueryCLA PVDValidatePassword	See appendix A for KPI details.	CLD PVD SLD

Table 2. Applicable SCA, API, KPI, and DLL for Logon

7.2 Resource Access Control

Discretionary access control for resources is primarily accomplished through the access control authority. The APIs/KPIs that will typically be used by the installable security subsystem for discretionary access control depends on what services the installable security subsystem provides. None of the security enabling services DLLs should need to be replaced for discretionary access control services.

SCA	API	KPI	DLL
ACA	SESCreateHandleNotify SESDeleteHandleNotify SESQuerySubjectHandle SESReserveSubjectHandle SESReleaseSubjectHandle SESQuerySubjectInfo SESQuerySubjectHandleInfo ESSetContextStatus SESQueryContextStatus SESResetThreadContext SESQuerySecurityContext SESQueryAuthorityID SESWaitEvent: SEND_SECURITY_CONXTXT SESRegisterDaemon SESStartEvent SESReturnEventStatus SESReturnWaitEvent	See appendix A for KPI details.	none

Table 3. Applicable SCA, API, KPI, and DLL for DAC

7.3 Audit

The independent software vendor can determine which events need to be audited in the system and can implement the auditing code in the installable security subsystem. KPIs are especially important to the audit function; they are the means for accessing basic file information such as open, close and delete. The installable security subsystem can add audit recording code to any of the system authorities, and can record the results of any of the system APIs and KPIs. The APIs and KPIs typically used by the installable security subsystem for audit depends on what services the installable security subsystem provides. None of the security enabling services DLLs should need to be replaced for audit services.

SCA	API	KPI	DLL
ACA APA CLA RLA SLA SPA UIA	This is an ISV defined function. Any of the authorities could call APIs which might pass back potentially auditable information.	See appendix A for KPI details.	none

Table 4. Applicable SCA, API, KPI, and DLL for Audit

7.4 Single Signon

Client logon authority has been designed to facilitate the perception of single signon for the system. To perform this task, the client logon authority must have access to the identification and authentication information previously examined by the user identification authority and system logon authority and must be able to create unique handles. The APIs and KPIs that will typically be used by the installable security subsystem for single signon depends on what services the installable security subsystem provides. The client logon driver could be replaced for single signon services.

SCA	API	KPI	DLL
CLA	SESSendSecurityContext SESCreateHandleNotify SESDeleteHandleNotify SESQuerySubjectHandle SESCreateInstanceHandle SESQuerySubjectInfo SESQuerySubjectHandleInfo SESSetContextStatus SESQueryContextStatus SESResetThreadContext SESQuerySecurityContext SESQueryAuthorityID SESWaitEvent: SEND_SECURITY_CONTEXT	See appendix A for KPI details.	CLD

Table 5. Applicable SCA, API, KPI, and DLL for Single Signon

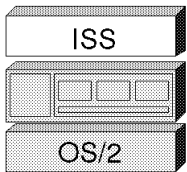
7.5 Trusted Program Support

Trusted program support is typically provided through installable security subsystem APIs. However, the security enabling services privileges associated with agent process authority and server process authority enable a process to exploit the trusted program support. The APIs and KPIs that will typically be used by the installable security subsystem for trusted programs depends on what services the installable security subsystem provides. None of the security enabling services DLLs should need to be replaced for trusted program support.

SCA	API	KPI	DLL
APA SPA	SESSendSecurityContext SESSetSubjectHandle (SPA only) SESQuerySubjectHandle SESReserveSubjectHandle (SPA only) SESReleaseSubjectHandle (SPA only) SESQuerySubjectInfo SESQuerySubjectHandleInfo SESQueryContextStatus SESResetThreadContext SESQuerySecurityContext SESQueryAuthorityID SESWaitEvent: SEND_SECURITY_CONTXT (SPA only)	See appendix A for KPI details.	CLD

Table 6. Applicable SCA, API, KPI, and DLL for Trusted Program Support

Chapter 8. Installable Security Subsystem Design Guidelines



Refer to Chapter 12, “Security Context Services (SCS)” on page 161 for information on building and installing an installable security subsystem.

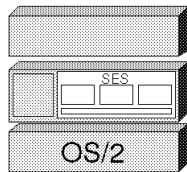
The following action list is provided as a guide for developing installable security subsystem applications.

Action	Explanation
Study Part 2, “Design Notes” on page 79	This section of the document provides details on the system architecture which the designer must understand before an installable security subsystem can be created. It shows the interactions of authorities and the system calls (APIs and KPIs) make.
Choose the security function to implement.	Determine the security function to add to the system. The system is designed to readily incorporate new identification and authentication, discretionary access control and audit functions. Some form of single signon capability already exists in the system; it may be easy to enhance the features already provided for this function, but replacing the function would be difficult. Trusted program support is also provided in the basic system design.

Action	Explanation
Study the predefined sequence of events for the function you want to implement.	Specific events have already been defined for the system. Examine them to discover the authorities, APIs, etc. you will need for your application.
Determine the authorities you need to implement your function.	Specific system privileges have been assigned to each of the defined authorities.
Do your applications need to run with multiple authorities?	Your applications can run with multiple authorities if needed. The system also allows multiple user identification authorities and client logon authorities.
Do you need to implement a client/server solution?	Client logon and remote logon are the authorities that have been designed for client/server environments.
Do you need to add to or modify the user display?	The system logon authority usually contains some kind of user display application. User identification authority and client logon authority have the ability to query the user for logon information.
Do you need to interface with pre-existing security solutions?	Determine the requirements of the pre-existing solution. For example, it is possible to associate the user credentials in the OS/2 environment with credentials created by a trusted program such as a LAN Server. This would be done in an installable security subsystems internal database. Make sure that any independent software vendor defined installable security subsystem APIs are compatible with the existing programs.

Action	Explanation
Determine the APIs and KPIs you need to use.	<p>To determine the APIs and KPIs that will be used, you will need to assess the following:</p> <ul style="list-style-type: none"> • Do you need to reference, delete, create or modify handles or threads? • Do you need to add to or pass user credentials on to a trusted application? • Do you need to receive notification of system events? <p>Note that these questions should also be asked when determining what authorities to run with.</p>
Do you need to modify or replace the contents of the CLD, SLD or PVD?	<p>Modify the client logon driver if you need to determine the order in which client logon authorities should be called. Modify the system logon driver if you need to determine the order in which UIAs should be called. Modify the password validation driver if you need to add password rule checking to the system.</p>
Do you need to use the POSIX compliant inheritance scheme?	<p>If the answer to this question is yes, make sure the /PROPAGATE flag in SECURE.SYS is yes for each program wanting the POSIX-compliant inheritance scheme.</p>
Do you need to modify CONFIG.SYS and SECURE.SYS?	<p>Modify CONFIG.SYS if you want to include an installable security subsystem in the system. Modify SECURE.SYS if you want to assign authorities to installable security subsystem applications.</p>

Chapter 9. SES Architecture Implementation



Security Enabling System (SES) is implemented as a set of application processes, dynamic link libraries, and kernel components (device driver, DevHlps). SES also requires changes to the OS/2 kernel and keyboard/mouse device drivers. Figure 16 on page 96 provides an overview of the SES architecture and related ISS components:

ISS API:	DLL functions that provide an application programming interface for ISS services.
ISS Security Daemon(s):	Ring-3 component(s) of the installable security subsystem, SLA, UIA, CLA, ACA, etc.
ISS Security Kernel:	Ring-0 component of the installable security subsystem (implemented as an OS/2 device driver).
SES API:	DLL functions that provide an application programming interface for security enabling services.
SES KPI:	IDC functions that provide a kernel programming interface for security enabling services.
SES Daemons:	Ring-3 components of SES (SES daemon, PSS daemon, SESShell daemon).
SES Device Driver:	Ring-0 component of security enabling services.

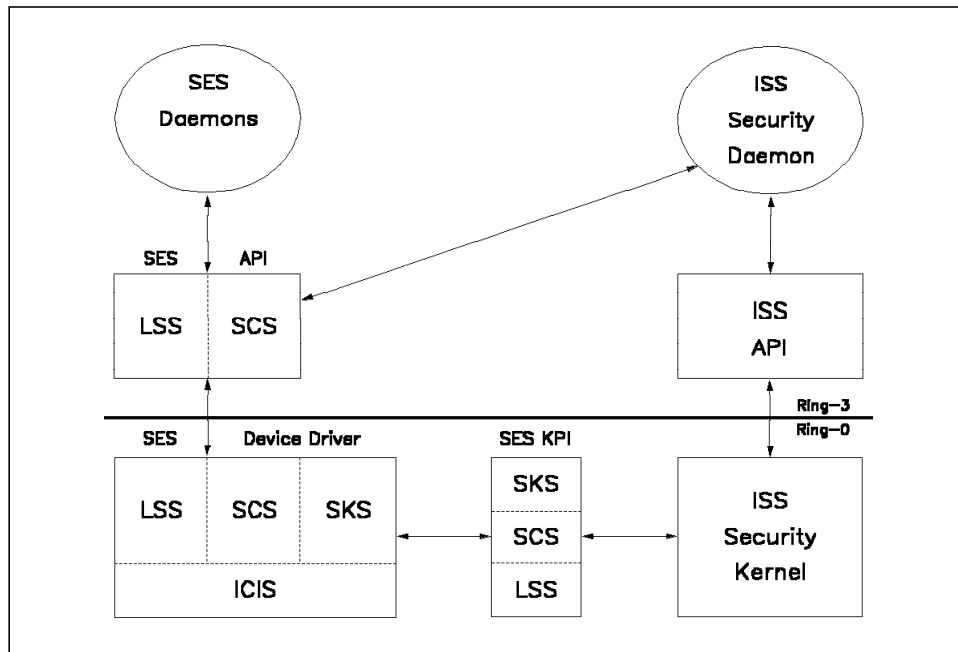
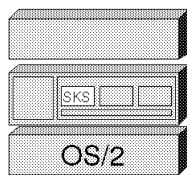


Figure 16. SES Architecture - Overview of Key Components

9.1 Security Kernel Services



The security kernel services architecture consists of a security event router and security helpers in the OS/2 kernel. The security event router supports a wide range of security solutions by providing a way for an installable security subsystem to specify exactly which security relevant events it wants to intercept. When specified events such as file access or process creation occur, the router notifies the installable security subsystem which then has the opportunity to grant or deny access. Security kernel services also provides a set of kernel level security helpers that enable an installable security subsystem to invoke operating system services that aren't otherwise available at the kernel level.

The following illustration demonstrates the general security kernel services model for interaction between the installable security subsystem security kernel and OS/2 kernel components such as the file system router.

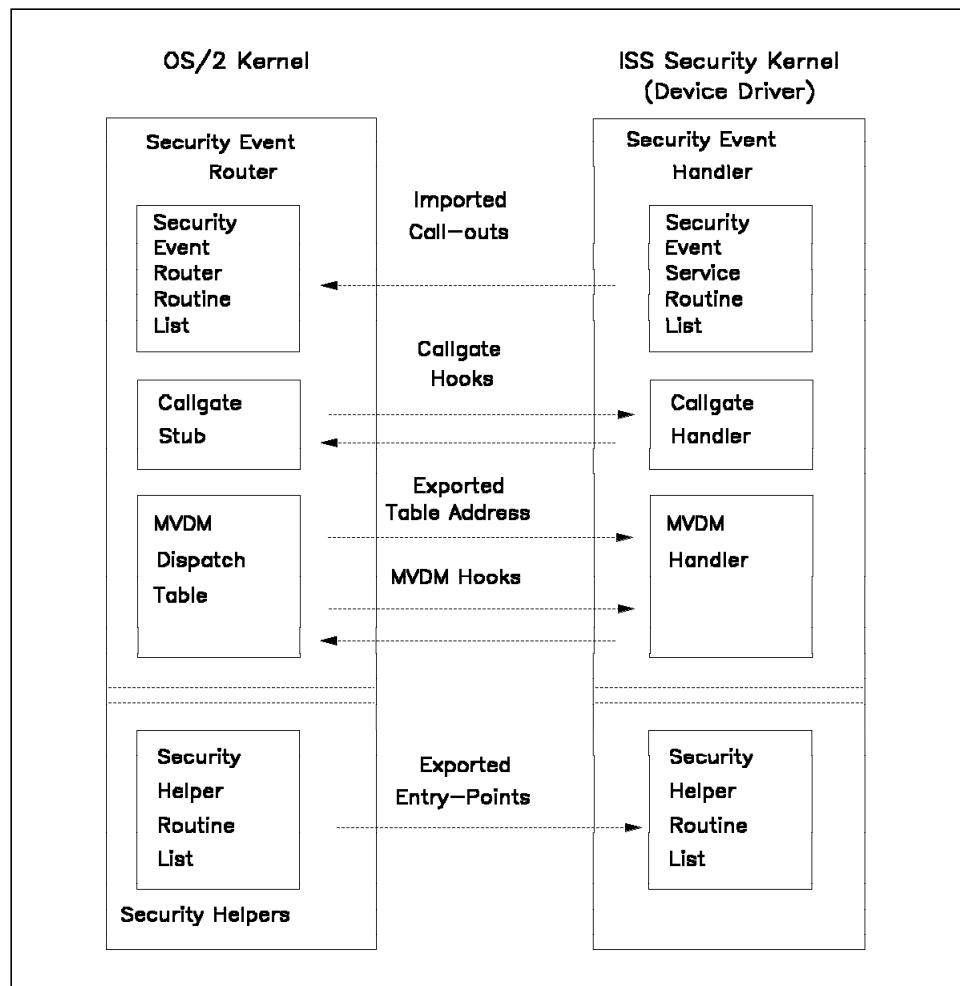


Figure 17. SKS Architecture - Interaction between SKS and ISS Security Kernel

9.1.1 Security Event Router

The security event router hooks enable an installable security subsystem security kernel to intercept the following security relevant OS/2 kernel services/support:

- Security Relevant OS/2 System Calls
- Callgate Level Support
- Multiple Virtual DOS Machine Support
- Logon Shell Services Trusted Path Support
- Security Enabling Services API Audit Support

9.1.1.1 Callouts for Security Relevant OS/2 System Calls

For a selected set of OS/2 kernel system calls (for example DosOpen, DosExecPgm) the OS/2 kernel worker routines have been modified to call a corresponding routine in the installable security subsystem.

The installable security subsystem services the notification and returns with the appropriate status condition for the event. If an error code is returned, then the worker routine does not continue.

The OS/2 system calls that are currently supported include the following:

- Change directory
- Change file pointer
- Close
- Delete
- Device input/output control
- Execute program
- Find close
- Find close 3X
- Find first
- Find first 3X
- Find next
- Get module
- Loader open
- Make directory
- Move
- Open
- Query file information
- Read
- Remove directory
- Set date/time
- Set file information
- Set file size

- Set path information
- Write

For further details see appendices.

9.1.1.2 Callouts for Callgate Level Support

Since it is impossible to anticipate every possible callout an installable security subsystem may require, an extensible architecture was developed for hooking calls into the kernel at the callgate level. This allows an installable security subsystem to monitor events for which there are no callouts. Callgate hooks are a second class citizen to the OS/2 system call hooks, they result in slower system performance and are more difficult for installable security subsystem developers to use.

Callgate level hooks are provided for both 16-bit and 32-bit OS/2 system calls.

9.1.1.3 Callouts for Multiple Virtual DOS Machine Support

The OS/2 system call hooks catch all file system calls in a Virtual DOS Machine (VDM). This should be sufficient to enforce access control in a DOS session. The security kernel services architecture provides an installable security subsystem the capability to hook the MVDM (Multiple VDM) Dispatch Table so, as with callgate hooks, an installable security subsystem has maximum flexibility in specifying and screening security events.

9.1.1.4 Callouts for Logon Shell Services Trusted Path Support

A callout is provided for the logon shell services trusted path support so that the installable security subsystem security kernel can be notified when the trusted path key combination is detected and can direct the keyboard device driver to take the appropriate action.

9.1.1.5 Callouts for Security Enabling Services API Audit Support

Callouts are provided for all security control services APIs so that the installable security subsystem security kernel can audit these functions.

9.1.2 Security Helpers

Security helpers are routines that an installable security subsystem can call from a device driver for basic operating system services that aren't otherwise available at Ring-0.

- File System Services
- Security Context Services

9.1.2.1 Security Helpers for File System Services

The file system services that are currently supported include the following:

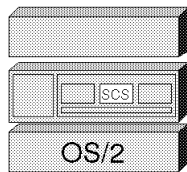
- Change pointer
- Close
- Find next
- Open
- Query size
- Read
- Return fully qualified path name given system file number
- Return system file table entry given system file number
- Write

For further details see appendices.

9.1.2.2 Security Helpers for Security Context Services

Security helpers are provided for most of the security context services APIs so that the installable security subsystem security kernel can invoke these from Ring -0. These SecHlps are described in 9.2, "Security Context Services."

9.2 Security Context Services



Security control services addresses the requirements to support multiple concurrently active security applications, multiple concurrently active users, and trusted programs. This is accomplished by associating each process with the information necessary for security applications (such as an installable security subsystem) to determine the credentials (for example, user ID, group ID(s), administrator privileges, etc.) of the user on whose behalf the process is executing. The information that security control services associates with each process is referred to as the security context of the process.

The security context associated with each process (and each thread of execution of a process) consists of the following:

- Set of subject handles that a security application can associate with user/group/process credentials
- Set of security context status flags that are used to maintain SES status information

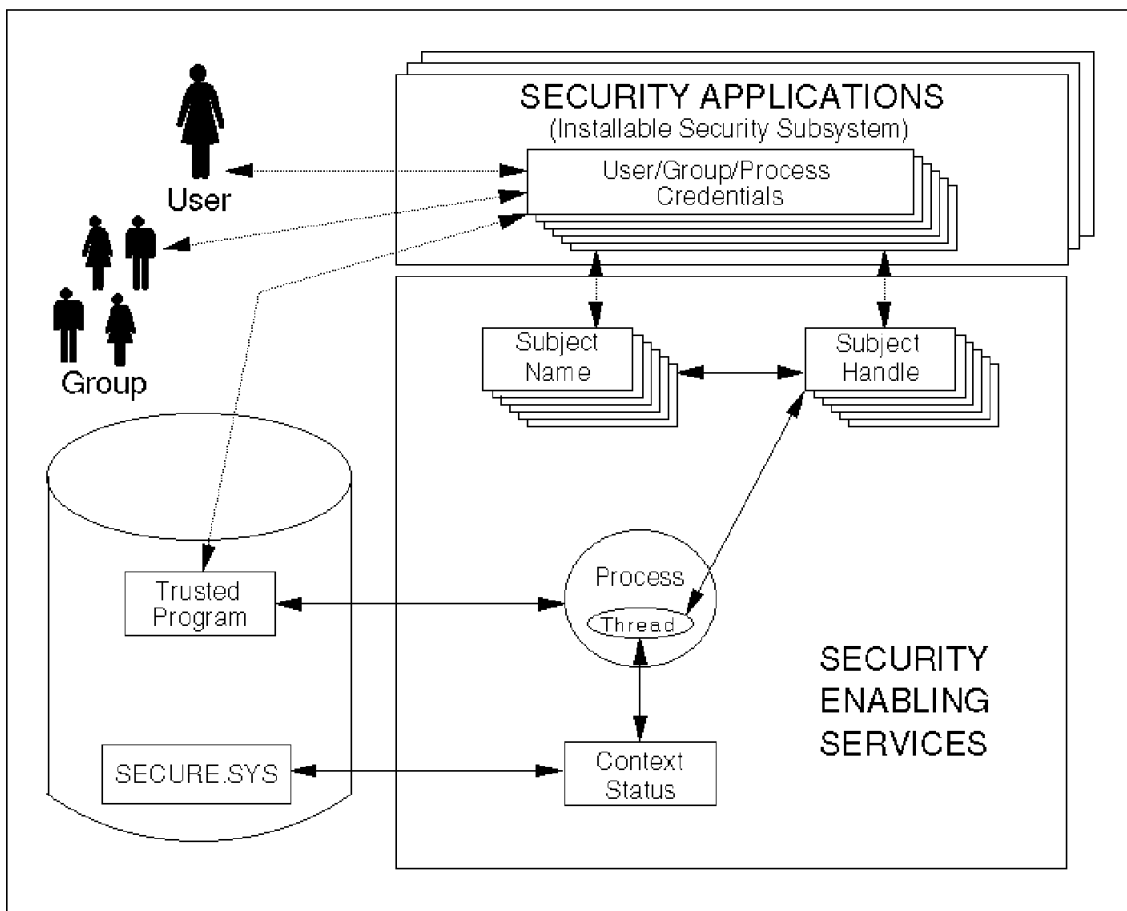


Figure 18. SCS Architecture - User/Group/Process Security Credentials

The following sections describe the security context and the associated services provided by security control services.

9.2.1 Process-User Association

Security control services enables each security application to manage its own view of the process-user association by doing the following:

- Associating credentials (user ID, group membership, trusted program privileges, etc.) for each user/group/process with a unique subject handle
- Associating each subject (for example process executing on behalf of a user/group) with a set of subject handles (user, group, process)

Using these subject handles, security applications can recognize what user/group/process credentials should be associated with a process by querying the subject handles that are associated with the process. This concept allows multiple security applications to interoperate without forcing a common view of user/group/process credentials.

To support trusted programs (for example, POSIX setuid), security enabling services maintains both client and agent user/group/process handles for each process/thread. The client handle is associated with the user/group/process using the services of a security application (for example, the user who invoked a trusted program). The agent handle is associated with the user/group/process credentials of the security application itself (for example, the owner of a trust program).

9.2.1.1 Subject Handles

The security context associated with each process/thread includes the following subject handles:

Client User Handle (CUH)	The client user handle enables an installable security subsystem to associate a process with user credentials for the user on whose behalf the process is executing.
Agent User Handle (AUH)	The agent user handle enables an installable security subsystem to associate a trusted program/process with special user credentials, for example the POSIX setuid model. In addition, a server process can assume the effective (client or agent) user handle of a client process (when requested by the client process) as the server's client user handle (not the server's agent user handle) thus extending the security enabling

	services trusted program/process model to a client/server IPC model.
Client Group Handle (CGH)	The client group handle enables an installable security subsystem to associate a process with group credentials for the group on whose behalf the process is executing.
Agent Group Handle (AGH)	The agent group handle enables an installable security subsystem to associate a trusted program/process with special group credentials, for example the POSIX setgid model. In addition, a server process can assume the effective (client or agent) group handle of a client process (when requested by the client process) as the server's client group handle (not the server's agent group handle) thus extending the security enabling services trusted program/process model to a client/server IPC model.
Client Process Handle (CPH)	The client process handle enables an installable security subsystem to associate a process with additional information that is not directly associated with user/group credentials and should not be affected by changing the effective user/group of the process for example the POSIX umask model.
Agent Process Handle (APH)	The agent process handle enables a server process to assume the effective (client or agent) process handle of a client process (when requested by the client process) as the server's client process handle (not the server's agent process handle) thus extending the security enabling services trusted program/process model to a client/server IPC model.

When a subject handle is created, it is associated with a subject name and optionally a subject token. The name is the key by which all security components recognize the same subject (user/group/process). The token is

used to facilitate the perception of single signon for multiple user identification and authentication components such as a local signature verification mechanism and a remote server password mechanism. The source of authority for creation of the handle is preserved in the source field. The instance field is discussed in the next section.

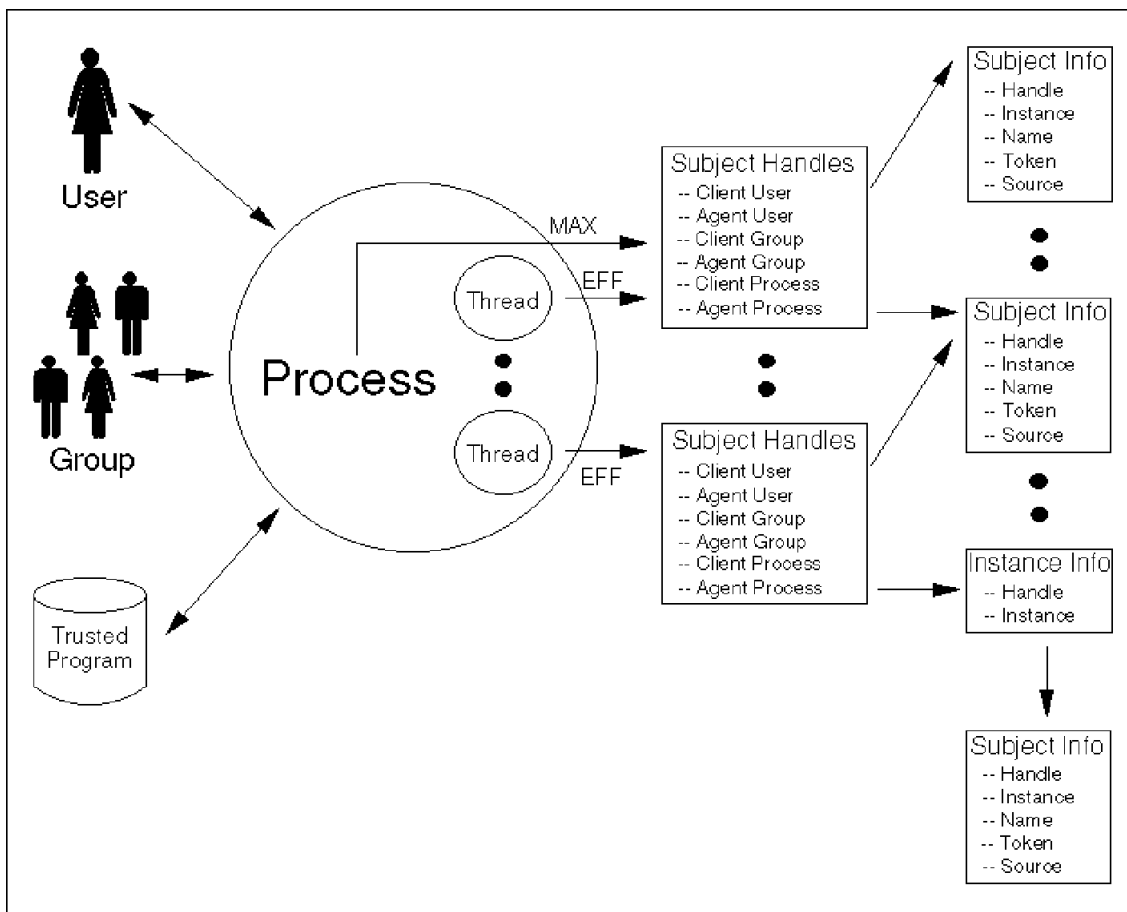


Figure 19. SCS Architecture - Subject Handles

The association between a subject handle (dynamically-generated unique value) and a user/group/process is unique for the life of a boot for example each handle can only be associated with one user/group/process (although one user/group/process can be associated with multiple subject handles).

Note that this association is dynamically created when the subject handle is created. For example it is not statically maintained across system boots

when the system is rebooted; all existing subject handle information is lost. Security applications (such as an installable security subsystem) must maintain the static association between a user/group/program and the associated credentials.

To facilitate security application management of user/group/process credentials associated with subject handles, security applications can register to be notified when subject handles are created or deleted.

- Security applications that are registered for create handle notification are notified when a subject handle is created, whether or not the subject handle is associated with any process/thread at the time it is created.
- Security applications that are registered for delete handle notification are notified when a subject handle is deleted and when the handle is no longer referenced by any process/thread. For example security applications will not be notified if either the subject handle is not deleted or if the subject handle is still referenced by any process/thread.

Note: For subject handles that are directly associated with a user and should only be regarded as valid while a user's processes are active, the subject handles should be deleted after they are set into the security context of the user's processes. This will allow the delete handle notification to proceed when these subject handles are no longer referenced by any process/thread.

For subject handles that are not directly associated with a user and should be regarded as valid even though they're not associated with a user's processes (for example, subject handles for groups or trusted programs), the subject handles should not be deleted. This will prevent the delete handle notification from proceeding even though a subject handle is not currently referenced by any process/thread (but may be associated with a process/thread in the future).

Two subject handles are reserved (for example, never created or deleted) and require special recognition by all security components.

Subject Handle = 0:	superuser or system process, should
- name = '' (null)	be granted full access to all objects
- token = '' (null)	(at least during system initialization).
- source = 0	

Subject Handle = -1:	unauthenticated user, should not be
- name = 'Guest'	granted access to protected objects
- token = '' (null)	(except for guest/public/default etc.).
- source = 0	

9.2.1.2 Instance Handles

Security control services enables a client logon authority, acting on behalf of a client process, to create/assign new user/group/process handles for the client process that the client logon authority can associate with different user/group/process credentials than the original subject handles of the client process. However, since a client logon authority can only change its view of what credentials are associated with its client process, the new handles that the client logon authority creates/assigns must not affect other security control authority views of what credentials are associated with the client logon authorities process. For example, for all security context authorities other than the client logon authority that created/assigned these new handles for a client process, the handles should be associated with the same credentials as the original subject handles that were associated with the client logon authorities client process.

Consequently, when a handle is created by a client logon authority (for a client process), it is marked as an instance of the original subject handle. For all security context authorities other than the client logon authority that creates an instance handle, the handle is associated with the same user/group/process credentials as the original subject handle. The effect of a client logon authority creating/assigning an instance handle is to change its view of the user/group/process credentials associated with its client process without affecting any other security context authorities view of the user/group/process credentials associated with the same process.

A client logon authority process may create/assign an instance handle for a client process only when the client logon authorities client logon authorities services are requested through security enabling services. Special security enabling services functions enable the client logon authority to create/assign instance handles for the subject handles of its client processes (only). Once instance handles are created/assigned by a client logon authority for a client process, the instance handles are managed the same as any subject handle (for example, a child process will inherit instance handles in the same way as other subject handles).

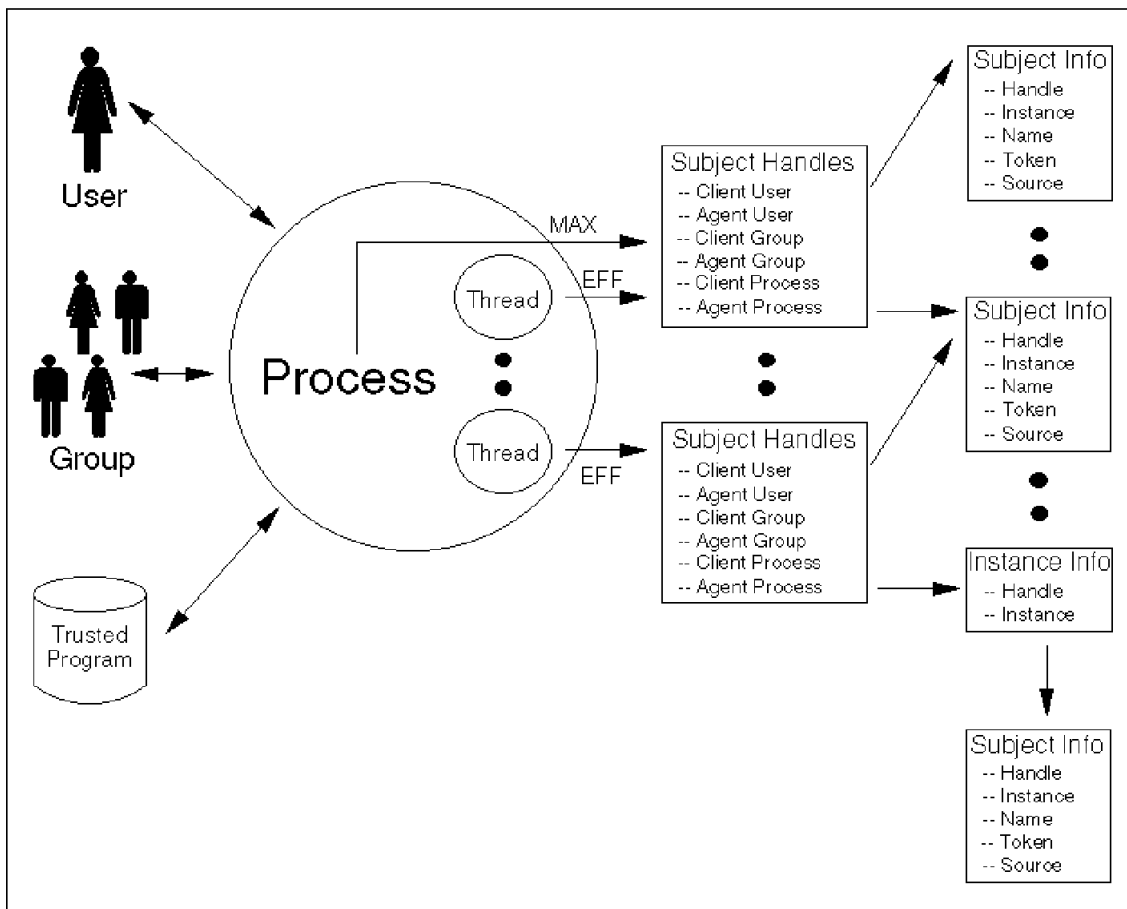


Figure 20. SCS Architecture - Instance Handles

Again, to facilitate security application management of user/group/process credentials associated with subject/instance handles, security applications can register to be notified when subject/instance handles are created or deleted. As discussed previously, to kick off delete handle notification for subject handles, they must be explicitly deleted after they are set in the security context of a process/thread. However, instance handles are implicitly deleted when they are set in the security context of the client logon authorities process. Consequently, delete handle notification, for instance handles, will automatically be kicked off when the instance handle is no longer referenced by the client logon authorities process (or any process that inherited the instance handle from the client logon authorities process).

- Security applications that are registered for create handle notification are notified when an instance handle is created, whether or not the instance handle is associated with any process/thread at the time it is created.
- Security applications that are registered for delete handle notification are notified when an instance handle is no longer referenced by any process/thread for example security applications will not be notified if the instance handle is still referenced by any process/thread.

9.2.2 Process-Status Association

The security context status of a process includes information about the state of the process (with respect to security enabling services functions for example the current state of the effective (client or agent) user/group/process flags) and the authority of the process (again with respect to security enabling services functions, for example, the authority of a process to invoke privileged security enabling services functions).

The security context status includes the following state flags:

EGF:	Effective Group Flag
EPF:	Effective Process Flag
EUF:	Effective User Flag
LUF:	Local User Flag
PAF:	Propagate Authority Flag

Some privileged security enabling services functions must only be invoked by processes that can be trusted to use the functions appropriately. A trusted process (program) that has the authority to invoke these privileged functions is referred as a security context authority (SCA). To support the concept of least privileged operation, the privileged security enabling services functions are divided into sets of operations that correspond to the various roles that a trusted application might serve in a secured OS/2 system:

ACA:	Access Control Authority
APA:	Agent Process Authority
CLA:	Client Logon Authority
RLA:	Remote Logon Authority
SLA:	System Logon Authority
SPA:	Server Process Authority
UIA:	User Identification Authority

Each security control authority role (and the corresponding set of privileged functions) is represented by an authority flag in the security context status that is associated with each process. When a privileged security enabling services function is invoked by a process, these flags are checked to ensure that the process has the appropriate authority to invoke the privileged function.

Note that a process (program) may be serving multiple security control authority roles and, consequently, may have more than one authority flag set in its security context. For example, an access control authority could also be an server process authority and, consequently, would have both the access control authority and server process authority flags set in its security context.

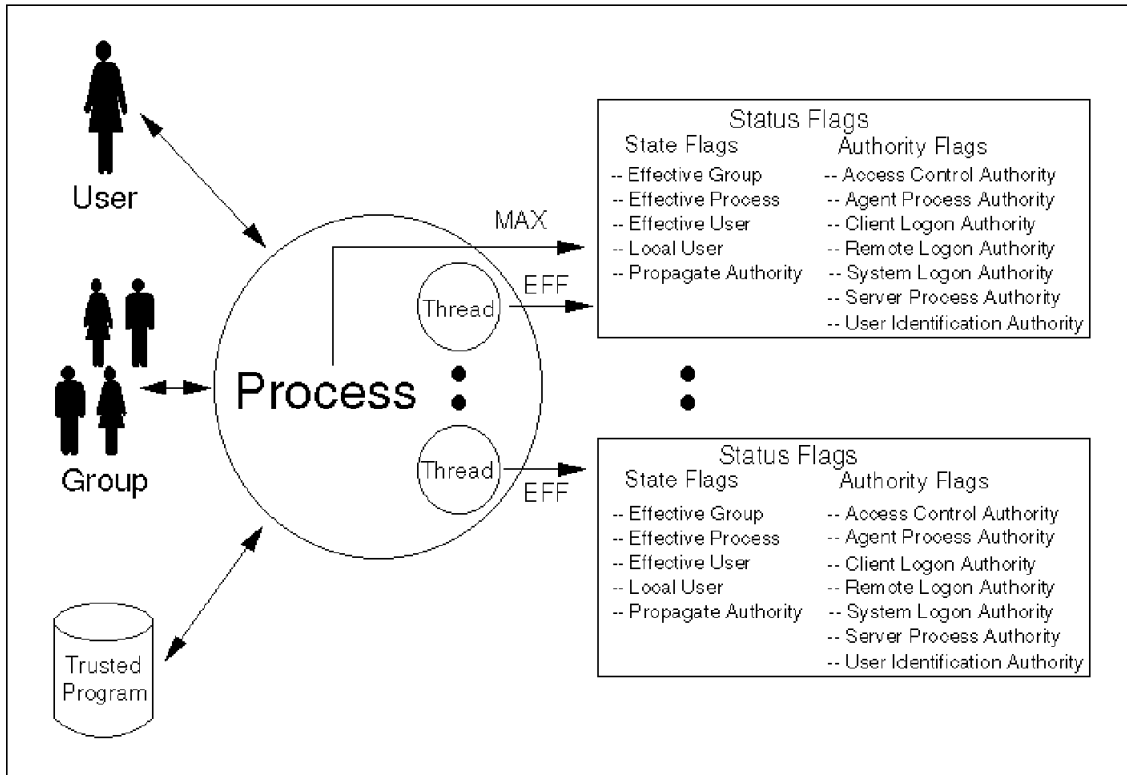


Figure 21. SCS Architecture - Process Authority

9.2.3 Definition of Security Context

The security context associated with each OS/2 process is defined as the set of subject handles (client/agent user/group/process) and status information that can be used by a security application (such as an installable security subsystem) to associate its own user/group/process credentials (user ID, group membership, trusted program privileges) with the process.

- Client User Handle (CUH)
- Agent User Handle (AUH)
- Client Group Handle (CGH)
- Agent Group Handle (AGH)
- Client Process Handle (CPH)
- Agent Process Handle (APH)
- Security Context Status

The security context associated with the process (but not necessarily with any thread of execution of the process) is referred to as the maximum security context. The security context associated with a thread of execution of a process is referred to as the effective security context.

9.2.3.1 Maximum Security Context

The maximum security context is established at process creation. All threads of a process always share the same maximum security context. The maximum security context of a process reflects the highest authority that any thread of the process can set in its effective security context.

9.2.3.2 Effective Security Context

A thread of a process can set its effective security context authority up (to the maximum) and down as necessary for example a thread may need to set its authority down when creating another process so that the new process doesn't inherit all of the authority of the thread.

The default for the effective security context is a process model such as all threads of a process share the same effective security context; so, if one thread makes a change in the effective security context, all other threads of the process will see the same change. This process security context model is enforced until a thread of an SCA process explicitly requests to have its own effective security context.

If a thread of a security control authority process explicitly requests to have its own effective security context, then the effective security context for the thread is maintained separate from the effective security context of the process (and separate from the maximum security context of the process).

When a thread that has requested a separate effective security context makes any changes to it, the changes will not affect the effective security context of any other thread of the process or the effective security context of the process (or the maximum security context of the process).

9.2.3.3 Subject Handle Information

The purpose of each of the subject handles was discussed previously, to summarize:

Client User Handle (CUH)	Enables association of a process with user credentials for the user on whose behalf the process is executing, for example, POSIX real uid.
Agent User Handle (AUH)	Enables association of a trusted process with user credentials for a special user associated with a trusted program, for example, POSIX saved uid.
Client Group Handle (CGH)	Enables association of a process with group credentials for the group on whose behalf the process is executing, for example, POSIX real gid.
Agent Group Handle (AGH)	Enables association of a trusted process with group credentials for a special group associated with a trusted program, for example, POSIX saved gid.
Client Process Handle (CPH)	Enables association of a process with credentials/information that should not be affected by changing the effective user/group, for example, POSIX umask.
Agent Process Handle (APH)	Enables association of a trusted server process with a client's process credentials/information or its own process credentials/information.

The information maintained for each subject handle includes the name of the subject associated with the handle, the token (password) of the user (applicable for user handles only), the source of authority for creation of the handle, and a pointer to the original handle of an instance handle.

- Handle
- Instance
- Name

- Token
- Source

The instance field is used to indicate whether the handle is an original handle created by a system logon authority or remote logon authority, or an instance of a handle created by a client logon authority. If the instance field is set to -1, then the handle is an original handle created by a system logon authority or remote logon authority. Otherwise the instance field contains the original handle that this handle is an instance of. Note that an instance handle may be an instance of another instance handle but an original handle created by a system logon authority or remote logon authority will be at the root of a chain of instance handles.

The source field indicates which SLA/RLA created the handle or which user identification authority authenticated the user (or what authentication rule was used to authenticate the user) for local system logon. This field is used to enforce the policy that RLAs can only set handles that they create, and to enforce the policy that SLA/RLAs can only delete handles that they create. This field could also be used by the installable security subsystem or an access control authority to give a process more or less privileges depending on the source of authority for creation of the handle, for example, force system administrators to use a fingerprint device for authentication, but allow other users to use a Personal Identification Number (PIN) device for authentication.

The value specified in the source field is determined by the following methods depending on how the subject handle is created:

1. If the subject handle is created by a system logon authority or remote logon authority, then the source field will be set equal to the authority ID of the creating process (SLA or RLA).
2. For the client user handle created during local system logon (by security enabling services), the source field may contain either the authority ID of the UIA that authenticated the user or an authentication rule number. An authentication rule number could be used to indicate that a more complex authentication algorithm, potentially involving multiple user identification authorities, was used to authenticate the user. For example, authentication rule numbers could be defined to indicate that the authentication for local system logon was accomplished by various combinations of user identification authorities.

The value specified in the source field indicates whether it is an authority ID or an authentication rule number:

- 4-255: Authority ID. The authority ID is discussed in more detail in the following section that describes the security context status.
- 256-MAX: Authentication Rule Number. The authentication rule number is discussed in more detail as part of the Logon Shell Services (LSS), Chapter 13, “Logon Shell Services (LSS)” on page 189.

9.2.3.4 Security Context Status

The security context status consists of an authority ID, state flags, and authority flags.

- The authority ID associates a process with the corresponding trusted programs specified in SECURE.SYS.
- The state flags represent the state of a process/thread (with respect to security enabling services for example whether the effective group handle is currently the client group handle or the agent group handle).
- The authority flags represent the authority of a process/thread (again with respect to security enabling services, for example, whether the process has access control authority privileges or not).

Authority ID: Each security control authority specified in SECURE.SYS (except agent process authority) is assigned a unique authority ID during security enabling services initialization. Security enabling services uses the authority ID for the following purposes:

1. The authority ID is used to designate the target security control authority for the SESSendSecurityContext() API.

For processes with no special authority (UPA) or only agent process authority (APA), the authority ID is set to zero (AuthorityID=0) and, consequently, UPAs and APAs can't be the target for SESSendSecurityContext().

Note that multiple cooperating processes that are acting as a single security control authority will have the same authority ID. For example, multiple programs may need to act together as a single client logon authority application the corresponding processes will be assigned the same authority ID.

2. An authority ID can be specified in the source field of the subject handle information structure. The authority ID of an SLA/RLA can be specified as the source of authority for creation of subject handles, or the authority ID of a user identification authority can be specified as the source of authentication for the client user handle during local system logon.

The source field in the subject handle information structure is used to enforce policies on setting/deleting subject handles (by comparing it against the authority ID of the process that is attempting to set/delete a subject handle).

State Flags: In the maximum security context, the state flags indicate whether the process can modify the corresponding state flags in the effective security context or not. In the effective security context, the state flags indicate the current state of a process/thread with respect to security enabling services, for example whether the effective group handle is currently the client group handle or the agent group handle.

- **Effective Group Flag (EGF)**

The effective group flag in the maximum security context specifies whether the process can modify the effective group flag in the effective security context or not:

Maximum EGF=0: Process/thread cannot modify EGF in effective security context

Maximum EGF=1: Process/thread can modify EGF in effective security context

The effective group flag in the effective security context specifies whether the process/thread effective group handle is the client group handle or the agent group handle:

Effective EGF=0: Process/thread Effective Group Handle
=> Client Group Handle

Effective EGF=1: Process/thread Effective Group Handle
=> Agent Group Handle

At process creation, the effective group flag will be set (EGF=1) in the maximum and effective security contexts if the process has any authority flags set (APA=1, SPA=1, etc.). The effective group flag in the effective security context may be reset (EGF=0) by the installable security subsystem at process creation.

- **EPF (Effective Process Flag)**

The effective process flag in the maximum security context specifies whether the process can modify the effective process flag in the effective security context or not:

Maximum EPF=0: Process/thread cannot modify EPF in effective security context

Maximum EPF=1: Process/thread can modify EPF in effective security context

The effective process flag in the effective security context specifies whether the process/thread effective process handle is the client process handle or the agent process handle:

Effective EPF=0: Process/thread Effective Process Handle
=> Client Process Handle
Effective EPF=1: Process/thread Effective Process Handle
=> Agent Process Handle

At process creation, the effective process flag will be set (EPF=1) in the maximum and effective security contexts if the process has any authority flags set (APA=1, SPA=1, etc.). The effective process flag in the effective security context may be reset (EPF=0) by the installable security subsystem at process creation.

- **Effective User Flag (EUF)**

The effective user flag in the maximum security context specifies whether the process can modify the effective user flag in the effective security context or not:

Maximum EUF=0: Process/thread cannot modify EUF in
effective security context
Maximum EUF=1: Process/thread can modify EUF in
effective security context

The effective user flag in the effective security context specifies whether the process/thread effective user handle is the client user handle or the agent user handle:

Effective EUF=0: Process/thread Effective User Handle
=> Client User Handle
Effective EUF=1: Process/thread Effective User Handle
=> Agent User Handle

At process creation, the effective user flag will be set (EUF=1) in the maximum and effective security contexts if the process has any authority flags set (APA=1, SPA=1, etc.). The effective user flag in the effective security context may be reset (EUF=0) by the installable security subsystem at process creation.

- **Local User Flag (LUF)**

Although OS/2 security context services can support multiple concurrent users, the OS/2 Presentation Manager (PM) can only support one local user at a time (for example only one local user can interact with PM through the keyboard, mouse, display). Consequently, a process needs to be able to determine whether it can communicate with the user through PM interfaces or not.

The local user flag in the maximum security context specifies whether the process can modify the local user flag in the effective security context or not:

Maximum LUF=0: Process/thread cannot modify LUF in effective security context
Maximum LUF=1: Process/thread can modify LUF in effective security context

The local user flag in the effective security context specifies whether the process/thread can communicate with the user through PM interfaces or not:

Effective LUF=0: Process/thread cannot communicate with the user through PM interfaces
Effective LUF=1: Process/thread can communicate with the user through PM interfaces

At process creation, the local user flag will be set (LUF=1) in the maximum and effective security contexts for the following:

- PM/WPS (local system logon)
- Processes created for programs with the /LOCALUSER=YES option specified in SECURE.SYS
- Processes created from a parent process with LUF set (LUF=1) in its effective security context, unless a process is created for a program with the /LOCALUSER=NO option specified in SECURE.SYS

- **Propagate Authority Flag (PAF)**

The propagate authority flag in the maximum security context specifies whether the process can modify the propagate authority flag in the effective security context or not:

Maximum PAF=0: Process/thread cannot modify PAF in effective security context
Maximum PAF=1: Process/thread can modify PAF in effective security context

The propagate authority flag in the effective security context specifies whether a child process will inherit the Agent User/Group/Process Handles and authority flags (APA, SPA, etc.) from the effective security context of the parent process/thread or not:

Effective PAF=0: Child maximum and effective AGH = parent process/thread effective CGH
Child maximum and effective APH = parent process/thread effective CPH
Child maximum and effective AUH = parent process/thread effective CUH

Child maximum and effective ACA = 0
Child maximum and effective APA = 0
Child maximum and effective CLA = 0

Effective PAF=1: Child maximum and effective AGH =
Parent process/thread effective AGH
Child maximum and effective APH = parent process/
thread effective APH
Child maximum and effective AUH = parent process/
thread effective AUH

Child maximum and effective ACA = parent process/
thread effective ACA
Child maximum and effective APA = parent process/
thread effective APA
Child maximum and effective CLA = parent process/
thread effective CLA

At process creation, the propagate authority flag will only be set (PAF=1) in the maximum and effective security contexts for the following:

- Processes created for programs with the /PROPAGATE=YES option specified in SECURE.SYS, for example the propagate authority flag can not be inherited from the parent process

Authority Flags: In the maximum security context, the authority flags indicate whether the process can modify the corresponding authority flags in the effective security context or not. In the effective security context, the authority flags indicate the current security context authority privileges of a process/thread for example whether the process has access control authority privileges or not.

A process with an authority flag set in its security context can perform the associated privileged operations. Each security control authority role is associated with the privileges it needs to accomplish its role in a secured OS/2 system. Some privileges are associated with more than one SCA role, but there is no defined hierarchy of security control authority roles.

- **Access Control Authority (ACA)**

The access control authority flag in the maximum security context specifies whether the process can modify the access control authority flag in the effective security context or not:

Maximum ACA=0: Process/thread cannot modify ACA flag in effective security context

Maximum ACA=1: Process/thread can modify ACA flag in effective security context

The access control authority flag in the effective security context specifies whether the process has access control authority privileges or not:

Effective ACA=0: Process/thread does not have ACA privileges

Effective ACA=1: Process/thread has ACA privileges

ACA privileges enable a process/thread to do the following:

- Switch the effective user handle to either client or agent
- Switch the effective group handle to either client or agent
- Switch the effective process handle to either client or agent
- Receive notification when subject handles are created or deleted

- **Agent Program/Process Authority (APA)**

The agent process authority flag in the maximum security context specifies whether the process can modify the agent process authority flag in the effective security context or not:

Maximum APA=0: Process/thread cannot modify APA flag in effective security context

Maximum APA=1: Process/thread can modify APA flag in effective security context

The agent process authority flag in the effective security context specifies whether the process has agent process authority privileges or not:

Effective APA=0: Process/thread does not have APA privileges

Effective APA=1: Process/thread has APA privileges

Agent process authority privileges enable a process/thread to do the following:

- Switch the effective user handle to either client or agent
- Switch the effective group handle to either client or agent
- Switch the effective process handle to either client or agent

- **Client Logon Authority (CLA)**

The client logon authority flag in the maximum security context specifies whether the process can modify the client logon authority flag in the effective security context or not:

Maximum CLA=0: Process/thread cannot modify CLA flag in effective security context

Maximum CLA=1: Process/thread can modify CLA flag in effective security context

The client logon authority flag in the effective security context specifies whether the process has client logon authority privileges or not:

Effective CLA=0: Process/thread does not have CLA privileges

Effective CLA=1: Process/thread has CLA privileges

Client logon authority privileges enable a process/thread to do the following:

- Switch the effective user handle to either client or agent
- Switch the effective group handle to either client or agent
- Switch the effective process handle to either client or agent
- Access the name and token of the local system logon user
- Create and assign instance handles for a client process

• Remote Logon Authority (RLA)

The remote logon authority flag in the maximum security context specifies whether the process can modify the remote logon authority flag in the effective security context or not:

Maximum RLA=0: Process/thread cannot modify RLA flag in effective security context

Maximum RLA=1: Process/thread can modify RLA flag in effective security context

The remote logon authority flag in the effective security context specifies whether the process has remote logon authority privileges or not:

Effective RLA=0: Process/thread does not have RLA privileges

Effective RLA=1: Process/thread has RLA privileges

Remote logon authority privileges enable a process/thread to do the following:

- Switch the effective user handle to either client or agent
- Switch the effective group handle to either client or agent
- Switch the effective process handle to either client or agent
- Create subject handles
- Set its client/agent user/group/process handles to subject handles it creates and -1

- **System Logon Authority (SLA)**

The system logon authority flag in the maximum security context specifies whether the process can modify the system logon authority flag in the effective security context or not:

Maximum SLA=0: Process/thread cannot modify SLA flag
in effective security context
Maximum SLA=1: Process/thread can modify SLA flag
in effective security context

The system logon authority flag in the effective security context specifies whether the process has system logon authority privileges or not:

Effective SLA=0: Process/thread does not have SLA privileges
Effective SLA=1: Process/thread has SLA privileges

System logon authority privileges enable a process/thread to do the following:

- Switch the effective user handle to either client or agent
- Switch the effective group handle to either client or agent
- Switch the effective process handle to either client or agent
- Create subject handles
- Set its client/agent user/group/process handles to any valid handle including -1 and 0
- Establish the security context for local system logon
- Establish the security context for SCAs specified in SECURE.SYS
- Establish the security context for processes not associated with local system logon

- **Server Program/Process Authority (SPA)**

The server process authority flag in the maximum security context specifies whether the process can modify the server process authority flag in the effective security context or not:

Maximum SPA=0: Process/thread cannot modify SPA flag in
effective security context
Maximum SPA=1: Process/thread can modify SPA flag in
effective security context

The server process authority flag in the effective security context specifies whether the process has server process authority privileges or not:

Effective SPA=0: Process/thread does not have SPA privileges
Effective SPA=1: Process/thread has SPA privileges

Server process authority privileges enable a process/thread to do the following:

- Switch the effective user handle to either client or agent
- Switch the effective group handle to either client or agent
- Switch the effective process handle to either client or agent
- Set its client user/group/process handles to the effective user/group/process handles of a client
- Reserve a client's user/group/process handles in a list of handles it is allowed to set
- Release a client's user/group/process handles from its list of allowed handles

- **User Identification Authority (UIA)**

The user identification authority flag in the maximum security context specifies whether the process can modify the user identification authority flag in the effective security context or not:

Maximum UIA=0: Process/thread cannot modify UIA flag in effective security context

Maximum UIA=1: Process/thread can modify UIA flag in effective security context

The user identification authority flag in the effective security context specifies whether the process has user identification authority privileges or not:

Effective UIA=0: Process/thread does not have UIA privileges

Effective UIA=1: Process/thread has UIA privileges

User identification authority privileges enable a process/thread to do the following:

- Switch the effective user handle to either client or agent
- Switch the effective group handle to either client or agent
- Switch the effective process handle to either client or agent
- Identify and authenticate a user for local system logon session events

A process that doesn't have any of the above security enabling services authority flags set in its security context status is referred to as a unprivileged process authority.

The following additional authority flags are reserved for security enabling services daemon processes only:

- SES (Security Enabling Services)
- PSS (Protected Shell Services)

9.2.3.5 Security Context Creation

The maximum security context and the initial effective security context of a process is established at program execution (process creation). The subject handles (CUH, AUH, CGH, AGH, CPH, APH) and status flags (ACA, APA, EGF, EPF) in the maximum and effective security context of a newly created process are determined from the following sources:

1. Inherited from the parent process
2. Specified in the SECURE.SYS file
3. Specified by the SLA in response to a process creation event
4. Specified by the ISS security kernel in response to a Pre-ExecPgm callout

Please note that the security enabling services APIs do not support modification of the maximum security context for a process after it is established at process creation.

Also please note that new threads do not inherit the effective security context of the creating thread, they inherit the effective security context of the process (which is maintained by thread-1 of the process). When a newly created thread first enters the SES device driver (via any SES API or KPI), the security context pointer in its trusted computing base will be set to the same security context pointer that thread-1 of the process has in its trusted computing base and, consequently, the newly created thread will have the same effective security context as the process (default process security context model) until it explicitly requests its own separate effective security context (optional thread security context model).

Security Context Inherited from Parent Process: The client user/group/process handles are inherited from the parent process/thread. The agent user/group/process handles can be inherited from the parent process/thread if it has the propagate authority flag set in its effective security context. All authority flags can be inherited from the parent process/thread if it has the propagate authority flag set in its effective security context. The local user flag is inherited from the parent process/thread (independent of propagate authority flag), unless overridden by the /LOCALUSER option in SECURE.SYS. EUF/EGF/EPF are set if any of the authority flags are set.

Note: Propagate authority flag cannot be inherited from the parent process/thread, it must be specified by the /PROPAGATE=YES option in SECURE.SYS.

The default security context inheritance model is that a child process does not inherit any privileges (authority) from the parent process. A more

POSIX-like security context inheritance model is an option that can be specified in SECURE.SYS for specific programs. This is necessary because existing OS/2 programs may need to execute with agent process authority privileges, and these existing programs won't set the authority appropriately in the effective security context before invoking a child process (one of the pains in moving from an unsecured OS/2 environment to a secured OS/2 environment).

Default Policy

Child processes do not inherit agent subject handles or authority flags from the parent process. This is the default policy or it can be specified for a program by the /PROPAGATE=NO option in SECURE.SYS. When a program with the default inheritance policy specified is executed, the propagate authority flag will be set to zero (PAF=0) in the maximum and effective security contexts of the process.

Note: The state flags (EGF, EPF, EUF, LUF, PAF) are not affected by this policy. At process creation, EGF/EPF/EUF will be set if the process has any authority flags set (APA, SPA, etc.), local user flag will be inherited from the parent process unless overridden by the /LOCALUSER option in SECURE.SYS, and PAF can only be set by the /PROPAGATE=YES option in SECURE.SYS.

The intent of this option is to enforce the safest inheritance policy for existing applications that were not developed as trusted programs, but need to execute with agent process authority privileges.

Parent Process	==>	Child Process
Effective CUH=U1	==>	Maximum and Effective CUH=U1
Effective AUH=U2	==>	Maximum and Effective AUH=U1
Effective CGH=G1	==>	Maximum and Effective CGH=G1
Effective AGH=G2	==>	Maximum and Effective AGH=G1
Effective CPH=P1	==>	Maximum and Effective CPH=P1
Effective APH=P2	==>	Maximum and Effective APH=P1
Effective ACA=X	==>	Maximum and Effective ACA=0
Effective APA=Y	==>	Maximum and Effective APA=0
Effective CLA=Z	==>	Maximum and Effective CLA=0

Optional Policy

Child processes may inherit agent subject handles and authority flags from the parent process. This policy can be specified for a program by the /PROPAGATE=YES option in SECURE.SYS. When a program with this optional inheritance policy specified is executed, the propagate authority flag will be set to one (PAF=1) in the maximum and effective security contexts of the process.

Note: The state flags (EGF, EPF, EUF, LUF, PAF) are not affected by this policy. At process creation, EGF/EPF/EUF will be set if the process has any authority flags set (APA, SPA, etc.), local user flag will be inherited from the parent process unless overridden by the /LOCALUSER option in SECURE.SYS, and PAF can only be set by the /PROPAGATE=YES option in SECURE.SYS.

The intent of this option is to enable an installable security subsystem to enforce a more POSIX-like security context inheritance model. For example a parent process with the propagate authority flag set in its effective security context can propagate the agent user/group/process handles and authority flags (APA, SPA, etc.) in its effective security context to a child process:

Parent Process	==>	Child Process

Effective CUH=U1	==>	Maximum and Effective CUH=U1
Effective AUH=U2	==>	Maximum and Effective AUH=U2
Effective CGH=G1	==>	Maximum and Effective CGH=G1
Effective AGH=G2	==>	Maximum and Effective AGH=G2
Effective CPH=P1	==>	Maximum and Effective CPH=P1
Effective APH=P2	==>	Maximum and Effective APH=P2

Effective ACA=X	==>	Maximum and Effective ACA=X
Effective APA=Y	==>	Maximum and Effective APA=Y
Effective CLA=Z	==>	Maximum and Effective CLA=Z

A process with the propagate authority flag set in its maximum security context may still control what the child inherits by setting the process/thread effective security context authority flags to zero (for authority

flags that should not be inherited) before spawning the child.

Security Context Specified in SECURE.SYS: All authority/state flags can be specified in SECURE.SYS. Authority flags are specified directly by the appropriate options (/ACA, /APA, etc.). Propagate authority flag and local user flags are specified directly via the /PROPAGATE=YES and /LOCALUSER=YES options. EUF/EGF/EPF are set if any of the authority flags are set.

Security Context Specified by SLA: In response to a process creation event, the ISS security daemon (SLA) can potentially modify the maximum effective security context of a process when it is created. The SLA's opportunity to modify the security context of a child process (when it is created) depends on whether the process is associated with the local system logon session or not.

- Processes associated with the local system logon session

Any process that inherits its security context from SESShell (PM) is considered to be associated with the local system logon session, for example, any applications started by the local system logon user. This includes detached processes that are started from the local system logon session, but does not include CALL= and RUN= processes that are started during CONFIG.SYS processing or any offspring of these processes (even if they are started during a local system logon session).

For processes that are associated with the local system logon session, a process creation event will be routed to the system logon authority to modify the security context of a child process when it is created if any authority flags are specified in SECURE.SYS or are inherited from the parent process.

- Processes not associated with the local system logon session

Any process that does not inherit its security context from SESShell (PM) is not considered to be associated with the local system logon session. This includes OS/2 system processes (such as the hard error daemon), all CALL= and RUN= processes that are started during CONFIG.SYS processing, and any offspring of these processes (even if they are started during a local system logon session).

For processes that are not associated with the local system logon session, a Process Creation event will be routed to the system logon authority to modify the security context of a child process when it is created whether or not any authority flags are specified in SECURE.SYS or are inherited from the parent process.

Note: For processes initiated before the SLA daemon registers to receive process creation events, the client/agent user/group/process handles will initially be set to zero. A private flag in the security context of these processes will be set to indicate that their subject handles haven't been assigned by the system logon authority yet.

After the system logon authority registers to receive process creation events, when a process with this private flag set in its security context requests any services that result in security enabling services being invoked (for example, the process calls a security enabling services API, or the process opens a file and the installable security subsystem security kernel calls a security enabling services KPI to get the security context of the process). This private flag will be detected and a process creation event will be routed to the system logon authority to modify the security context of the process before the SES API/KPI is allowed to continue.

If a process creation event is routed to the system logon authority to modify the security context of a child process when it is created, the system logon authority can specify the client/agent user/group/process handles and agent process authority flag in the maximum and effective security contexts, and can specify EUF/EGF/EPF in the effective security context.

Specified by SLA	==>	Child Process
CUH=U1	==>	Maximum CUH=U1, Effective CUH=U1
AUH=U2	==>	Maximum AUH=U2, Effective AUH=U2
CGH=G1	==>	Maximum CGH=G1, Effective CGH=G1
AGH=G2	==>	Maximum AGH=G2, Effective AGH=G2
CPH=P1	==>	Maximum CPH=P1, Effective CPH=P1
APH=P2	==>	Maximum APH=P2, Effective APH=P2
APA=X	==>	Maximum APA=X, Effective APA=X

If all SCA flags are set to zero and the SLA specifies APA=0:

EGF=G	==>	Maximum EGF=0, Effective EGF=0
EPF=P	==>	Maximum EPF=0, Effective EPF=0
EUF=U	==>	Maximum EUF=0, Effective EUF=0

If any SCA flags are set to one or the SLA specifies APA=1:

EGF=G	==>	Maximum EGF=1, Effective EGF=G
EPF=P	==>	Maximum EPF=1, Effective EPF=P

EUF=U ==> Maximum EUF=1, Effective EUF=U

Security Context Specified by ISS Security Kernel: In response to a Pre-ExecPgm callout, the installable security subsystem security kernel can specify the client/agent user/group/process handles and agent process authority flag in the maximum and effective security contexts, and can specify EUF/EGF/EPF in the effective security context.

Specified by ISS ==> Child Process

CUH=U1	==>	Maximum CUH=U1, Effective CUH=U1
AUH=U2	==>	Maximum AUH=U2, Effective AUH=U2
CGH=G1	==>	Maximum CGH=G1, Effective CGH=G1
AGH=G2	==>	Maximum AGH=G2, Effective AGH=G2
CPH=P1	==>	Maximum CPH=P1, Effective CPH=P1
APH=P2	==>	Maximum APH=P2, Effective APH=P2

APA=X	==>	Maximum APA=X, Effective APA=X
-------	-----	--------------------------------

If all SCA flags are set to zero and the ISS specifies APA=0:

EGF=G	==>	Maximum EGF=0, Effective EGF=0
EPF=P	==>	Maximum EPF=0, Effective EPF=0
EUF=U	==>	Maximum EUF=0, Effective EUF=0

If any SCA flags are set to one or the ISS specifies APA=1:

EGF=G	==>	Maximum EGF=1, Effective EGF=G
EPF=P	==>	Maximum EPF=1, Effective EPF=P
EUF=U	==>	Maximum EUF=1, Effective EUF=U

9.2.3.6 Security Context Modification

The maximum security context of a process (and all threads of a process) is set at process creation and cannot be modified via the security enabling services APIs.

The security enabling services APIs can only be used to modify the effective security context of a process/thread.

All requested modifications to the effective security context of a process/thread via security enabling services APIs are granted based on the authority specified in the maximum security context of the requesting process.

Note: The system logon authority is not notified (via a security enabling services event) of modifications to the effective security context of a process/thread and, consequently, the system logon authority cannot control modifications to the effective security context of a process/thread.

The system logon authority is only notified (via a security enabling services event) when the maximum and effective security contexts for a process are being initialized. The system logon authority can control the maximum security context and initial effective security context of a process.

9.2.4 SCS Programming Interfaces

Security control services provides an application programming interface (API) and a kernel programming interface (KPI) to support an installable security subsystem at the application level (Ring 3) and the kernel level (Ring 0).

9.2.4.1 SCS API

The security control services API is a 32-bit application level interface that enables security applications (for example, the ISS security daemon) to invoke security control services functions through the security enabling services DLL. The following sections provide a high-level representation of the security control services API.

Context Status Management

```
SESQueryContextStatus( input=PID; output=ContextStatus )  
SESSetContextStatus( input=ContextStatus )
```

Subject Handle Management

```
SESCreateHandleNotify( output=SubjectInfo )  
SESCreateInstanceHandle( input/output=SubjectHandle )  
SESCreateSubjectHandle( input/output=SubjectInfo )  
SESDeleteHandleNotify( output=SubjectHandle )  
SESDeleteSubjectHandle( input=SubjectHandle )  
SESQuerySubjectHandle( input=PID,TargetSubject; output=SubjectHandle )  
SESReleaseSubjectHandle( input=TargetSubject,SubjectHandle )  
SESReserveSubjectHandle( input=TargetSubject )  
SESSetSubjectHandle( input=TargetSubject,SubjectHandle )
```

Subject Info Management

```

SESQuerySubjectHandleInfo( input=SubjectHandle; output=SubjectInfo )
SESQuerySubjectInfo( input=PID,TargetSubject; output=SubjectInfo )
SESSetSubjectInfo( input=TargetSubject; input/output=SubjectInfo )

```

Security Context Management

```

SESQuerySecurityContext( input=PID; output=SecurityContext )
SESResetThreadContext( input=TargetContext )
SESSetSecurityContext( input=SecurityContext )

```

Inter-Process Communication

```

SESQueryAuthorityID( input=AuthorityTag; output=AuthorityID)
SESSendSecurityContext(input=AuthorityID,Timeout; input/output=Message,MessageLength)

```

Process Management

```

SESControlProcessCreation( input=ActionCode )
SESKillProcess( input=PID )
SESQueryProcessInfo( input=ActionCode,SubjectHandle,ProcessBuf;
input/output=ProcessCount )

```

Integrity Violation Log

```

SESlogIntegrityViol( input=logFlag,logData )

```

9.2.4.2 Security Control Services KPI

The Security Control Services KPI is a 32/16-bit kernel level interface that enables the Installable Security Subsystem Security kernel to do the following:

- Invoke security control services functions from Ring-0 (through the security kernel services security helper services)
- Receive callouts when security control services APIs are invoked (through the security kernel services security event router)

Note: The callouts for security control services APIs cannot be used to control access to the services they can only be used to audit the security control services APIs.

The installable security subsystem security kernel registers for the security control services callouts and security helpers through the DevHlp_Security services provided by security kernel services. The DevHlp_Security services are described in the security kernel services system design section, which also includes descriptions of the callouts for security control services API audit. The following sections provide a high-level representation of the security control services security helpers.

Context Status Management

```
SecHlpQueryAuthorityID( input=PID,TID; output=AuthorityID )  
SecHlpQueryContextStatus( input=PID,TID; output=ContextStatus )  
SecHlpSetContextStatus( input=PID,TID,ContextStatus )
```

Subject Handle Management

```
SecHlpQuerySubjectHandle( input=PID,TID,TargetSubject; output=SubjectHandle )  
SecHlpReleaseSubjectHandle( input=AuthorityID,TargetSubject,SubjectHandle )  
SecHlpReserveSubjectHandle( input=AuthorityID,TargetSubject,SubjectHandle )  
SecHlpSetSubjectHandle( input=PID,TID,TargetSubject,SubjectHandle )
```

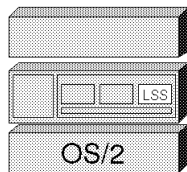
Subject Info Management

```
SecHlpQuerySubjectHandleInfo( input=SubjectHandle; output=SubjectInfo )  
SecHlpQuerySubjectInfo( input=PID,TID,TargetSubject; output=SubjectInfo )
```

Security Context Management

```
SecHlpQuerySecurityContext( input=PID,TID; output=SecurityContext )  
SecHlpResetThreadContext( input=PID,TID,TargetContext )  
SecHlpSetChildSecurityContext( input=SecurityContext )  
SecHlpSetSecurityContext( input=PID,TID,SecurityContext )
```

9.3 Logon Shell Services



Logon shell services enables the many specialized components comprising a secured OS/2 system to work together. Logon shell services defines operations such as logon, logoff, lock, and unlock as events, and provides the function necessary to route the events to participating components. This is how logon shell services facilitates the perception of a single signon and enables a consistent user identification and authentication policy.

The simple definition of a logon shell services event is the complete series of operations and flows that result from a single `SESStartEvent()` call until the results are returned to the calling process. The logon shell services events that can be invoked via `SESStartEvent()` include the following:

- Logon

- Logoff
- Shutdown
- Lock
- Unlock
- Change password
- Create user profile
- Delete user profile
- Identification and authentication

In addition, two of the security context services functions are implemented as pseudo-events. These pseudo-events are triggered by security control services functions (documented in the previous section).

- Send security context
- Process creation

Each event requires communication between security enabling services and the cooperating components (SLA, UIAs, CLAs, SPAs). As the processing of a logon shell services event proceeds, the participating components are dispatched to perform various tasks (identification and authentication, logon, system shutdown, etc.). Each of the participating security components needs to understand the state of the logon shell services event and what task they are being asked to perform. This is especially important for security components that can perform multiple tasks for the same event (for example, one process of a security component could be an SLA/UIA/CLA and be dispatched three times for the same local system logon event).

9.3.1 Key Components

The following diagram shows the key components that constitute the logon shell services state machine and the relevant components of a secured system.

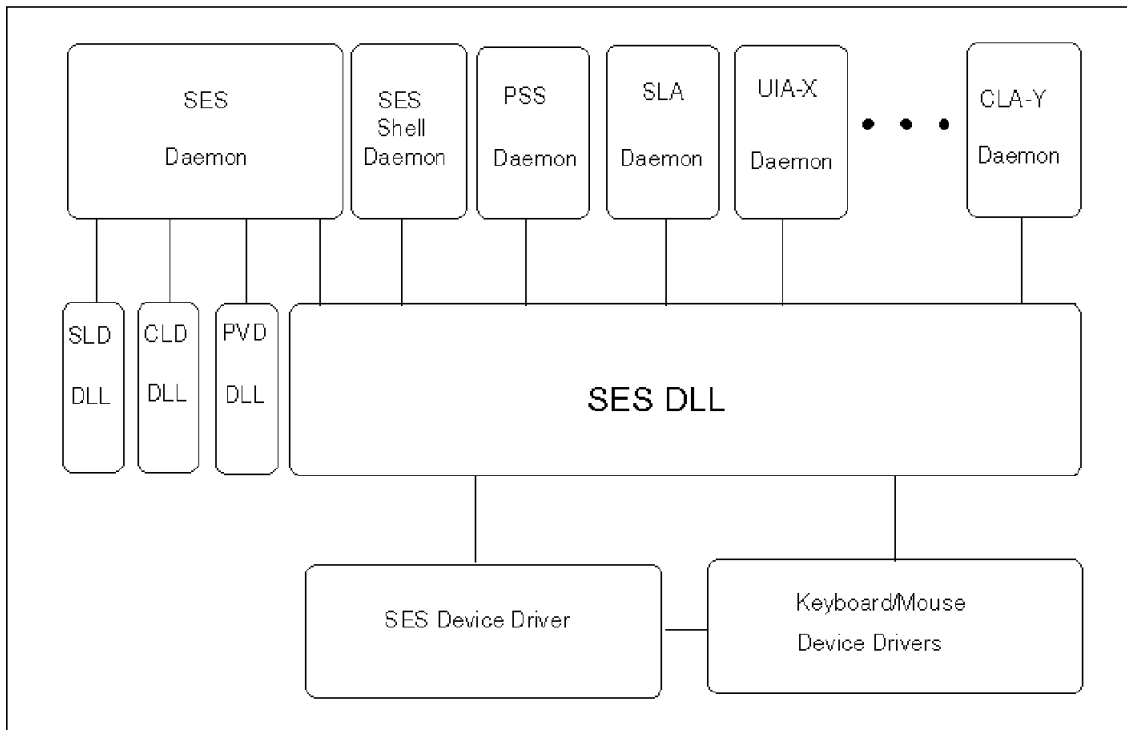


Figure 22. LSS Architecture - Interaction between LSS and Security Applications

The following sections describe these key components and the cooperating security application components that participate in logon shell services events.

9.3.1.1 Security Enabling Services Daemons

The security enabling services daemon is the engine for all logon shell services functions and event flows. It drives a state machine for each event, coordinating the order and target of each event notification. The PSS daemon monitors user activity for logon/unlock events and controls the keyboard/mouse/display when the logon shell services state machine is in the lock or logoff states. The SESShell daemon starts PM and initiates/terminates the user shell (for example WPS) for local system logon sessions.

9.3.1.2 User Identification Authority Daemons

The user identification authority daemons have the authority to identify and authenticate a user for local system logon. A user identification authority must verify that the users attempting to access the local system are who they claim to be. For example, users can be authenticated by passwords, signatures, fingerprints, etc.

9.3.1.3 System Logon Authority Daemon

The system logon authority daemon interacts with logon shell services to provide the security context for local system logon after identification and authentication has occurred. After the user identification authorities have completed identification and authentication the resulting status (for example, authenticated, not authenticated, guest, error) is passed to the system logon authority. The system logon authority makes the final decision for local system logon. For example, even though a user is authenticated by a user identification authority, the system logon authority might not have the user in the SLA's list of authorized users in this case, the system logon authority might only allow the user to log on as a guest.

9.3.1.4 Client Logon Authority Daemons

The client logon authority daemons can receive notification of local system events such as logon, logoff, shutdown, and change password. A client logon authority is also authorized to obtain the password that the user entered during local system logon. Given the user's local name and password (and any additional CLA-specific information about the user, for example, LAN Server domain and user ID), the client logon authority may be able to log the user on to remote services without additional intervention by the user. The objective is to enable transparent identification and authentication for remote services, thus creating the perception of single signon.

9.3.1.5 Security Enabling Services Device Driver

The logon shell services component of the SES device driver provides kernel level services for the security enabling services daemon. These services control (under the direction of the security enabling services daemon) the blocking and unblocking of captive threads of the various cooperating component processes (daemons), and also provide a central point for transfer of event specific data between processes.

9.3.1.6 SES Dynamic Link Library

The security enabling services DLL includes the logon shell services APIs that are used by an security control authority (such as SLA, CLA, UIA, SPA) daemon to participate in logon shell services events:

- SESRegisterDaemon
- SESReturnEventStatus
- SESReturnWaitEvent
- SESStartEvent
- SESWaitEvent

9.3.1.7 SLD/CLD/PVD Dynamic Link Libraries

The following additional security enabling services components provide the policy for logon shell services interaction with user identification authorities, client logon authorities, and a password validation service during the processing of logon shell services events.

System Logon Driver (SLD):	Determines which user identification authorities participate (and in what order) for the logon, unlock, identification and authentication, change password, create user profile and delete user profile events.
Client Logon Driver (CLD):	Determines which client logon authorities participate (and in what order) for the logon, logoff, lock, unlock, change password, create user profile and delete user profile events.
Password Validation Driver (PVD):	Checks password rules (such as composition, history, dictionary) for the change password and create user profile events.

These drivers are implemented as DLLs that contain the functions invoked by the security enabling services daemon during the processing of logon shell services events:

- SLDInit
- SLDQueryUIA
- CLDInit
- CLDQueryCLA
- PVDValidatePassword

Security enabling services includes a rudimentary implementation of these three DLLs; but, these DLLs may be replaced by independent software vendors or customers to satisfy more complex security requirements.

9.3.1.8 Keyboard/Mouse Device Drivers

The keyboard/mouse device drivers detect inactivity (keystroke or mouse button) for a specified time period independent of the state of the user interface services (for example, without regard to which screen group is active). The keyboard/mouse device drivers support an logon shell services API that specifies the inactivity time period and blocks (captive thread) until the device drivers detect inactivity for the specified time period.

The keyboard device driver detects the trusted path key combination (Ctl-Alt-Del or Ctl-Alt-NumLock-NumLock) independent of the state of the user interface services (for example, without regard to which screen group is active). The keyboard device driver supports a callout to the installable security subsystem security kernel when the trusted path key combination is detected, and also supports an logon shell services API to control keyboard monitors so that they cannot intercept keystrokes when the installable security subsystem is providing trusted path services.

9.3.2 Definition of Local System Logon

The OS/2 user interface services, Presentation Manager (PM) and Workplace Shell (WPS), are inherently single-user (serial multi-user). Consequently, the logon shell services definition of local system logon is the association of one user with the local PM/WPS user interface services.

Note: However, the OS/2 security context services supports concurrent multi-user operation. For example, one user could access the system through the local keyboard/mouse/display at the same time that multiple remote users could access the system through a dial-in facility.

Prior to local system logon, PM (and the user shell if RESTARTUSERSHELL=NO) will be associated with the logoff state security context. The default logoff state security context is:

Client user handle	= -1 (unauthenticated user)
Agent user handle	= -1
Client group handle	= -1
Agent group handle	= -1
Client process handle	= -1
Agent process handle	= -1
Security context status	= 0 (all state/authority flags=0)

Note: The system logon authority may specify the logoff state security context when the system logon authority registers with security enabling services. See logon shell services system design for details.

After local system logon, the security context associated with the local system logon session (PM process, user shell process, and all untrusted processes started by the local system user) depends on the logon state:

- **Explicit Logon State**

After an explicit local system logon, the security context associated with the local system logon session is established during the logon event:

Client user handle	= reserved by SES, specified by SLD/UIAs
Agent user handle	= client user handle
Client group handle	= specified by the SLA
Agent group handle	= client group handle
Client process handle	= specified by the SLA
Agent process handle	= client process handle
Security context status	= LUF (LUF=1, all other state/authority flags=0)

- **Auto-Guest Logon State**

After an auto-guest local system logon, the security context associated with the local system logon session is set equal to the auto-guest logon state security context (see the definition of auto-guest in the next section). For example, the system logon authority does not establish the security context for the local system logon session for an auto-guest logon.

After local system logoff, PM (and the user shell if RESTARTUSERSHELL=NO) will again be associated with the logoff state security context.

9.3.3 Definition of a Guest User

Security enabling services configuration includes specification of a subject name to be associated with the client user handles for all unauthenticated users. This is referred to as the unauthenticated user name or guest name, and may be specified by the GUESTNAME environment variable.

The default GUESTNAME is GUEST.

For access control authority, the logon shell services definition of a guest user is any user whose subject name is the same as the guest name. This definition of a guest user is sufficient to enable an access control authority to apply the appropriate access control policy. For example the access control

authority doesn't need any additional information about the subject handle associated with a guest user because the access control authority implicitly accepts the user authentication accomplished by user identification authority during local system logon (which, for a guest user, probably resulted in no authentication).

However, this definition of a guest user is not sufficient for client logon authority because the client logon authority does not implicitly accept the user authentication accomplished by user identification authorities during local system logon. Client logon authorities must be able to distinguish one guest user from another for example each guest user must have a unique security enabling services subject handle. Consequently, when a user explicitly logs on as a guest user, a unique client user handle will be created and the security enabling services subject name associated with the client user handle will be set to whatever is specified in the GUESTNAME environment variable.

Note: Client logon authorities are provided the name and token (password) presented by the local system logon user for identification and authentication, even if the user is a guest (unauthenticated) user.

- **Explicit logon as a guest user**

During the normal logon flow, the result of the identification and authentication process may be that the user is not authenticated, either because the user identification authorities failed to authenticate the user or because the user explicitly chose to bypass identification and authentication by logging on as a guest user. After the authentication status of the user is determined and this information is provided to the system logon authority, the system logon authority can allow an unauthenticated user to explicitly logon as a guest user.

Please note that in the case of an unauthenticated user explicitly logging on as a guest user, the standard logon process occurs including creation of a unique security enabling services subject handle for the client user handle for the guest user even though the security enabling services subject name associated with the client user handle will still be set to whatever is specified in the guestname environment variable.

Resulting CUH when a user explicitly logs on as a guest:

```
Handle = Unique value for each guest user logon session
Name   = String specified in GUESTNAME environment variable
Token  = Null
Source = 0
```

Consequently, client logon authorities can distinguish one guest user from another and can associate the appropriate client logon authority user/group/process credentials with the unique client user handle created for the guest user.

- **Definition of Autoguest**

When the auto-guest feature is selected, the entire user identification authority identification and authentication process is bypassed and the user shell is started. In this case, the client user handle is set to -1 (the special security enabling services subject handle reserved for an unauthenticated user) which is associated with the security enabling services subject name specified in the guestname environment variable.

Resulting CUH for Auto-guest logon:

```
Handle = -1
Name   = String specified in GUESTNAME environment variable
Token  = Null
Source = 0
```

For an auto-guest logon, the security context returned by the system logon authority is ignored and the local user logon session security context is set equal to the auto-guest logon state security context:

```
Client user handle      = -1
Agent user handle       = -1
Client group handle     = -1
Agent group handle      = -1
Client process handle   = -1
Agent process handle    = -1
Security context status = LUF (LUF=1, all other state/authority flags=0)
```

Since one auto-guest user cannot be distinguished from another, client logon authorities should not allow an auto-guest user to logon as a client. If an auto-guest user attempts to access client logon authority resources/services, the client logon authority should prompt the user to explicitly logon (either as an authenticated user or as a guest user).

Note: The installable security subsystem should provide a convenient user interface to enable an auto-guest user to initiate an explicit logon (implicit logoff) either as an authenticated user or as an explicit unauthenticated guest user.

In either case (auto-guest logon or explicit logon as a guest user), the security enabling services subject name associated with the client user handle will be the guest name specified in the GUESTNAME environment variable.

9.3.4 Overview of Keyboard/Mouse Support

Logon shell services provides trusted path support, keyboard/mouse activity detection to instigate logon/unlock events, and keyboard/mouse inactivity detection to automatically lock the user interface services after a specified time period.

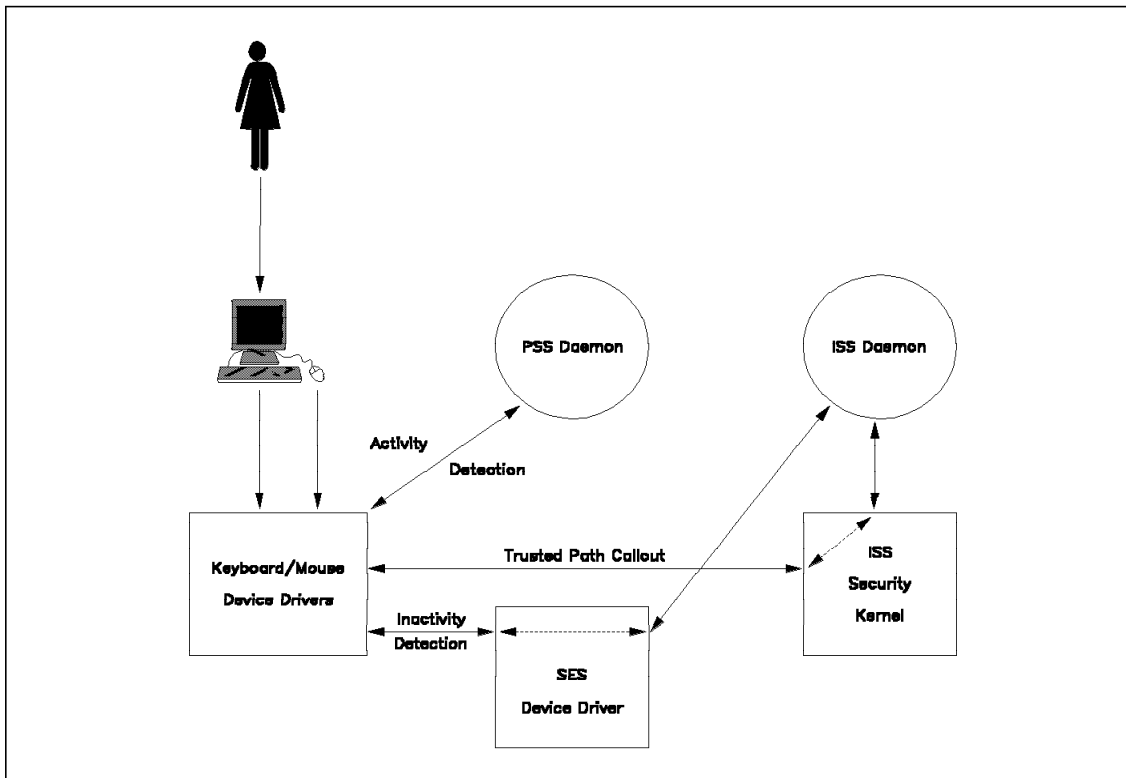


Figure 23. LSS Architecture - Overview of Keyboard/Mouse Support

Logon shell services works with the keyboard/mouse device drivers to detect three keyboard/mouse input conditions, independent of the state of the user interface services (for example, what screen group is active).

- Invocation of the trusted path key combination
- Keyboard/mouse activity (in the logoff and lock states)
- Keyboard/mouse inactivity for a specified period of time

Logon shell services also provides functions that enable the installable security subsystem security daemon (SLA) to do the following:

- Control keyboard monitors for trusted path services

- Monitor keyboard/mouse inactivity for automatic lock services

To exploit the logon shell services trusted path support, the installable security subsystem needs both Ring-0 (ISS security kernel) and Ring-3 (ISS security daemon) components. The installable security subsystem security kernel provides the trusted path callout function, which responds to a trusted path callout and directs the keyboard device driver to take the appropriate action. The installable security subsystem security daemon provides trusted path user interface services (for example, turn off keyboard monitors and display a security menu) associated Ring-3 logic. To enable the logon shell services trusted path support, the SET TRUSTEDPATH=YES environment variable must be specified in CONFIG.SYS.

When the logon shell services trusted path support is not enabled and logon shell services is either in the logoff state (after local system logoff event, prior to local system logon event) or in the lock state (after local system lock event, prior to local system unlock event), logon shell services will take control of the OS/2 user interface services and monitor keyboard/mouse input for user activity. When user activity is detected in the logoff/lock states, logon shell services will initiate a logon/unlock event.

When the installable security subsystem security daemon registers for keyboard/mouse inactivity detection and specifies the time period, logon shell services works with the keyboard/mouse device drivers to detect inactivity regardless of the state of the OS/2 user interface services (for example what screen group is active). When no keyboard/mouse activity (keystroke or mouse button) has been detected for the specified time period, the ISS security daemon will be notified so that it can initiate the appropriate logon shell services event (typically lock, potentially logoff or shutdown).

9.3.5 LSS Programming Interfaces

Logon shell services provides an Application Programming Interface (API) and a Kernel Programming Interface (KPI) to support an installable security subsystem at the application level (Ring 3) and the kernel level (Ring 0).

9.3.5.1 Logon Shell Services API

The logon shell services API is a 32-bit application level interface that enables security applications (for example, the ISS security daemon) to invoke logon shell services functions through the security enabling services DLL. The following sections provide a high-level representation of the logon shell services API.

SES Event Management

```
SESRegisterDaemon( input=EventList; output=DaemonNumber )
SESReturnEventStatus( input/output=EventInfo )
SESReturnWaitEvent( input=Timeout; input/output=EventInfo )
SESStartEvent( input/output=StartEventInfo )
SESWaitEvent( input=Timeout; input/output=EventInfo )
```

Keyboard/Mouse Support

```
SESControlKBDMonitors( input=ActionCode; output=Status )
SESinactivityNotify( input=Timeout )
```

9.3.5.2 Logon Shell Services KPI

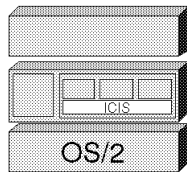
The logon shell services KPI is a 32/16-bit kernel level interface that enables the installable security subsystem security kernel to do the following:

- Receive callouts for trusted path invocation (through the security kernel services security event router)
- Receive callouts when logon shell services APIs are invoked (through the security kernel services security event router)

Note: The callouts for logon shell services APIs cannot be used to control access to the services they can only be used to audit the logon shell services APIs.

The installable security subsystem security kernel registers for the logon shell services callouts through the DevHlp_Security services provided by security kernel services. The DevHlp_Security services are described in the security kernel services system design section, which also includes descriptions of the callouts for logon shell services trusted path invocation and logon shell services API audit.

9.4 Installation, Configuration, Initialization Support



The installation, configuration and initialization support architecture addresses the requirement for secure installation, configuration, and initialization of security enabling services and an installable security subsystem.

9.4.1 Installation

The security enabling services facility makes use of some of the OS/2 components (OS2KRNL, DOSCALL1.DLL, PMWP.DLL, NWIAPI.DLL, MOUSE.SYS, KBD01.SYS, KBD02.SYS). Because SES requires that these components have been modified to include the necessary enhancements to support security enabling services, a pre-requisite fixpak level will be required for some releases of OS/2. For example, for OS/2 Version 2.11, security enabling services is supported from fixpak XR_B100. This fixpak was the first fixpak to include the necessary base code modifications for supporting security enabling services.

The security enabling services installation process will copy several security enabling services files (SES device driver, SES daemon programs, SES dynamic link libraries, etc.) to the appropriate drive/directories. However, the security enabling services installation process will not make the changes to CONFIG.SYS that are necessary to enable security enabling services. When a customer installs a security product that requires security enabling services, the installable security subsystem installation process must include the appropriate modifications to CONFIG.SYS (and SECURE.SYS) to enable security enabling services.

9.4.2 Configuration

Configuration of security enabling services (and other necessary security components such as an installable security subsystem) is accomplished by modifying two files, CONFIG.SYS and SECURE.SYS, to include the appropriate information.

Note: Security enabling services does not include configuration APIs or GUI.

9.4.2.1 CONFIG.SYS

This section describes the security enabling services environment variables and SES/ISS components that are defined in the CONFIG.SYS file.

Most of the security enabling services kernel code is implemented as a base device driver, as is the installable security subsystem security kernel, both of these base device drivers must be defined in CONFIG.SYS (BASEDEV=).

Security enabling services includes three special programs (SESSTART.EXE, SESSHELL.EXE, and PSSDMON.EXE) that must be specified in CONFIG.SYS (CALL=, PROTSHELL=, and RUNWORKPLACE=) so that they are started at the appropriate time during the initialization (boot) process.

The following environment variables must be defined in CONFIG.SYS to configure security enabling services:

AUTOGUEST	The AUTOGUEST environment variable is used to enable or disable auto-guest logon. To enable auto-guest logon, the AUTOGUEST=YES environment variable must be specified. If the AUTOGUEST=NO environment variable is specified or the AUTOGUEST environment variable is not specified, auto-guest logon is not enabled.
BACKGROUNDBITMAP	The BACKGROUNDBITMAP environment variable specifies the bit map to be displayed when no local system user is currently logged on or the local system user interface is locked.
GUESTNAME	The GUESTNAME environment variable defines the name that is associated with all unauthenticated users. The default GUESTNAME is GUEST.
RESTARTUSERSHELL	The RESTARTUSERSHELL environment variable specifies whether the user shell process is killed/restarted or remains running between local system logon sessions. The default is RESTARTUSERSHELL=YES (the user shell process is restarted at each logon and killed at each logoff). If RESTARTUSERSHELL=NO is specified, the user shell is not killed at each logoff and is not restarted at each logon.

SESDBPATH	SECURE.SYS, SES.LOG, and the SLD/CLD/PVD DLLs reside in the directory specified by the SESDBPATH environment variable. SESDBPATH must be specified.
TRUSTEDPATH	The TRUSTEDPATH environment variable enables/disables trusted path detection. If TRUSTEDPATH=YES is specified, SES will not initiate a logon/unlock event when keyboard/mouse activity is detected. If TRUSTEDPATH=NO is specified or the TRUSTEDPATH environment variable is not specified, SES will initiate a logon/unlock event when keyboard/mouse activity is detected.
USERSHELL	The USERSHELL environment variable defines the user interface shell that will be started when a user logs on. If USERSHELL is not defined, WPS (PMSHELL.EXE) is the default user interface shell.

9.4.2.2 SECURE.SYS

Components that require special security enabling services privileges are defined in SECURE.SYS (system logon authority, user identification authority, etc.). See the security control services (security context services) architecture section for a description of security enabling services privileges and SECURE.SYS configuration.

9.4.3 Initialization

The requirement to ensure that security enabling services and installable security subsystem components are initialized (and are able to enforce security policies) prior to allowing a user to logon (and use the workstation) is addressed by:

- Modifying the following backup system configuration files to control the security characteristics of the system that is booted when a user types ALT-F1 during system boot
 - \OS2\INSTALL\CONFIG.SYS
 - \OS2\INSTALL\OS2.INI
 - \OS2\INSTALL\OS2SYS.INI
- Modifying CONFIG.SYS and SECURE.SYS to include the necessary security characteristics:

The required modifications to SECURE.SYS and CONFIG.SYS (original and backup) are discussed in the system design section.

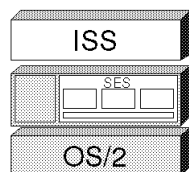
The OS/2 initialization process is as follows. The OS/2 initialization process loads all base device drivers and then all device drivers in the order in which they appear in the CONFIG.SYS file. As a result, the SESDD32.SYS basedev must appear prior to any other vendor's base device drivers in the CONFIG.SYS file. Additionally, for the OS/2 loader to locate basedevs, they must be located in the root directory of the boot drive or in its OS2 subdirectory. The ISS statements appearing in the CONFIG.SYS file should be the next DEVICE= and RUN= statements in CONFIG.SYS (after the SES statements).

After all devices are loaded, the OS/2 initialization process executes the CALL= and RUN= applications in the CONFIG.SYS file in the order in which they appear. Applications that are invoked with the CALL= command execute synchronously (CALL= processes must terminate before OS/2 initialization continues), where as applications that are invoked with the RUN= command execute asynchronously (OS/2 initialization continues without waiting for termination of RUN= processes).

SESSTART.EXE must be the first executable application (CALL= or RUN=) specified in the CONFIG.SYS file and it must be invoked by a CALL= statement to ensure that the SES daemon (SESDMON.EXE) completes initialization prior to the start of any other applications. The function of SESSTART.EXE is merely to start the SES daemon and wait for notification from it that initialization completed. SESSTART.EXE then terminates, which allows the OS/2 initialization process to continue. At this point in the initialization process, the security enabling services are now available.

The recommended method for starting security applications (such as the ISS security daemon) is via the security configuration file, SECURE.SYS. This file contains security enabling services configuration information for security context authority (for example SLA, UIA, CLA) programs. One of the SECURE.SYS configuration options (/START) specifies that a security control authority is to be started by security enabling services. This option ensures that the security control authority is started after PM (so that the security control authority can use the PM user interface services), but before any user is allowed to log on.

Chapter 10. Interoperation of SES and ISS



This section describes how key security enabling services components cooperate to enable an installable security subsystem to provide operating system security. For each set of services (SKS, SCS, LSS, ICIS), the interfaces provided to utilize the services (APIs, KPIs, and/or configuration requirements, as appropriate for each set of services) are defined and scenarios are described to demonstrate the flow of control/data.

The following diagram depicts the key security enabling services components, relevant OS/2 components, and relevant installable security subsystem (and/or other security application) components:

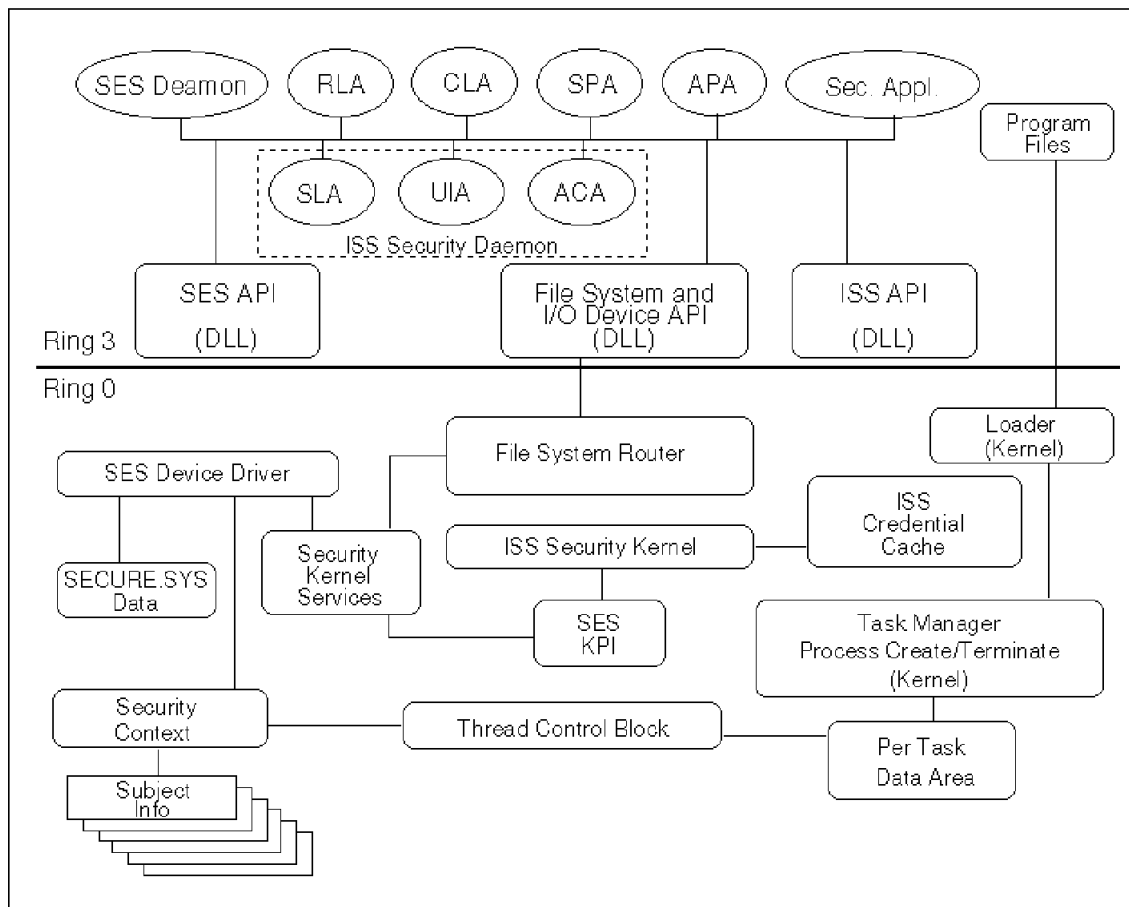


Figure 24. SES System Design - Interoperation of Key Components

Security Application Components

- ACA Daemon: Has the authority to be notified of subject handle creation/deletion.
- APA Process: Has the authority to execute on behalf of a client user and a trusted agent user.
- CLA Daemon: Has the authority to access the name and token (password) of the local system logon user and to create instance handles for a client process.
- RLA Daemon: Has the authority to create/set subject handles.

- SLA Daemon: Has the authority to create/set subject handles and establish the security context for the local system logon user.
- SPA Daemon: Has the authority to execute on behalf of multiple client users and a trusted agent user.
- UIA Daemon: Has the authority to identify and authenticate users local system logon session events.

Installable Security Subsystem (ISS) Components

- ISS API: DLL functions that provide an application programming interface for ISS services.
- ISS Security Daemon(s): Ring-3 component(s) of the ISS are SLA, UIA, CLA, ACA, etc.
- ISS Security Kernel: Ring-0 component of the ISS (implemented as an OS/2 device driver).
- ISS Credential Cache: User/group/process credentials associated with active subject handles (maintained dynamically in memory for fast access by the ISS).

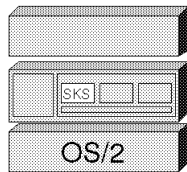
Security Enabling Services (SES) Components

- SES API: DLL functions that provide an application programming interface for SES.
- SES KPI: IDC functions that provide a kernel programming interface for SES.
- SES Daemons: Ring-3 components of SES (SES daemon, PSS daemon, SESShell daemon).
- SES Device Driver: Ring-0 component of SES.
- Security Context: Subject handles and status flags associated with a process/thread.
- Subject Info: Name/token maintained by SES for each subject handle.
- SECURE.SYS: Configuration data, primarily for security context authority.

OS/2 Operating System Components

File System API:	DLL functions to access files and directories. Note: No changes are required to these APIs.
I/O Device API:	DLL functions to access I/O devices. Note: No changes are required to these APIs.
File System Router:	Routes file system requests to the appropriate file system (FAT or IFS) or I/O device. Requests to an IFS are directed to the file system's File System Driver (FSD). Requests to an I/O device are directed to the device's Physical Device Driver (PDD).
Loader:	Loads a program into memory.
Task Manager:	Establishes a program as a process in the system. A Per Task Data Area is allocated for the process and a Thread Control Block is allocated for each thread.
Per Task Data Area:	Data structure that contains process status information.
Thread Control Block:	Data structure that contains thread status information, the pointer to a thread's security contexts (maximum and effective) are in this structure.

Chapter 11. Security Kernel Services (SKS)



The design of security kernel services must enable an installable security subsystem security kernel (device driver) to specify the security events which are of interest to it and receive notification (callouts) when one occurs. In addition, security kernel services must provide a limited set of services (file system access and security context services) for the installable security subsystem security kernel that would otherwise not be available at Ring-0.

Security Kernel Services Installation

The OS/2 components that have been modified to include the necessary enhancements to support security kernel services (OS2KRNL, MOUSE.SYS, KBD01.SYS, KBD02.SYS) will be installed as part of the optional OS/2 installation for security enabling services. However, the security enabling services installation process will not make the changes to CONFIG.SYS that are necessary to enable/exploit security kernel services, the installable security subsystem installation process must include the modifications to CONFIG.SYS to enable/exploit security kernel services (for example, BASEDEV=).

Security Kernel Services Initialization

The OS/2 system boots normally if an installable security subsystem is not present. If an installable security subsystem is present, it loads as a base device driver. Security kernel services is enabled and communication with the installable security subsystem is established during initialization of the installable security subsystem device driver. The installable security subsystem security kernel registers with security kernel services by calling a new OS/2 device helper function: DevHlp_Security.

Security Kernel Services Performance

The router design produces no measurable slowdown to OS/2 when an installable security subsystem is not loaded. The non-secured path in the kernel is optimized, even if it detracts from the performance of the secured path. Additional code and data is minimized; for example, secured data is allocated from the heap so that only secured OS/2 is grown. Any security code that can go in either the OS/2 kernel or the SES/ISS device drivers is put into the device drivers. Security enabling services code is allocated in swappable memory where possible (some of the security enabling services code is not swappable).

11.1 Kernel Callouts Imported from the ISS

An installable security subsystem can call `DevHlp_Security` to register to receive callouts for the following:

- Security Relevant OS/2 System Calls
- Callgate Level Support
- Multiple Virtual DOS Machine Support
- Logon Shell Services Trusted Path Support
- Security Enabling Services API Audit Support

11.1.1 Callouts for Security Relevant OS/2 System Calls

The installable security subsystem security kernel (device driver), during its initialization, makes a `DevHlp_Security` call to the OS/2 kernel requesting notification on specific security related events. The installable security subsystem device driver passes a list of worker or service routines that correspond to each of the supported security events. When one of those specified events occurs, the kernel simply calls the appropriate routine. The installable security subsystem device driver services the notification and returns with the appropriate status condition for the event.

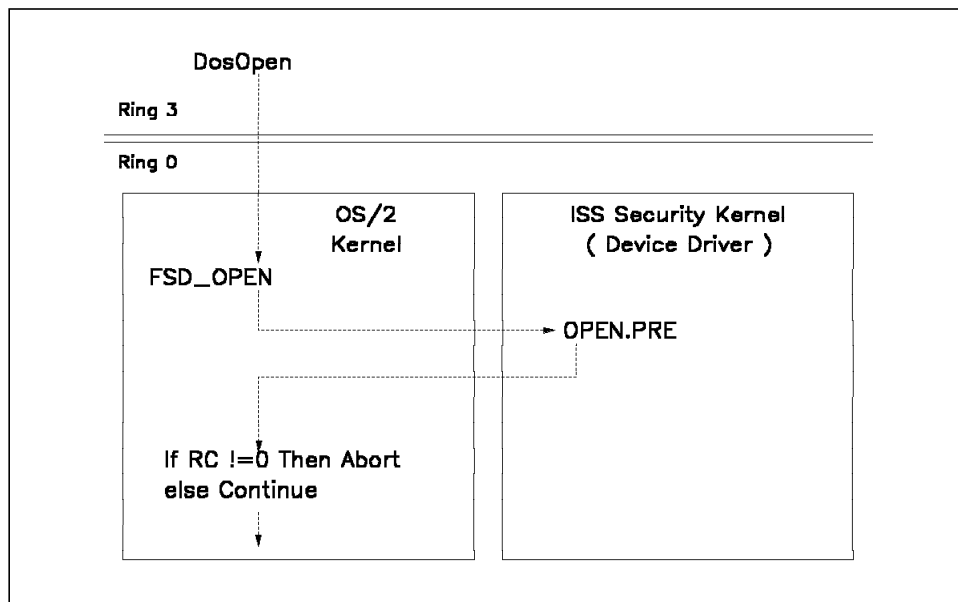


Figure 25. SKS System Design - OS/2 System Call Hooks

11.1.2 Callouts for Callgate Level Support

Callgate level hooks allow an installable security subsystem to receive notification of events that are not covered by specific callouts for file access, process creation. Access to the callgate level hooks is divided into the following categories depending on the type of the OS/2 worker routine to be hooked:

- 16 bit
- 32 bit

Callgate hooks are enabled when an installable security subsystem provides an address for the callgate handler routines (either 16 bit, 32 bit, or both) in the list of imported callouts for the kernel. Once a class (16 bit or 32 bit) is hooked, all kernel calls for this class are routed to the callgate handler. The callgate handler is then responsible for calling the OS/2 worker if the call is allowed to proceed.

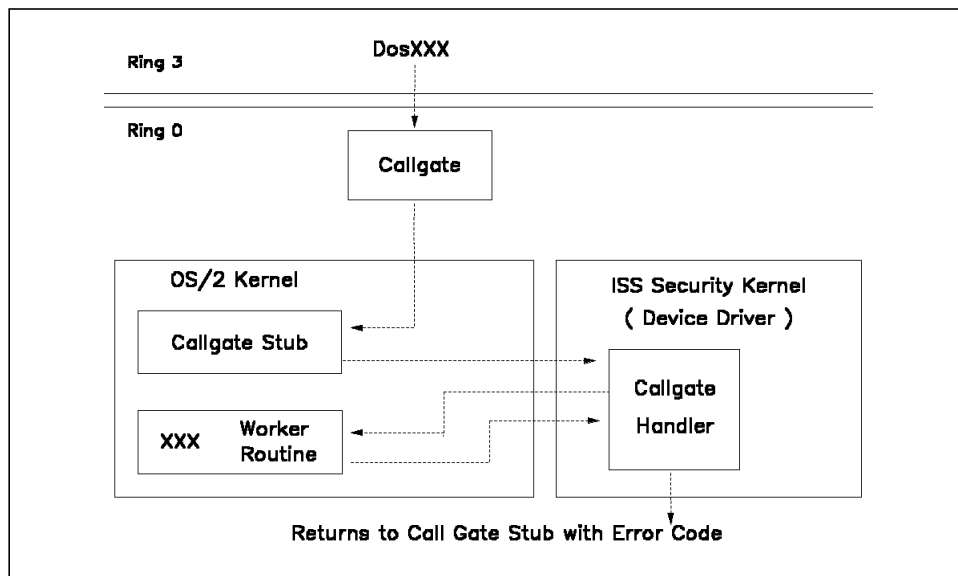


Figure 26. SKS System Design - OS/2 Callgate Level Hooks

The order of events for a callgate hook are:

Application	OS/2 Kernel	ISS Device Driver
Call DosXXX	Route to callgate stub, which calls the callgate handler providing the ordinal number, worker routine, and pointer to the user stack.	
	Do work for DosXXX.	If valid then call OS/2, else return RC now.
	Exit and return RC.	Return RC to callgate stub.
Receives RC from OS/2.		

11.1.3 Callouts for Multiple Virtual DOS Machine Support

The Multiple Virtual DOS Machine (MVDM) dispatch table contains the function code of an MVDM (INT21) request and the address of its worker routine. The table is pointed to by `apDemSvc` which can be obtained from the security export data structure.

Each entry in the MVDM Dispatch Table is a 32-bit flat pointer to a worker routine. The installable security subsystem device driver can replace this

address with the address of its own MVDM handler routine. In this manner, it can hook any desired function. The handler routine receives a pointer in EBX to a structure containing the registers at the time the HALT instruction occurred. To determine what the register-parameter association is at that time, ISVs can either do some reverse engineering by using the kernel debugger or contact IBM for information.

ISS Note that these hooks occur after the OS/2 kernel receives the INT21. This means that a TSR could have hooked the INT21 and the kernel would not know the difference. For example, some LAN redirectors hook the INT21 in the DOS box and do not pass the requests through OS/2 (they take them directly to their virtual device driver). Therefore, installable security subsystem developers should also write a virtual device driver (VDD) to monitor INT21 calls.

The DOS kernel (which handles INT21 calls for a VDM) can make calls directly to the OS/2 kernel. It can also make calls for which there is no corresponding DOS interrupt. It is impossible for even the VDD to catch these calls. Since these calls also bypass the callgate hooks, a way of hooking them needs to be developed for completeness.

As with callgate hooks, everything is subject to change. Kernel data structures and code can be changed in the future, thereby breaking a previously functioning hook or hooks. It is not recommended that these be used. The callouts or hooking the INT21 in the VDM are preferable methods. File system calls from a VDM that are identified as security relevant are hooked with the callouts.

11.1.4 Callouts for Logon Shell Services Trusted Path Support

When the keyboard device driver receives a Ctrl-Alt-Del or Ctrl-Alt-Numlock-Numlock it will call the TrustedPathControl() function provided by the installable security subsystem security kernel. TrustedPathControl() should return an appropriate value to the keyboard device driver, which will then take action based on that return code. If an illegal value is returned, the keyboard device driver will take the action it normally would have taken (for example reboot, in the case of Ctrl-Alt-Del).

11.1.5 Callouts for Security Enabling Services API Audit Support

Callouts are provided for all security control services, APIs so that the installable security subsystem security kernel can audit these functions.

11.2 Kernel Services Exported to the ISS

An installable security subsystem can call `DevHlp_Security` to register for the following security helper functions:

- File System Services
- Security Context Services

11.2.1 Security Helpers for File System Services

The list of security helper functions returned by the `DevHlp_Security` call include the following functions to invoke the following OS/2 file system operations:

- `SecHlpChgFilePtr`
- `SecHlpClose`
- `SecHlpFindNext`
- `SecHlpOpen`
- `SecHlpPathFromSFN`
- `SecHlpQFileSize`
- `SecHlpRead`
- `SecHlpSFFromSFN`
- `SecHlpWrite`

11.2.2 Security Helpers for Security Context Services

The list of security helper functions returned by the `DevHlp_Security` call includes functions to invoke security context services. These functions are discussed in the security context services section.

11.3 Security Kernel Services KPI

Security kernel services provides a new device helper (`DevHlp_Security`) function that enables an installable security subsystem security kernel to register to receive callouts for selected OS/2 kernel events and to invoke security helper (`SecHlp`) functions.

The installable security subsystem security kernel uses the `DevHlp_Security` function to provide security kernel services with the security import data

structure. This structure contains the entry points of functions provided by the installable security subsystem security kernel for selected callouts.

The installable security subsystem security kernel also uses the `DevHlp_Security` function to retrieve from security kernel services the address of the security export data structure. This structure contains the entry points of all security helper functions and the MVDM dispatch table pointer.

On each invocation, `DevHlp_Security` performs one of these two services (import or export addresses) as specified by the function code passed in by the installable security subsystem:

`DHSEC_SETIMPORT`: Set (input to SKS) list of functions to handle callouts
`DHSEC_GETEXPORT`: Get (output from SKS) list of exported security helpers

Full details of the security kernel services KPI functions may be found in Appendix A, “Security Kernel Services KPI Details” on page 239.

11.4 Security Kernel Services Scenarios

The following scenarios are described in this section:

- ISS Security Kernel Initialization
- File System Open Callout

11.4.1 ISS Security Kernel Initialization

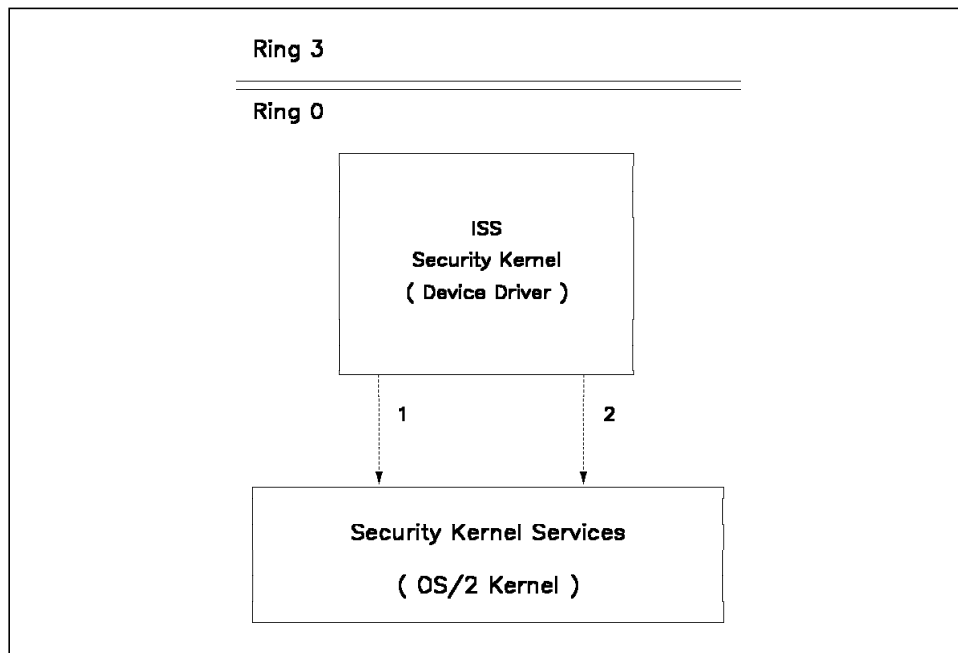


Figure 27. SKS System Design - ISS Initialization Scenario

1. To use the security enabling services security helpers, the installable security subsystem device driver calls `DevHlp_Security()` with the `DHSEC_GETEXPORT` function code, which returns the security export data structure.
2. When the installable security subsystem device driver is ready to begin monitoring security events, it calls `DevHlp_Security()` with the `DHSEC_SETIMPORT` function code, which passes in security import data structure that contains the installable security subsystem security event service routine entry points.

11.4.2 File System Open Callout

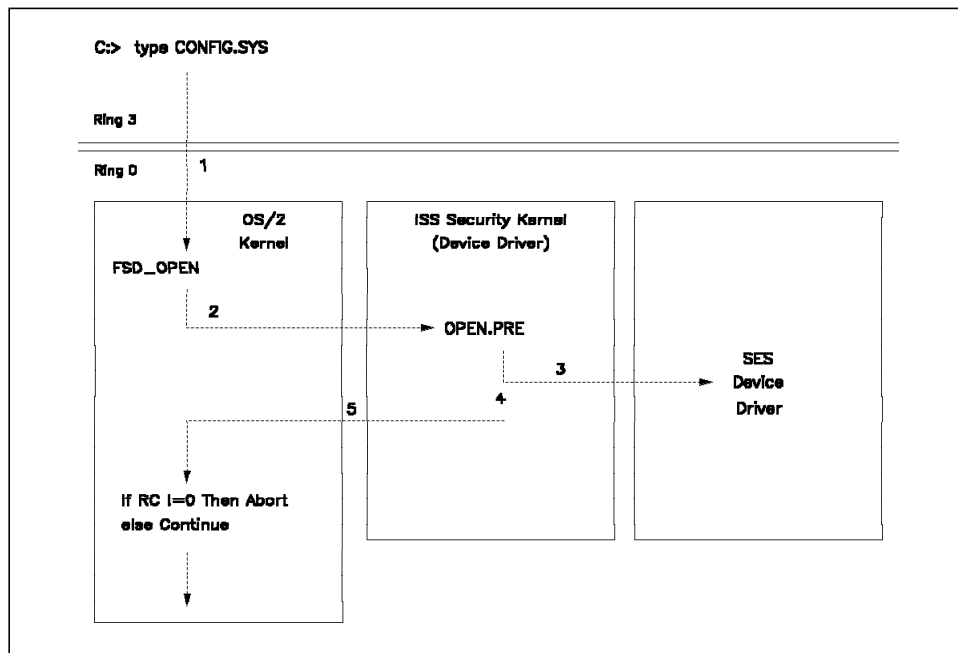
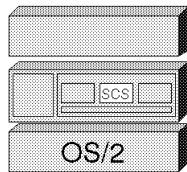


Figure 28. SKS System Design - File Open Callout Scenario

1. At an OS/2 command prompt, a user attempts to type a file. The TYPE command generates a DosOpen which is routed to the file system routine FSD_Open.
2. If security is enabled, security kernel services makes a callout to the address of the installable security subsystem device driver routine supplied for the Open_Pre event in the security import data structure.
3. The installable security subsystem device driver issues the security enabling services security helper SecHlpQuerySubjectInfo() to get the subject handle and user information for the current process/thread.
4. The installable security subsystem device driver looks up the user credentials associated with the handle to examine the user's access rights to this file.
5. If the user has access to the file, then Open_Pre returns to security kernel services with RC=0 and the FSD_Open call continues. If the user does not have access, then Open_Pre returns with RC=ERROR_ACCESS_DENIED and the FSD_Open call aborts.

Chapter 12. Security Context Services (SCS)



This section describes how key security control services components work together to provide the security context services. The security enabling services APIs for security control services are defined and scenarios are provided to show the flow of control/data for several key functions.

12.1 Security Context Authority Roles

To enable multiple security applications to work together, several security application roles have been defined. Each security role has a specific responsibility (for example, authenticate a user, establish a security context, protect a resource, etc.) and a specific set of privileges to invoke the security enabling services functions that are necessary for it to perform its responsibilities.

A process can have one or more of security enabling services privileges associated with it as indicated by the appropriate authority flags in the security context of the process. A process that has one or more of the security enabling services authority flags set in its security context. For example, a process that is performing the responsibilities of one of the defined security component roles is referred to as a security context authority.

The following sections describe the security control authority roles/authorities defined by security enabling services.

12.1.1 Access Control Authority (ACA)

An access control authority controls access to typically local resources/services based on the security context established by an SLA/RLA. When a subject handle is created or deleted by an SLA/RLA, the access control authority is notified so that it can establish its own set of credentials that it associates with the user/group/process handle.

For an access control request, an access control authority should query the subject handles associated with the requesting process/thread and perform the access control check based on the credentials associated with these handles such as the ACA should provide process/thread level access control instead of just assuming that all processes on the system are executing on behalf of the local system user. This is necessary to support trusted programs and multi-user applications (for example, dial-in facilities that support multiple concurrent users).

12.1.2 Agent Process Authority (APA)

One of the key Agent Process Authority (APA) requirements is to enable existing applications to run as agent programs. For example programs must be able to execute with agent user/group/process handles that are different than the client user/group/process handles of the user who invoked the program without requiring the process to call an security enabling services function to set its agent process authority flag.

SCS provides the following mechanisms that enable an installable security subsystem to establish the security context for an agent process authority during creation of the process:

1. A program can be specified as an agent process authority in SECURE.SYS (just like any other security context authority).

When any security control authority (including agent process authority) specified in SECURE.SYS is invoked, the appropriate security context is established and a process creation event is initiated when the system logon authority receives this event, it will be passed the default security context of the new process (inherited from parent or specified in SECURE.SYS) and the name of the program that is being executed.

When the system logon authority returns the event status, it can modify the client/agent user/group/process handles and the APA/EGF/EPF/EUF flags for the process that is being created.

2. If an installable security subsystem provides its own mechanisms for managing agent programs (for example POSIX SETUID function), the installable security subsystem can hook process creation (Pre-DosExecPgm) to modify the security context for the process that is being created.

Prior to the Pre-DosExecPgm callout, security enabling services temporarily resets the security context pointer in the trusted computing base of the calling thread (of the parent process) to point to the new security context being established for the child process. When the

installable security subsystem security kernel receives the Pre-DosExecPgm callout, the security context for the child process has already been established in accordance with the following:

- The inheritance policy specified by the parent process
- The authority/state flags specified in SECURE.SYS
- The handle/flag modifications specified by the system logon authority (if appropriate)

When the installable security subsystem security kernel is invoked for the Pre-DosExecPgm callout, it can modify the client/agent user/group/process handles and the APA/EGF/EPF/EUF flags for the child process that is being created by calling the SecHlpSetChildSecurityContext() KPI.

Note: Since subject handles can't be created from a Ring-0 KPI, the installable security subsystem security kernel must work with the installable security subsystem security daemon (SLA) to create whatever client/agent user/group/process handles are appropriate for the agent process authority program.

One alternative would be to have the installable security subsystem security daemon make a call into the installable security subsystem security kernel and block the thread of execution in the installable security subsystem security kernel until it needs the installable security subsystem security daemon to create the subject handles. When this captive thread of the installable security subsystem security daemon is unblocked, it can create the appropriate handles and return the values to the installable security subsystem security kernel.

When the child process is started, it will inherit the security context as modified by the installable security subsystem security kernel during the Pre-DosExecPgm callout. Prior to the Post-DosExecPgm callout to the installable security subsystem security kernel, security enabling services resets the security context pointer in the trusted computing base of the calling thread to point back to its own (for example, the parent process/thread) security context.

12.1.3 Client Logon Authority (CLA)

A client logon authority (CLA) acts as an agent for typically remote resources/services where the remote server performs its own authentication of a user. To enable the perception of single signon for the local system logon, a client logon authority is allowed to access the user name and token

used for local system logon by issuing the `SESQuerySubjectInfo()` or the `SESQuerySubjectHandleInfo()` call.

Some client logon authority services require the client logon authority to associate multiple sets of user credentials with the same subject handle and this must be accomplished without affecting any other SCA's views of its user credentials that are associated with the same handle.

Security enabling services provides a mechanism to enable a client logon authority to associate a client logon authority instance handle with a process (thread) that only has meaning to the client logon authority (all other SCAs will treat an instance handle exactly the same as the corresponding subject handle). Client logon authority instance handles can only be created when implicitly authorized to do so by another process and even then the client logon authority can only associate these instance handles with the requesting process (thread). However, please note that these instance handles will be inherited on process/thread creation just like any other subject handles and also when a server process authority assumes the client user/group/process handle of a client process/thread.

In all of the APIs, instance handles have the same functionality as other subject handles and will come from the same handle space as all subject handles. Consequently, the client logon authority can treat all handles exactly the same by managing a subject handle for which it has no instance handles as the first instance of the handle.

Note: CLAs must not unnecessarily create instance handles, for example, the only time instance handles should be created is when the client logon authority security context for the regular handle changes after the client logon authority security context is established for the regular handle. This is important because instance handles impact storage and performance.

CLAs do not delete instance handles. An instance handle is automatically deleted after it is set into the security context of the CLA's client process. For example, when an instance handle is no longer referenced by the CLA's client process (or any other process that inherited an instance handle from the CLA's client process), the delete handle notification for the instance handle will proceed.

A client logon authority may associate an instance of a subject handle (for example, instance handle) with the security context of a client process. It issues the `SESCreateInstanceHandle()` call to create an instance handle and

the `SESReturnEventStatus()` call to associate the instance handle with the client process. The flow is something like the following:

1. The client logon authority provides a thread for the security enabling services device driver to communicate with it by the `SESWaitEvent()` API this thread is blocked in the security enabling services device driver.
2. When another process requests some service that causes the client logon authority `SESWaitEvent()` thread to become unblocked, the client logon authority can create instance handles for the effective user/group/process handles of the client process/thread.
3. The client logon authority then signals that it is done with this request for service via the `SESReturnEventStatus()` API and that it is ready for another request for service via the `SESWaitEvent()` API not necessarily in that order (for example a multi-threaded process could have multiple `SESWaitEvent()` threads). One of the options of `SESReturnEventStatus()` that is only available to CLAs is to set the instance handles for the client process/thread.

Note that, although many instance handles can be created for a subject handle or for other instance handles, each client logon authority can maintain the heritage of its own instance handles if required.

12.1.4 Remote Logon Authority (RLA)

A remote logon authority (RLA) can establish the security context for typically remote system users by calling the `SESCreateSubjectHandle()` API to create user/group/process handles and by calling either the `SESSetSubjectHandle()` or `SESSetSecurityContext()` API to set the handles in its security context (which will be inherited by any child processes it creates for the remote user). A remote system user is defined as a user who is not associated with the OS/2 PM/WPS user interface services.

An remote logon authority can only set handles that it creates and handle -1 (the handle reserved for unauthenticated users). This policy is important for the following reasons:

- Each set handle operation (by a remote logon authority) should be regarded as a separate instance of the security context established at the root of a process tree. Set handle operations should not be used by a remote logon authority to impersonate a user (that's what server process authority is supposed to be used for).
- We don't want applications depending on the superuser remote logon authority powers to accomplish what could be done with server process authority or agent process authority. If we don't try to limit this

somehow, we'll end up facing all of the same problems UNIX had for so long with proliferation of superuser powers.

12.1.5 System Logon Authority (SLA)

A system logon authority (SLA) establishes the security context for local system logon and all processes created through the PM/WPS interfaces by the local system user. This is accomplished via the logon shell services event mechanism (see logon shell services system design for more details on local system logon).

The system logon authority can also establish the security context for other processes via the `SESCreateSubjectHandle()` call and `SESSetSubjectHandle()` or `SESSetSecurityContext()` calls.

Security enabling services also enables the system logon authority to provide trusted program support (for example, POSIX SETUID function) by allowing it to establish the agent security context of a process as it is started. The system logon authority is notified once (at process execution) via the process creation event if the process has any of the authority flags set in its security context (as inherited from its parent process or as specified in `SECURE.SYS`). When the system logon authority receives the process creation event, the default security context for the process is set in the event structure.

Programs starting before the system logon authority is active and has registered for the process creation event (for example, `SES`, `PSS`, `OS/2` system processes, `RUN=processes`) will be flagged so that this event will be instigated the first time the process enters the security enabling services device driver after the system logon authority has registered.

Note: The system logon authority must provide a security context for processes started before the system logon authority but the security context can include all of the handles being set to 0. Note that the system logon authority is the only security component that can set handle 0 because the system logon authority is responsible for enforcing the associated superuser policy if it chooses to do so. We're not recommending this policy.

12.1.6 Server Process Authority (SPA)

A server process authority (SPA) executes on behalf of multiple clients by assuming the effective user/group/process handles of the requesting client process as the client user/group/process handles of one of its threads via the `SESWaitEvent()` call. After assuming the handles, the SPA may reserve these handles for later use by issuing the `SESReserveSubjectHandle()` call. This call places the SPA's client user/group/process handles in a list of handles that any process/thread of the server process authority (for example, any process/thread with the same authority ID) is allowed to set in its client user/group/process handles at some later time.

Note: A thread of a server process authority process is allowed to set the client user/group/process handles in its effective security context equal to the handles in its allowed list (created via the `SESReserveSubjectHandle()` call) or the handles in the maximum security context of the server process authority process.

When the handles in the SPA's list of allowed handles are no longer required, it can issue the `SESReleaseSubjectHandle()` call to remove them from the list.

The process requesting service from the server process authority is not required to take any explicit action to authorize the server process authority to assume its effective user/group/process handles; this authorization is implied by the use of the server process authority services. For example:

- An access control authority such as a database manager might need to impersonate the credentials of its client processes. It would be allowed to do this (if it's also registered as an SPA in `SECURE.SYS`) when it receives the security context of a client process via `SESSendSecurityContext()`.
- A client logon authority such as a distributed computing environment client program might need to impersonate the credentials of its client processes. It would be allowed to do this (if it's also registered as an server process authority in `SECURE.SYS`) when it receives notification of client logon via the logon shell services single signon services.

The flow is something like the following:

1. The server process authority provides a thread for the security enabling services device driver to communicate with it by the `SESWaitEvent()` API. This thread is blocked in the security enabling services device driver.
2. When another process requests some service that causes the server process authority `SESWaitEvent()` thread to become unblocked, the

effective user/group/process handles of the requesting process will be copied into the client user/group/process handles of the server process authority.

Please note that this is not a security exposure (at least not for a discretionary access control policy) because the server process authority can choose to be executing on behalf of its agent user/group/process handles when it calls and returns from `SESWaitEvent()`. But, after doing whatever checking it deems appropriate, the server process authority can choose to execute on behalf of its client user/group/process handles to impersonate the credentials of the client process/thread.

Also please note that there is no control over what the server process authority can do with the effective user/group/process handles of the requesting process/thread for example create other processes/threads that will inherit these handles. But, again, this is not a security exposure (at least not for a discretionary access control policy).

3. The server process authority then signals that it is done with this request for service via the `SESReturnEventStatus()` API and that it is ready for another request for service via the `SESWaitEvent()` API not necessarily in that order (for example, a multi-threaded process could have multiple `SESWaitEvent()` threads).

The motivation for this policy is to support the principle of least privilege operation for example give the server process authority enough privilege to do what it needs to do, but without giving it the essentially unlimited privilege to execute on behalf of any user (for example, a very powerful user) without at least the implicit permission of the process requesting service from the server process authority.

Note: The assumption of the client's security context by a server process authority is not allowed on all `SESWaitEvent()` events. For example, when a user identification authority is invoked during system logon, the user has not yet been fully authenticated so, even if the user identification authority is also an server process authority, it will not be able to assume the client security context during this operation.

The effect of this policy is to enforce the same privilege model on APAs and SPAs. For example, an agent or server process can only inherit the security context a client process when invoked to perform some service for (and on behalf of) the client process. The key differences between an agent process authority and server process authority are described in Figure 29 on page 169.

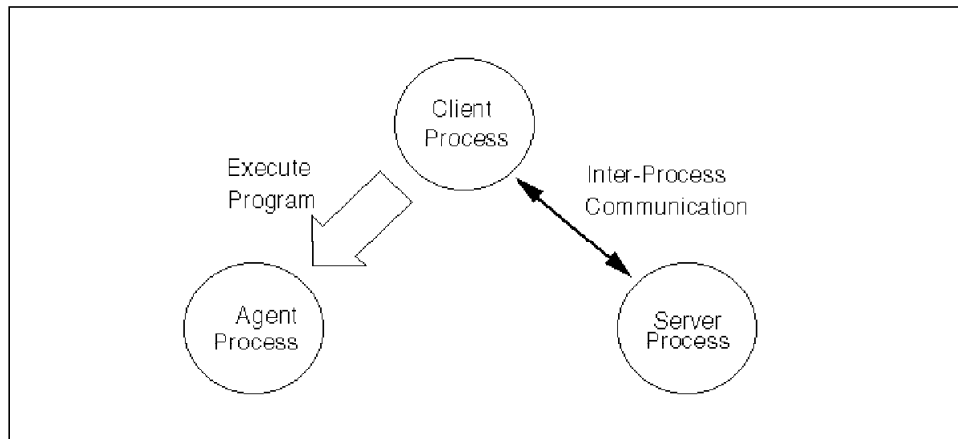


Figure 29. SCS System Design - Trusted Agent/Server Programs

An agent process authority can only execute on behalf of one set of client user/group/process handles at time.

However, some threads of an agent process authority can be acting on behalf of its own agent user/group/process handles at the same time that other threads are acting on behalf of the client.

Consequently, an agent process authority must enforce access control policy between the threads acting on behalf of the client and the threads acting on behalf of the agent (since all threads of a process can access objects opened by any thread of the process).

A server process authority can have threads acting on behalf of multiple sets of client user/group/process handles at the same time.

In addition, some threads of a server process authority can be acting on behalf of its own agent user/group/process handles at the same time that other threads are acting on behalf of clients.

Consequently, a server process authority must enforce access control policy between the threads acting on behalf of the different clients, and between the threads acting on behalf of the clients and threads acting on behalf of the agent (since all threads of a process can access objects of a process can access objects opened by any thread of the process).

From the viewpoint of the client, the effect is the same and the privilege model is the same.

12.1.7 User Identification Authority (UIA)

A user identification authority (UIA) performs identification and authentication of users for local system logon.

To enable the perception of single signon in environments where CLAs perform their own authentication, the system administrator needs to ensure that all of the UIAs and CLAs recognize a user by the same security enabling services subject name (the string name associated with a subject handle). In addition, the system administrator needs to ensure that each user identification authority and client logon authority authenticates the user with the same security enabling services subject token (the string token (password) associated with a subject handle) and keep these authentication tokens (passwords) in sync when a password is changed for one of the UIAs or CLAs.

Regardless of the method used by the user identification authority to identify and authenticate the user (for example, fingerprint, signature), when a user identification authority is invoked by security enabling services for identification and authentication of a local system user and the user is identified and authenticated, the user identification authority needs to return the subject name/token to security enabling services that is recognized by other UIAs, CLAs, ACAs, etc. (even though that may not be the same information used by the user identification authority to identify and authenticate the user).

For example, a fingerprint can be used to identify and authenticate the user in one operation that provides neither the subject name nor subject token, so, to enable the perception of single signon, the fingerprint user identification authority would need to be able to return the subject name/token to security enabling services that is recognized by other UIAs, CLAs, ACAs, etc. If the user identification authority cannot provide the common/shared subject name/token, then the perception of single signon cannot be supported. Note also that, to enable the perception of single signon, the system administrator needs to establish (and keep in sync) a common/shared subject name/token for each user for each UIA/CLA/ACA/etc. on a workstation.

12.2 Initialization of Security Context Services

Security control services is initialized, but disabled, when the security enabling services device driver is loaded during system initialization. The point in time where the security enabling services device driver is considered enabled is set when the SLA calls `SESRegisterDaemon()` for the Process Creation event. Calls to security enabling services APIs before this point will return `ERROR_SES_DISABLED`.

All processes starting before the system logon authority has registered (when security enabling services is considered disabled) will be associated with a default security context. They will be assigned handles equal to zero, and the authority flags will be set as specified in `SECURE.SYS` or inherited from the parent process. An unpublished flag will be set in the security context of all processes started before security enabling services is enabled to indicate that the system logon authority hasn't had the opportunity to modify the security context yet.

When these processes make their way back through security enabling services (after the system logon authority has registered for the process creation event), either by calling a security enabling services API or on behalf of a request coming through a security enabling services KPI, this flag will kick off a process creation event to give the system logon authority the opportunity to modify the security context if appropriate (for example, the process is not associated with the local system logon session or has at least one SES authority flag set).

The `SECURE.SYS` file contains all security context authority programs, for example, programs with any of the security enabling services authority flags specified (`/SLA`, `/UIA`). Each security control authority program (or set of programs) specified in `SECURE.SYS` (except agent process authority) must have a unique authority tag (`/SCA='xxxx'`, see the installation, configuration, initialization support system design section for details on `SECURE.SYS`).

During initialization, each security control authority (except agent process authority) will be assigned a unique authority ID as specified by the authority tag in `SECURE.SYS`. Authority IDs are assigned so that security enabling services can uniquely identify each security control authority that interacts with security enabling services through a captive thread mechanism.

Note: One of the requirements of the agent process authority model is to enable an installable security subsystem to provide POSIX `SETUID` function, which implies that the owner (not system administrator) of a program file can specify that the program executes as a process with

the SETUID (APA) privilege. Consequently, an installable security subsystem must be able to designate programs as APAs without necessarily listing the program file in SECURE.SYS (since only a system administrator should have access to SECURE.SYS, not the owner of the SETUID (APA) program file).

Consequently, by design, APAs do not need to have an authority ID assigned because security enabling services does not interact with an agent process authority (with no other security enabling services authority) through a captive thread mechanism, and, consequently, APAs (with no other security enabling services authority) do not need to be defined in SECURE.SYS.

All authority IDs are assigned from the range of 1 to 255. The following authority IDs are defined:

Authority ID: Security Context Authority	

0	: none
1	: SES
2	: PSS
3	: reserved
4	: SLA
5-15	: reserved
16-255	: all other SCAs

Multiple cooperating security control authority programs (processes) can have the same authority ID so that they can function as a single security control authority. This is accomplished by associating the cooperating programs with the same authority tag in SECURE.SYS (/SCA='xxxx', see the installation, configuration, initialization support system design section for details on SECURE.SYS).

The authority ID assigned to the security control authority is the lowest authority ID that can be assigned to any of the programs that function as a single security control authority. For example, if a security control authority includes programs that need system logon authority, access control authority, and server process authority privileges, then each program/process of the security control authority will be assigned authority ID=4 (the SLA's authority ID).

12.3 Establishment of Security Context at Process Creation

At process creation, the process security context is established as specified in SECURE.SYS and/or as inherited from the parent process.

The following are key policies that will be enforced at process creation:

- Client User Handle (CUH):

The child's maximum (and effective) client user handle is inherited from the parent's effective client user handle.

- Agent User Handle (AUH):

If the child program has any authority specified in SECURE.SYS or if the child process inherits any authority from the parent process, then the child's maximum (and effective) agent user handle can be provided by the system logon authority via the process creation event.

If the system logon authority doesn't provide an agent user handle and if the parent's propagate authority flag is set and the child process inherits any authority from the parent process, then the child's maximum (and effective) agent user handle is inherited from the parent's effective agent user handle.

If the system logon authority doesn't provide an agent user handle and if the child's maximum (and effective) agent user handle isn't inherited from the parent's effective agent user handle, then the child's maximum (and effective) agent user handle is inherited from the parent's effective client user handle.

- Client Group Handle (CGH):

The child's maximum (and effective) client group handle is inherited from the parent's effective client group handle.

- Agent Group Handle (AGH):

If the child program has any authority specified in SECURE.SYS or if the child process inherits any authority from the parent process, then the child's maximum (and effective) AGH can be provided by the system logon authority via the process creation event.

If the system logon authority doesn't provide an agent group handle and if the parent's propagate authority flag is set and the child process inherits any authority from the parent process, then the child's maximum (and effective) agent group handle is inherited from the parent's effective agent group handle.

If the system logon authority doesn't provide an agent group handle and if the child's maximum (and effective) agent group handle isn't inherited from the parent's effective agent group handle, then the child's maximum (and effective) agent group handle is inherited from the parent's effective client group handle.

- Client Process Handle (CPH):

The child's maximum (and effective) client process handle is inherited from the parent's effective client process handle.

- Agent Process Handle (APH):

If the child program has any authority specified in SECURE.SYS or if the child process inherits any authority from the parent process, then the child's maximum (and effective) agent process handle can be provided by the system logon authority via the process creation event.

If the system logon authority doesn't provide an agent process handle and if the parent's propagate authority flag is set and the child process inherits any authority from the parent process, then the child's maximum (and effective) agent process handle is inherited from the parent's effective agent process handle.

If the system logon authority doesn't provide an agent process handle and if the child's maximum (and effective) agent process handle isn't inherited from the parent's effective agent process handle, then the child's maximum (and effective) agent process handle is inherited from the parent's effective client process handle.

- Security Context Status:

- Authority ID:

If the child program has any authority specified in SECURE.SYS except agent process authority, then the child's authority ID is determined by the /SCA='xxxxxxx' option in SECURE.SYS.

If the child program has no authority or only agent process authority specified in SECURE.SYS but inherits authority other than agent process authority from the parent process, then the child's authority ID is inherited from the parent's authority ID.

If the child program has no authority or only agent process authority specified in SECURE.SYS and inherits no authority other than agent process authority from the parent process, then the child's authority ID is set to zero.

- Status Flags:

- Effective Group Flag:

The default setting of the flag is based on child's authority.

If the child process has any authority (if any of the child's maximum authority flags are set), then the child's maximum (and effective) effective group flag is set.

If the child process has no authority (if none of the child's maximum authority flags are set), then the child's maximum (and effective) effective group flag is not set.

The effective effective group flag can be reset (EGF=0) by the system logon authority via the process creation event or by the installable security subsystem security kernel via the ExecPgm callout.

- Effective Process Flag:

The default setting of the flag is based on child's authority.

If the child process has any authority (if any of the child's maximum authority flags are set), then the child's maximum (and effective) effective process flag is set.

If the child process has no authority (if none of the child's maximum authority flags are set), then the child's maximum (and effective) effective process flag is not set.

The effective effective process flag be reset (EPF=0) by the system logon authority via the process creation event or by the installable security subsystem security kernel via the ExecPgm callout.

- Effective User Flag:

The default setting of the flag is based on child's authority.

If the child process has any authority (if any of the child's maximum authority flags are set), then the child's maximum (and effective) effective user flag is set.

If the child process has no authority (if none of the child's maximum authority flags are set), then the child's maximum (and effective) effective user flag is not set.

The effective effective user flag can be reset (EUF=0) by the system logon authority via the process creation event or by the installable security subsystem security kernel via the ExecPgm callout.

- Local User Flag:

If the child program has `/LOCALUSER=YES` specified in `SECURE.SYS`, then the child's maximum (and effective) local user flag is set.

If the child program has `/LOCALUSER=NO` specified in `SECURE.SYS`, then the child's maximum (and effective) local user flag is not set.

If the `/LOCALUSER` option is not specified in `SECURE.SYS`, then the child's maximum (and effective) local user flag is inherited from the parent's effective local user flag.

- Propagate Authority Flag:

If the child program has `/PROPAGATE=YES` specified in `SECURE.SYS`, then the child's maximum (and effective) propagate authority flag is set.

If the child program has `/PROPAGATE=NO` specified in `SECURE.SYS` or if the `/PROPAGATE` option is not specified in `SECURE.SYS`, then the child's maximum (and effective) propagate authority flag is not set.

- Authority Flags:

If the child program has any authority specified in `SECURE.SYS`, then the corresponding authority flags are set in the child's maximum (and effective) security context status.

If the parent's propagate authority flag is set, then the child's maximum (and effective) authority flags will be inherited from the parent's effective authority flags (in addition to whatever authority flags are set as specified in `SECURE.SYS`). The maximum (and effective) agent process authority flag can also be set by the system logon authority via the process creation event or by the installable security subsystem security kernel via the `Pre-ExecPgm` callout.

Otherwise, none of the child's maximum (or effective) authority flags will be set (indicating that it has no authority).

12.4 Association of Security Context with OS/2 IPC

The standard OS/2 IPC services don't provide any way for a server process to obtain the security context of a client process. Consequently, a server process can exploit the `SESSendSecurityContext()` call to determine the security context of a client process that it is communicating with via one of the standard OS/2 IPC services. This can be accomplished by the server process associating some unique information with each instance of an OS/2 IPC service (for example a unique session identifier), and then exchanging this information with the client via a `SESSendSecurityContext()` call from the client (so that the client's security context can be associated with the unique instance of the OS/2 IPC service).

To use the `SESSendSecurityContext()` call, the sending process (client) must know the authority ID of the target security control authority process (server). The authority ID for each security control authority is defined when security enabling services is initialized as specified by the authority tag (`/SCA='xxx'`) in `SECURE.SYS`. The `SESQueryAuthorityID()` call can be used to obtain the authority ID for a security control authority given its `AuthorityTag` (as defined in `SECURE.SYS`). Consequently, each security control authority must provide some way to make either its authority tag or its authority ID known to client processes that need to communicate with it.

Note: Establishing IPC between clients and servers typically requires publishing some well known information about the server that clients can use to establish communication with the server, for example a well known named pipe.

The same types of well known solutions can be used by an SCA to make its `AuthorityTag` or authority ID known to a client process, for example publishing a well known `AuthorityTag` or publishing the name of a well known file that contains the authority ID (which the security control authority updates during initialization).

Please note that mechanisms used to establish IPC between clients and servers aren't usually exposed to customers, client/server products typically include client library APIs that hide these mechanisms. For example, the well known mechanism used to establish IPC between a client process and a server process could be hidden in a `ConnectToServer()` API.

A process that is not an security control authority (for example, it doesn't have any security enabling services authority flags set in its security context) or a process that is only an agent process authority (for example, it has only

the agent process authority flag set in its security context) cannot be the target for `SESSendSecurityContext()` because the authority ID will be set to zero (`AuthorityID=0`) in its security context.

Please note that, although `SESSendSecurityContext()` can be used to exchange short messages between client/server processes, the primary intent is to augment (not replace) the standard OS/2 IPC services (for example, pipes, queues, shared memory, semaphores); and, consequently, provides very limited IPC function:

- The security context (and short message) can only be sent to a security control authority.
- The exchanged messages can be at most 512 bytes long.
- No prioritized messages, no typed messages, no peeking.
- The send/receive is always synchronous, for example the target security control authority must respond before the sender can proceed.

12.5 Security Context Services API

The security context services (SCS) APIs are implemented as part of the security enabling services DLL, which can be invoked by any process. Use of the security control services APIs is restricted based upon the effective authority of the calling thread. The table below indicates the functions available to each type of authority. If there is insufficient authority for the call, the function returns with invalid authority. Any returned data is invalid.

Note: Only the security enabling services APIs for security control services are listed in the following table. Additional security enabling services APIs for logon shell services are listed in the logon shell services system design section.

SCS Functions	SES Authority							
	UPA	APA	SPA	RLA	SLA	ACA	CLA	UIA
SESControlProcessCreation()	no	no	no	yes	yes	no	no	no
SESCreateHandleNotify()	no	no	no	no	no	yes	no	no
SESCreateInstanceHandle()	no	no	no	no	no	no	yes	no
SESCreateSubjectHandle()	no	no	no	yes	yes	no	no	no
SESDeleteHandleNotify()	no	no	no	no	no	yes	no	no
SESDeleteSubjectHandle()	no	no	no	* 1	* 1	no	no	no
SESKillProcess()	no	no	no	yes	yes	no	no	no
SESlogIntegrityViol()	* 7	yes	yes	yes	yes	yes	yes	yes
SESQueryAuthorityID()	yes	yes	yes	yes	yes	yes	yes	yes
SESQueryContextStatus()	yes	yes	yes	yes	yes	yes	yes	yes
SESQueryProcessInfo()	yes	yes	yes	yes	yes	yes	yes	yes
SESQuerySecurityContext()	yes	yes	yes	yes	yes	yes	yes	yes
SESQuerySubjectHandle()	yes	yes	yes	yes	yes	yes	yes	yes
SESQuerySubjectHandleInfo()	* 5	* 5	* 5	* 5	* 5	* 5	yes	yes
SESQuerySubjectInfo()	* 5	* 5	* 5	* 5	* 5	* 5	yes	yes
SESReleaseSubjectHandle()	no	no	yes	no	no	no	no	no
SESReserveSubjectHandle()	no	no	yes	no	no	no	no	no
SESResetThreadContext()	no	yes	yes	yes	yes	yes	yes	yes
SESSendSecurityContext()	yes	yes	yes	yes	yes	yes	yes	yes
SESSetContextStatus()	no	yes	yes	yes	yes	yes	yes	yes
SESSetSecurityContext()	no	no	* 6	* 6	* 6	no	no	no
SESSetSubjectHandle()	no	no	* 2	* 3	* 4	no	no	no
SESSetSubjectInfo()	no	no	no	yes	yes	no	no	no

NOTES:

- (* 1) Can only delete the handles that it creates.
- (* 2) A server process authority can set the client handles in its effective security context to the handles that are in its list of allowed handles or that are in its maximum security context.
- (* 3) An remote logon authority can set the CLIENT/AGENT handles in its effective security context to the handles that it creates and handle -1.
- (* 4) A system logon authority can set the CLIENT/AGENT handles in its effective security context to any valid handle including 0 and -1.
- (* 5) A security control authority without CLA/UIA can only can only query the name, source, handle, and instance (only a CLA/UIA can query the token).

- (* 6) For setting the handles Note *2 applies to server process authority, Note *3 to remote logon authority, and Note *4 to SPA. Context status changes are as for `SESSetContextStatus`.
- (* 7) The installable security subsystem should enforce an access control policy on the `SES.LOG` file for example untrusted processes should not be allowed to access it.

Full details of the security control services API functions may be found in Appendix B, “Security Context Services (SCS) API Details” on page 241.

12.6 SCS Scenarios

The following scenarios are described in this section:

- A User Logs on Remotely to an Application Server
- An Untrusted Process Creates an Untrusted Child Process
- A Process Sends Its Security Context to an SCA Process
- An SPA Process Acts as Proxy for Its Client Processes

12.6.1 A User Logs on Remotely to an Application Server

The way how a user logs on remotely to an application server is described in Figure 30 on page 181

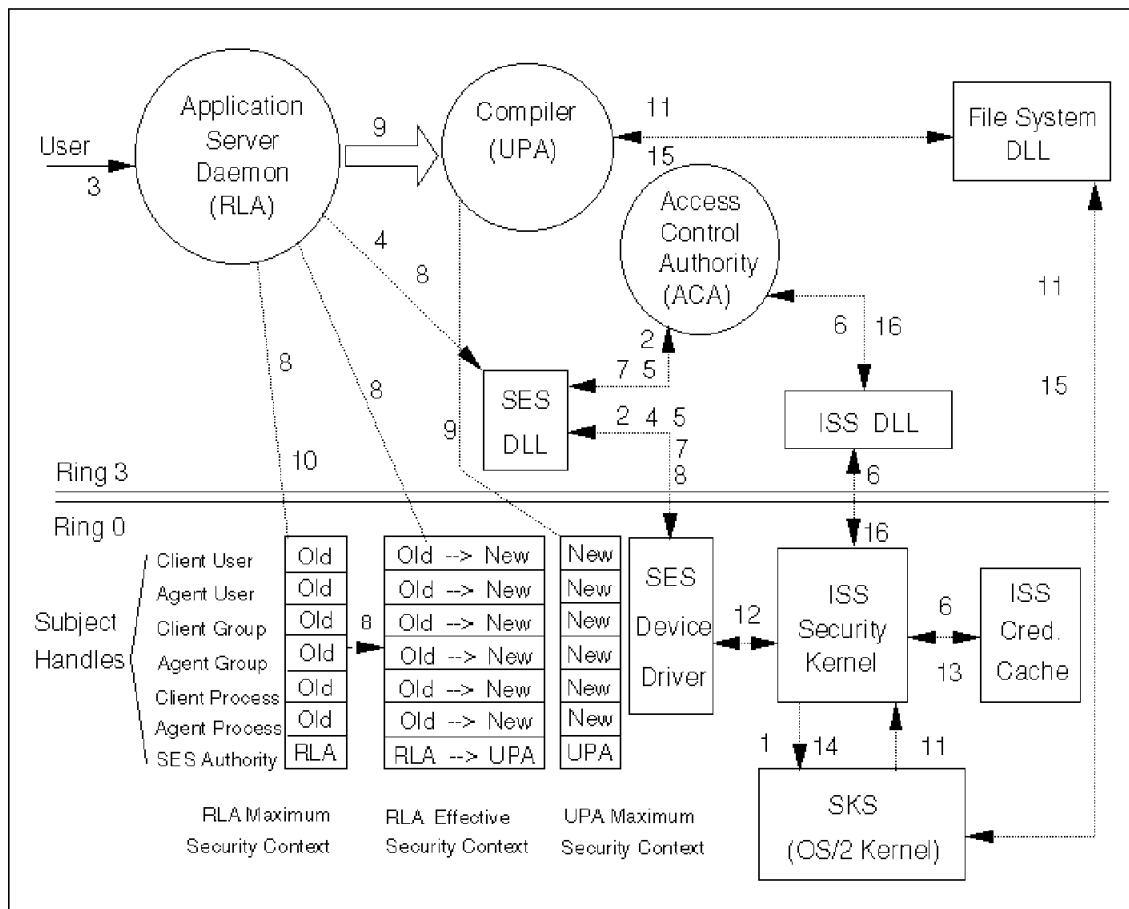


Figure 30. SCS System Design - Remote Logon Scenario

1. The installable security subsystem device driver issues the security device helper call (1) when it initializes in order to request notification from the security kernel services of security events that are of importance to it.
2. An access control authority issues the SCSCreateHandleNotify() call (2) so that it is notified of all new users entering the system.
3. The remote user initiates a remote logon request by entering a user name and password (3).
4. The application server daemon (an remote logon authority) receives the request and authenticates the user on the local system. If the user was authenticated, the daemon issues the SCSCreateSubjectHandle call using the user's name and password as the user information for the handle (4).

5. The `SESCreateHandleNotify` call() returns to the access control authority with the security control services handle and user information for the user (5).
6. The access control authority looks up the user's credentials, associates the security control services handle with it, and then sends the data down to the installable security subsystem device driver so that it can be cached in Ring 0 memory for fast access (6).
7. The access control authority then re-issues the `SESCreateHandleNotify`() so that it can receive notification of the next user that has entered or enters onto the system (7).
8. The daemon now executes the user's request to compile a file. It issues `SESResetThreadContext` call to reset its effective security context to be a private copy of security context (contents of the private copy of security context are copied from the daemon's maximum security context). The daemon sets the handles in the new effective security context to the handles it created for the user by issuing the `SESSetSubjectHandle` call (8) a multiple number of times. It sets the authority flags to zero in the new effective security context (to make it UPA) via `SESSetContextStatus` call.
9. The daemon then executes the compiler program by issuing a `DosExecPgm` call (9). The created compiler process inherits the daemon's effective security Context as its maximum security context.
10. The daemon issues `SESResetThreadContext` call to reset its effective security context to its maximum security context and therefore re-obtains its previous security context (for example, old user as subject handles, remote logon authority). The private copy of security context is deleted since there is no other reference to it (10).
11. When the compiler process attempts to read a file, the security kernel services notifies the installable security subsystem device driver via a PRE-READ callout (11).
12. The installable security subsystem device driver responds by issuing the `SecHlpQuerySubjectHandle` call (12).
13. The security enabling services device driver, upon receiving this call through its KPI, checks the security control services initialization state of the calling thread and finds that it is initialized. The security control services handle is returned from the new set of attributes and the installable security subsystem device driver uses it to locate the user's credentials. The user's access rights to the file are then examined (13).

14. If the user has read access to the file, the installable security subsystem device driver returns to the security kernel services with a return code indicating no error (14).
15. If the user does not have access, a return code of access denied will be indicated and the file access is terminated. The security kernel services passes the received return code back to the file system which in turn passes it back to the compiler process (15).

12.6.2 An Untrusted Process Creates an Untrusted Child Process

Figure 31 describes how the untrusted process creates an untrusted child.

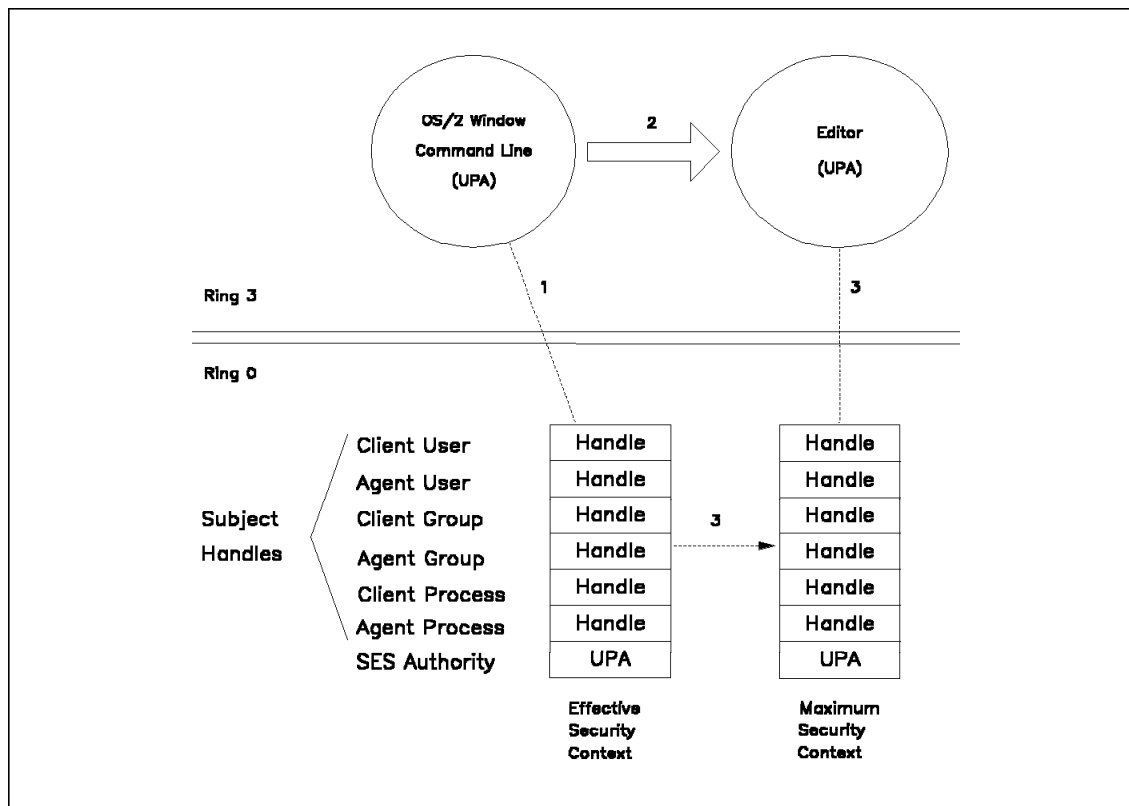


Figure 31. SCS System Design - Process Creation Scenario

1. A very common event in a system is the creation of one process, a child process, by another process, a parent process. The parent process, an OS/2 Window or command line, prior to creating the child process, an editor, has the effective security context (which is its maximum security context) that indicates the user has the authority of UPA (1).

2. The OS/2 command line, in order to create the editor process, issues the DosExecPgm call (2).
3. The child process inherits the parent thread's effective security context as its maximum security context. The majority of processes in the system are of this type and created in this manner (3).

12.6.3 A Process Sends Its Security Context to an SCA Process

Figure 32 shows how security context sends a process to an SCA process.

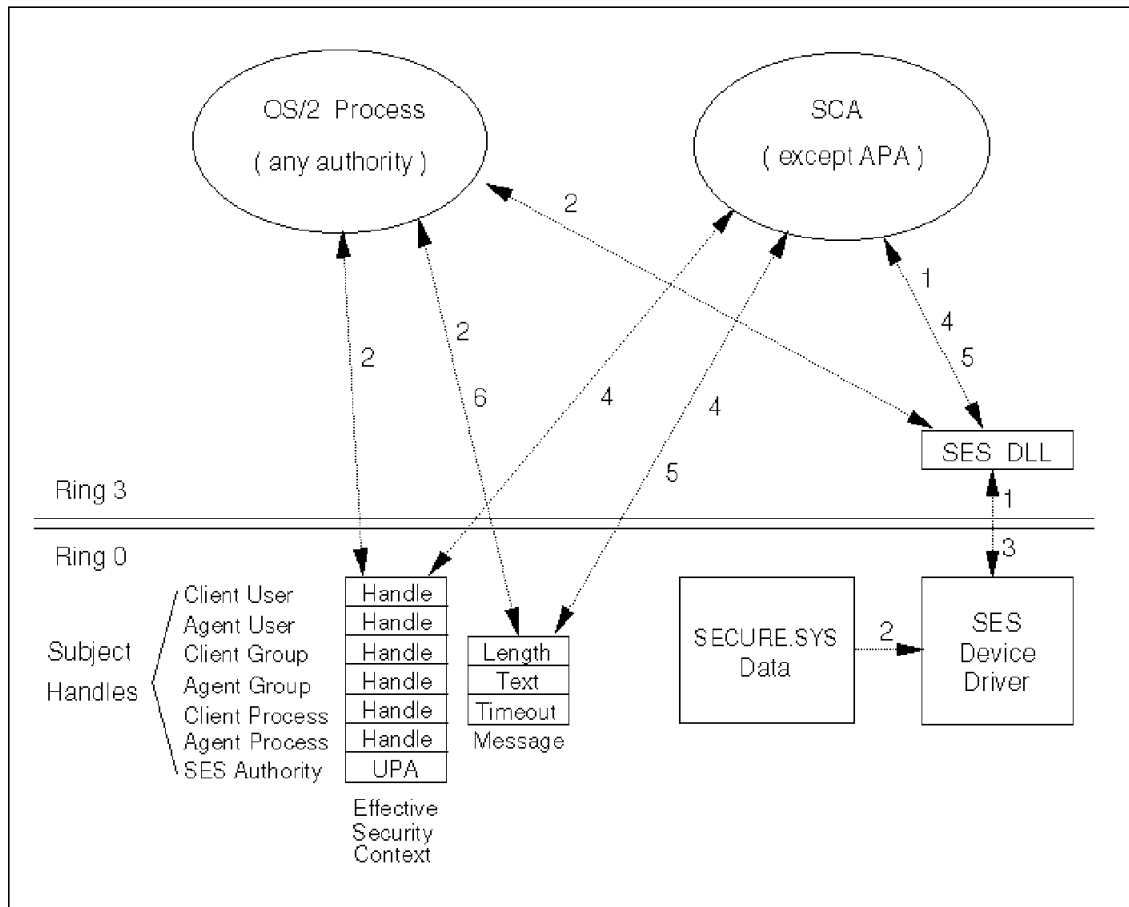


Figure 32. SCS System Design - Inter-Process Communication Scenario

1. A security control authority process (except an agent process authority) provides a thread for the security enabling services device driver to

communicate with it by the `SESWaitEvent` call send security context event. The thread is blocked in the device driver (1).

2. Another process needs to send a message with its security context to the security control authority. The process issues the `SESQueryAuthorityID` call to retrieve the SCA's `AuthorityID` by providing the SCA's `AuthorityTag` defined in the `SECURE.SYS`. The sending process then issues the `SESSendSecurityContext` call to send the message to the security control authority identified by its `AuthorityID` (2).
3. The sending process/thread is blocked until either the message and the sender's effective security context is delivered to the security control authority or the specified timeout value is exceeded (3).
4. The SCA's thread waiting on the `SESWaitEvent` call for the send security context event is unblocked and receives the message and the sender's security context (4).
5. The SCA's thread indicates that it has received (and optionally processed) the message by issuing the `SESReturnEventStatus` call with an optional return message (5).
6. The sending process/thread which is blocking on the `SESSendSecurityContext` call is unblocked by the security enabling services device driver and receives the return message (6).

12.6.4 An SPA Process Acts As Proxy for Its Client Processes

Figure 33 shows how the SPA process acts as proxy for a client process.

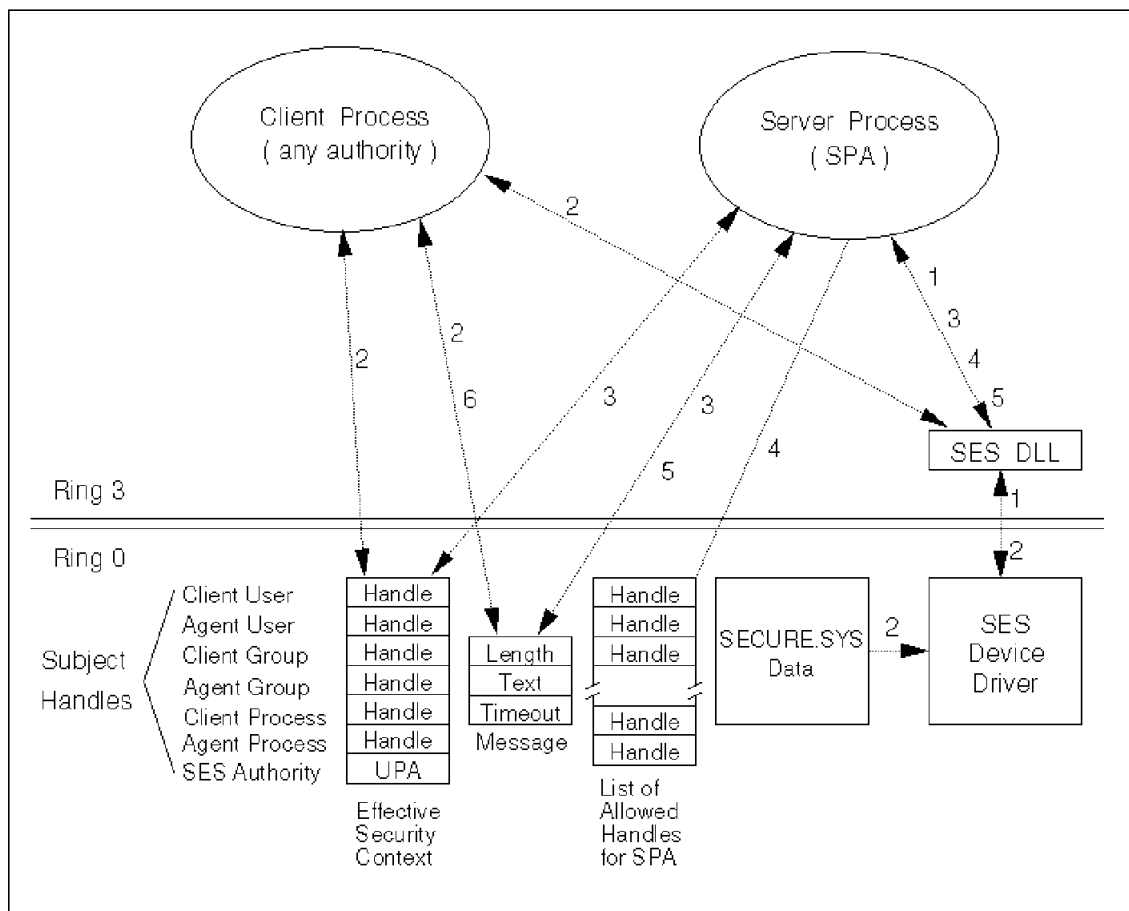


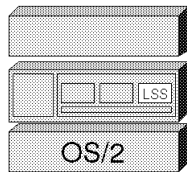
Figure 33. SCS System Design - Trusted Server Process Scenario

1. A server process authority process provides a thread for the security enabling services device driver to communicate with it by the `SESWaitEvent` for send security context call. The thread is blocked in the device driver (1).
2. A client process wants to send a request to server process authority. The process issues the `SESQueryAuthorityID` call to retrieve the SPA's authority ID by providing the SPA's `AuthorityTag` defined in the `SECURE.SYS`. The client process then issues the `SESSendSecurityContext` call to send the request and its effective

security context to the server process authority. The client process is blocked until the request is delivered to the server process authority (2).

3. The SPA's thread waiting on the `SESWaitEvent` call for the send security context event is unblocked and receives the request. The effective user and group handles of the client process is copied into the client user and group handles of the server process authority by the security enabling services device driver (3).
4. The server process authority may add its client user/group/process handles (for example, its client's effective handles) to a list of handles that any process/thread of the SPA's program group can set as its client user/group/process handles, by issuing the `SESReserveSubjectHandle` and `SESSetSubjectHandle` calls (4).
5. The server process authority indicates that it is done with this request for service by issuing the `SESReturnEventStatus` call and that it is ready for another request for service by issuing the `SESWaitEvent` call (5).
6. The client process which is blocking on the `SESStartEvent` call is unblocked by the security enabling services device driver and receives the reply (6).

Chapter 13. Logon Shell Services (LSS)



The logon shell services are implemented as a state machine triggered by events that require coordination among various cooperating security components.

For example, when a user is logging on to the local system, the following components may be involved:

- Logon shell services (or some other component) starts a logon event.
- The UIAs authenticate the user for local system logon.
- The system logon authority establishes the local security context for the user and associates the appropriate user/group/process installable security subsystem credentials (for example user ID, group ID(s), default access control policy for object creation) with the user's client user/group/process handles for the local system logon session.
- The CLAs authenticate the user for remote services that don't accept the local authentication, and associate the appropriate user/group/process client logon authority credentials with the user's client user/group/process handles.
- The ACAs associate the appropriate user/group/process access control authority credentials with the user's client user/group/process handles.
- Logon shell services starts the user shell (if `RESTARTUSERSHELL=YES`) or associates the user shell with the user's client user/group/process handles (if `RESTARTUSERSHELL=NO`).

13.1 Overview of LSS Event Flows

The following sections provide an overview of selected logon shell services event flows. Detailed scenarios are provided after the APIs are defined.

13.1.1 Logon/Unlock

Logon is initiated by the `SESStartEvent()` API. Security enabling services processes this event and calls the system logon driver to determine which UIA(s) will participate in identifying and authenticating the user.

Security enabling services reserves a client user handle to associate with the user's name for this event and invokes the user identification authority specified by the system logon driver. The first user identification authority typically obtains the user name and password (if available). The user name and password are propagated by security enabling services to each subsequent user identification authority. Each user identification authority will authenticate the user and return the result back to security enabling services. Security enabling services passes the result of the previous user identification authority back to the system logon driver, which then decides whether to invoke the services of another user identification authority or not.

When identification and authentication is complete, the system logon authority is notified of the final status determined by the system logon driver. The system logon authority applies its logon policy and returns status to security enabling services indicating if system logon is to be performed. If system logon is indicated, the client user handle reserved for local system logon is created by security enabling services (for example, registered ACAs will now be notified of the creation of the client user handle), and the client user/group/process handles specified by the system logon authority (the client group/process handles must be created by the system logon authority if they don't already exist) will be used by security enabling services to associate the user with the local system logon session (PM, user shell). This ensures that all processes executed on behalf of the local user will inherit the appropriate client user/group/process handles.

Note: At this point in the local system logon flow, security enabling services will delete the client user handle that it created. This ensures that, when the user logs off and the handle is no longer referenced by any process/thread (including an SPA's list of allowed handles), ACAs will be notified that the handle has been deleted.

Upon concluding its local logon tasks, security enabling services calls the client logon driver to determine which CLAs to notify. Security enabling

services then notifies each client logon authority of the local system logon, passing it the user's name and local password (again, as entered by the user during local system logon). The client logon authority can attempt to use this name (or associated CLA-specific information such as user ID, domain) and password (if the local system password and the CLA's password are kept in sync) to authenticate the user without further user intervention, or the client logon authority can prompt the user for the necessary information (user ID, domain, password) as necessary.

Next, logon shell services will either start the user shell (if `RESTARTUSERSHELL=YES`) or will associate the user shell with the user's security context (if `RESTARTUSERSHELL=NO`).

The final step in the logon process is to return the status to the process which started the logon operation.

The unlock flow is very similar to the logon flow.

13.1.2 Lock/Logoff/Shutdown

Logoff is controlled through security enabling services. If logoff is initiated, the first step is to query the system logon authority to determine if a logoff is appropriate (the user may have inadvertently initiated the logoff, or may have forgotten that important processes haven't completed yet).

Note: The `SESQueryProcessInfo()` API can be used to get information about the user's active processes, which can then be displayed to the user before prompting for confirmation of the logoff request (same for shutdown).

Security enabling services then calls the client logon driver to determine which CLAs to signal that the local system logoff event has occurred.

CLA A client logon authority can log the user off of its remote system and return with status.

Next, logon shell services will either terminate the user shell (if `RESTARTUSERSHELL=YES`) or will associate the user shell with the logoff state security context (if `RESTARTUSERSHELL=NO`).

Security enabling services also notifies the system logon authority that the local system logoff event is in progress.

SLA The system logon authority can then enforce its security policy, for example terminate all untrusted processes in the system that are associated with the current system user who is logging off.

Security enabling services then associates the PM process with the logoff state security context.

The lock and shutdown flows are very similar to the logoff flow.

13.1.3 Change Password

If a change password event is initiated, the following information must be supplied by the calling process:

- Name of user associated with the calling process
- Current password of user associated with the calling process
- Name of target user
- New password for target user

Note: For user's who are changing their own password, the name of the user associated with the calling process and the name of the target user will be the same and the current password will be this user's current password.

But, for a system administrator who is changing someone else's password, the system administrator's name will be the one associated with the calling process and the current password will be the system administrator's password.

The new password will be validated by the password validation driver and then passed to the UIAs (as determined by the system logon driver) and to the CLAs (as determined by the client logon driver).

13.2 Overview of Keyboard/Mouse Support

LSS works with the keyboard/mouse device drivers to provide the following:

- Trusted path support
- Keyboard/mouse activity detection
- Keyboard/mouse inactivity detection

13.2.1 Trusted Path Support

Logon shell services modifies the function of the Ctrl-Alt-Del and Ctrl-Alt-NumLock-NumLock key combinations to support installable security subsystem trusted path services by enabling the installable security subsystem security kernel to control the action the keyboard device driver will take when these trusted path key combinations are detected. In the absence of an installable security subsystem security kernel, the trusted path

key combinations invoke the standard function (for example, reboot for Ctrl-Alt-Del). The actions the installable security subsystem security kernel can cause the keyboard device driver to take are:

1. Perform normal function (for example, reboot for Ctrl-Alt-Del)
2. Do nothing (remove the keystroke from the input stream)

To enable the trusted path support, the TRUSTEDPATH=YES environment variable must be specified in CONFIG.SYS.

Note: This stops logon shell services from initiating a logon/unlock events if keyboard/mouse activity is detected while in the logoff/lock states. If TRUSTEDPATH=YES is specified, it is the responsibility of the installable security subsystem trusted path services to initiate the logon/unlock events.

The flow will be something like the following:

1. The keyboard device driver detects the trusted path key combination and notifies the installable security subsystem security kernel via the trusted path callout.
2. The installable security subsystem security kernel unblocks a captive thread of the installable security subsystem security daemon and directs the keyboard device driver to ignore the trusted path key combination.
3. The installable security subsystem security daemon takes control of the OS/2 user interface services (for example, disable keyboard monitors), presents a trusted path interface to the user (for example, a menu of options supported by the installable security subsystem), and determines the appropriate action to take (for example, do nothing, log on, unlock, shutdown).

13.2.2 Keyboard/Mouse Activity Detection

When the logon shell services state machine is in a logoff/lock state and the TRUSTEDPATH=YES environment variable is not specified, logon shell services will do the following:

- Monitor keyboard/mouse input for user activity
- Start a logon/unlock event when user activity is detected

13.2.3 Keyboard/Mouse Inactivity Detection

The security enabling services device driver communicates with the keyboard/mouse device drivers to specify a memory address, the input activity address, that is used for keyboard/mouse inactivity detection. The keyboard/mouse device drivers increment the IAA every time they sense activity (mouse button clicks or keystrokes). The security enabling services device driver inspects the IAA on a regular basis to detect user inactivity.

The `SESIActivityNotify()` API enables the installable security subsystem security daemon (SLA) to specify the inactivity time period and to be notified of user inactivity after the specified time period. The thread executing this API is blocked in the security enabling services device driver until user inactivity is detected. When no keyboard/mouse activity is detected for the specified time period, the thread will be unblocked and the installable security subsystem security daemon can take the appropriate action (usually to initiate a lock event, but perhaps a logoff or shutdown event).

13.3 System Logon Driver API

This section describes the APIs and the parameters that are passed to the system logon driver. When a logon shell services event occurs for the local system logon session (such as a local system logon event), the system logon driver is called to determine which UIAs participate in the event.

The first call to `SLDQueryUIA()` for an event returns the authority ID of the first user identification authority that should be invoked for the event. Subsequent calls to `SLDQueryUIA()` will pass in the status of the previous UIA's response, and will return the authority ID of the next user identification authority that should be invoked for the event. This process continues until the system logon driver determines that all target UIAs have been invoked.

The system logon driver indicates that it has determined the final status of the event by setting the appropriate return code and the final values for the following:

EventStatus: This field should now be set to the final status that the system logon driver wants to have passed to the system logon authority, for example authenticated or unauthenticated, regardless of the various intermediate results that each user identification authority returned to the system logon driver.

For example, a system logon driver could query all of the UIAs and determine the status based on the combined results of all of the UIAs, some might return AUTHENTICATED, UNAUTHENTICATED, NOT_AVAILABLE, etc, but the system logon driver must return the combined result (authenticated or unauthenticated) on the return for the final (last one for this event) SLDQueryUIA() call.

AuthoritySource: If the final EventStatus is authenticated, the AuthoritySource contains the source of authority for the authentication that the system logon driver wants to have associated with the client user handle for the user logging on.

In the simplest case, this could be the authority ID of the single user identification authority that returns an authenticated status (possible value range 4 to 255). For more complex policies, the AuthoritySource could represent an authentication rule number (possible value range 256 and above).

Note that the definition of an authentication rule for the source of authority associated with a CUH must be provided by the system logon driver and understood by any SCAs who want to use it in their policies. For example, an installable security subsystem that is providing a system logon driver could also provide a system logon authority and access control authority that associate, more or less privileges with a client user handle depending on the source of authority (authentication rule) associated with it.

The system logon driver DLL may be replaced, for example, as a component of an independent software vendor security product or customer application.

Note: The default policy for the system logon driver provided with security enabling services is to process the UIAs in the order they are listed in SECURE.SYS, and to stop when one user identification authority returns either authenticated or unauthenticated. If no user identification authority returns either authenticated or unauthenticated, then the final status is unauthenticated.

Full details of the system logon driver API may be found in Appendix C, “System Logon Driver API Details” on page 271.

13.4 Client Logon Driver API

This section describes the APIs and the parameters that are passed to the client logon driver. When a logon shell services event occurs for the local system logon session (such as a local system logon event), the client logon driver is called to determine which CLAs participate in the event.

The first call to `CLDQueryCLA()` for an event returns the authority ID of the first client logon authority that should be invoked for the event. Subsequent calls to `CLDQueryCLA()` will pass in the status of the previous CLA's response, and will return the authority ID of the next client logon authority that should be invoked for the event. This process continues until the client logon driver determines that all target CLAs have been invoked.

The client logon driver DLL may be replaced, for example as a component of an independent software vendor security product or customer application.

Note: The default policy for the client logon driver provided with security enabling services is to process all of the CLAs listed in `SECURE.SYS` in the order they are listed. The return codes from the CLAs are ignored.

Full details of the client logon driver API may be found in Appendix D, "Client Logon Driver API Details" on page 275.

13.5 Password Validation Driver API

This section describes the API and the parameters that are passed to the password validation driver. Whenever a password must be validated, (for example dictionary check, composition rules) the security enabling services daemon calls this API.

The password validation driver DLL may be replaced, for example as a component of an independent software vendor security product or customer application.

Note: The default policy for the password validation driver provided with security enabling services is to allow any password.

Full details of the password validation driver API may be found in Appendix E, "Password Validation Driver API Details" on page 279.

13.6 Logon Shell Services API

The logon shell services APIs are implemented as part of the security enabling services DLL, which can be invoked by any process. Use of the logon shell services APIs is restricted based upon the effective authority of the calling thread. The table below indicates the functions available to each type of authority. If there is insufficient authority for the call, the function returns with invalid authority. Any returned data is invalid.

Note: Only the security enabling services APIs for logon shell services are listed in the following table. Additional security enabling services APIs for security control services are listed in the security control services system design section.

Logon Shell Services (LSS) Functions	SES Authority							
	UPA	APA	SPA	RLA	SLA	ACA	CLA	UIA
SESControlKBDMonitors()	no	no	no	no	yes	no	no	no
SESIactivityNotify()	no	no	no	no	yes	no	no	no
SESRegisterDaemon()	no	no	yes	yes	yes	yes	yes	yes
SESReturnEventStatus()	no	no	yes	yes	yes	yes	yes	yes
SESReturnWaitEvent()	no	no	yes	yes	yes	yes	yes	yes
SESStartEvent()	* 1	* 1	* 1	* 1	* 1	* 1	* 1	* 1
SESWaitEvent()	no	no	yes	yes	yes	yes	yes	yes
Receiving SES events via SESWaitEvent() or SESReturnWaitEvent()								
Change Password	no	no	no	no	yes	no	yes	yes
Create Profile	no	no	no	no	yes	no	yes	no
Delete Profile	no	no	no	no	yes	no	yes	no
Identify and Authenticate	no	no	no	no	no	no	no	yes
Lock	no	no	no	no	yes	no	yes	yes
Logoff	no	no	no	no	yes	no	yes	yes
Logon	no	no	no	no	yes	no	yes	yes
Process Creation	no	no	no	no	yes	no	no	no
Send Security Context	no	no	yes	yes	yes	yes	yes	yes
Shutdown	no	no	no	no	yes	no	yes	yes
Unlock	no	no	no	no	yes	no	yes	yes

Table 7. Logon Shell Services Functions

NOTE:

(* 1) Calling process must have local user flag set to one (LUF=1) in its effective security context.

Full details of the logon shell services API may be found in Appendix F, "Logon Shell Services API Details" on page 281.

13.7 Logon Shell Services Kernel Programming Interface

The logon shell services kernel programming interfaces enable the installable security subsystem security kernel to do the following:

- Receive callouts for trusted path invocation (through the SKS security event router)
- Receive callouts when LSS APIs are invoked (through the SKS security event router)

13.7.1 Callouts to the ISS Security Kernel for LSS Functions

The callouts for logon shell services functions are described in Chapter 11, “Security Kernel Services (SKS)” on page 151 and Appendix A, “Security Kernel Services KPI Details” on page 239.

13.8 Logon Shell Services (LSS) Scenarios

The following scenarios are described in this section:

- Logon
- Unlock
- Logoff, Shutdown
- Lock
- Change User Password
- Create User Profile, Delete User Profile
- Identification and Authentication
- Send Security Context
- Process Creation
- Trusted Path

13.8.1 Logon

The `SES_EVENT_LOGON` event is called to start a logon procedure for local system logon. Any process may call this event.

The event may be started with or without information about the user to be logged on. The event may also be started to log on a guest user or to initiate an auto-guest logon. Authorities participating in this event should call `SESWaitEvent` with the event `SES_EVENT_LOGON`.

Note that an auto-guest logon can only be initiated by logon shell services. Also note that the SLD/UIAs are not invoked for an auto-guest logon.

Event Flow

1. SESStartEvent is called to initiate the event.
2. Logon shell services reserves a client user handle for possible future allocation. For auto-guest logon, security enabling services will obtain the guest user name from the environment variable, set the client user handle to -1 and bypass all UIAs.
Go to step 8.
3. Logon shell services calls the system logon driver to obtain the identity of a user identification authority. System logon driver returns the identity of a user identification authority. For a guest user, the system logon driver should indicate there are no more UIAs and return a final authentication status of SES_STATUS_GUEST_USER.
4. Logon shell services unblocks the SESWaitEvent() call of the specified user identification authority with a SES_EVENT_LOGON_UIA event.
5. The user identification authority prompts the user for identification and authorization information.
6. After the user identification authority authenticates the user, the user identification authority calls SESReturnEventStatus() to pass the authentication results back to logon shell services.
7. Logon shell services invokes the system logon driver with the status of the previous user identification authority. Steps (3) through (7) are repeated until the SLD indicates that no more UIAs need to be notified. The system logon driver determines the final user authentication status.
8. LSS unblocks the SESWaitEvent() call for the system logon authority with a SES_EVENT_LOGON_SLA event and provides the user authentication status.
9. The system logon authority specifies (creates if necessary) the appropriate client group/process handles in the SESReturnEventStatus() call with a status of SES_STATUS_NO_ERROR.

Logon shell services associates the local system logon session with the security context established by the system logon authority. The client user handle must be the subject handle reserved by logon shell services and passed to the system logon authority as part of the event data. The agent user/group/process handles are set equal to the client user/group/process handles. All authority flags and the EGF/EPF/EUF/PAF flags are set to zero. The local user flag is set to one.

Note: For auto-guest logon, the subject handles returned by the system logon authority are ignored and all six subject handles (CUH, AUH, CGH, AGH, CPH, APH) are set to -1.

10. Logon shell services calls the client logon driver to obtain the identity of a client logon authority.
11. Client logon driver returns the identity of a client logon authority.
12. Logon shell services unblocks the `SESWaitEvent()` call for the specified client logon authority with an `SES_EVENT_LOGON_CLA` event.
13. After the client logon authority logs the user on to the remote resource, the client logon authority calls `SESReturnEventStatus()`. Steps (10) through (13) are repeated until the client logon driver indicates that no more CLAs should be notified.
14. Logon shell services unblocks the `SESWaitEvent` call for personal shell services with an `SES_EVENT_LOGON` event.
15. If `RESTARTUSERSHELL=YES`, personal shell services notifies `SESShell` to start the user shell. Personal shell services then calls `SESReturnEventStatus()`.
16. LSS unblocks the `SESStartEvent()` thread completing the LOGON event.

Event Data

UserName - SES subject name
 UserNameLen - length of SES subject name
 UserToken - SES subject token
 UserTokenLen - length of SES subject token
 Client User Handle - SES subject handle

Event Data Usage

SESStartEvent

UserName (input) - User name or null if user is not identified.
 UserNameLen (input) - Length of user name or 0 if user is not identified.
 UserToken (input) - User token or null if the token has not been obtained.
 UserTokenLen (input) - Length of token or 0 if the token has not been obtained.
 Client User Handle - N/A

SESWaitEvent

UserName (output) - User name or null if user is not identified.
 UserNameLen (output) - Length of username or 0 if user is not identified.
 UserToken (output) - User token (for example password) or null if the token has not been obtained.
 UserTokenLen (output) - Length of token or 0 if the token has not been obtained.
 Client User Handle (output) - Handle reserved by SES.

SESReturnEventStatus

UserName (input)	- User name if the UIA obtained the user name of a previously unidentified user.
UserNameLen (input)	- Length of user name if the UIA obtained it for a previously unidentified user.
UserToken (input)	- User token if the UIA obtained a previously undefined token.
UserTokenLen (input)	- Length of token if the UIA obtained a previously undefined token.
Client User Handle	- N/A.

Note: The system logon authority specifies (creates if necessary) the appropriate client group/process handles in the SecurityContext of the event structure. The client user handle must be the handle reserved by logon shell services and passed to the system logon authority as part of the event data. The agent user/group/process handles are set equal to the client user/group/process handles.

System Logon Driver Interface

The system logon driver will receive as input Event, Event ID, EventStatus, Logon Event Data (for example, user name, user token, CUH).

SLDQueryUIA (input) the event status may be null, or it may contain SES_STATUS_ID_ONLY or SES_STATUS_GUEST_USER. If a logon is initiated with an event status of SES_STATUS_AUTOGUEST, the system logon driver will be bypassed. The logon event data may contain null data, user name, or user name and password.

SLDQueryUIA (output), SLDQueryUIA may either return an authority ID of the next user identification authority to call, or a return code to indicate there are no more UIAs. If no more UIAs are left, the event status field will contain the final status (SES_STATUS_USER_AUTHENTICATED, SES_STATUS_USER_UNAUTHENTICATED, SES_STATUS_GUEST_USER). If event status was SES_STATUS_GUEST_USER on input, the system logon driver should return an event status of SES_STATUS_GUEST_USER and indicate there are no more UIAs to process. The system logon driver final event status will be propagated to the system logon authority.

Client Logon Driver Interface

The client logon driver will receive as input Event, Event ID, EventStatus, Logon Event Data (UserName, UserToken, CUH).

CLDQueryCLA (input), the EventStatus may be null, or it may contain SES_STATUS_GUEST_USER or SES_STATUS_AUTOGUEST. The logon event data may contain null data, user name, or user name and password.

CLDQueryCLA (output), the client logon driver may either return an authority ID of the next client logon authority to call, or a return code to indicate there are no more CLAs. The event status may be null or it may contain SES_STATUS_GUEST_USER or SES_STATUS_AUTOGUEST. The client logon driver event status will be propagated to the CLAs.

Event Definitions

SES_EVENT_LOGON_UIA

A user identification authority receives notification of this event when being asked to identify and/or authenticate a user for system logon. If the user has not been identified yet, the user identification authority may prompt the user for identification and authentication data.

If the user is already identified, the user identification authority may just authenticate or elect to do nothing. The user identification authority may attempt to change the user name. However, the system logon driver will determine if a name change will be accepted. The OS/2 system logon driver will not care if the user name changes.

After the user identification authority has performed identification and/or authentication, the results are returned to logon shell services in the event status field via an SESReturnEventStatus (or SESReturnWaitEvent) call.

The eventstatus may be one of the following:

- SES_STATUS_USER_AUTHENTICATED - The user identification authority authenticated the user. The user name and user token fields are filled in.
- SES_STATUS_USER_UNAUTHENTICATED - The user identification authority tried, but failed the authentication of the user. User name and user token are undefined.
- SES_STATUS_NOT_APPLICABLE - The authentication services aren't applicable for the specified user.
- SES_STATUS_NOT_AVAILABLE - The authentication services aren't available (for example the authentication server is not operational).
- SES_STATUS_ID_ONLY - The user identification authority only identified the user, but did not authenticate the user. User name is filled in.

- `SES_STATUS_GUEST_USER` - The user identification authority has requested that the user be logged in as guest. The system logon driver policy will determine if guest logon is supported.

SES_EVENT_LOGON_SLA

The system logon authority will receive this event after the UIAs and system logon driver have been given the chance to authenticate the user. The system logon authority will receive the user name and a reserved client user handle. The system logon authority does not receive the user token. The event status field contains the final authentication status from the system logon driver. Although the user identification authority (and system logon driver) determine if a user has been authenticated, ultimately, the system logon authority will determine if a user is granted access to a system.

If logon is continued by the system logon authority for an authenticated user, the system logon authority specifies (creates if necessary) the appropriate client group/process handles in the `SESReturnEventStatus()` call with event status set to `SES_STATUS_NO_ERROR`. Logon shell services associates the local system logon session (PM, user shell) with the security context provided by the system logon authority.

Note: The client user handle must be the handle reserved by logon shell services and passed to the system logon authority as part of the event data. Logon shell services sets the agent user/group/process handles equal to the client user/group/process handles provided by the system logon authority in the `SESEVENT` structure. All authority flags and `EUF/EGF/EPF/PAF` are set to zero. The local user flag is set to one.

If the user was not authenticated or not recognized by the system logon authority, the system logon authority may set event status to `SES_STATUS_GUEST_USER`, which causes the normal logon flow except that the user name associated with the client user handle is set to whatever is specified in the guest name environment variable (or the default guest name).

If the user was not authenticated or not recognized by the system logon authority, the system logon authority may set event status to `SES_STATUS_EVENT_FAILURE`. The `SESStartEvent` returns immediately with the failure status.

SES_EVENT_LOGON_CLA

A client logon authority receives notification of this event after a user has been identified and authenticated to the local system. The client logon authority will receive a user name, user token, and CUHandle. At this point, the client logon authority may perform a remote logon on behalf of the logged on user.

The event status is ignored.

SES_EVENT_LOGON

Personal shell services gets notified with this event when the user has been authenticated by the UIAs and approved by the system logon authority.

13.8.2 Unlock

The SES_EVENT_UNLOCK event is similar to SES_EVENT_LOGON. The UNLOCK event is called to start an unlock procedure for the local system logon session. Any process may call this event.

Authorities participating in this event should call SESWaitEvent with the event SES_EVENT_UNLOCK.

Event Flow

1. SESStartEvent is called to initiate the event.
2. Logon shell services calls the system logon driver to obtain the identity of a user identification authority. System logon driver returns the identity of a user identification authority.
3. Logon shell services unblocks the SESWaitEvent() call of the specified user identification authority with an SES_EVENT_UNLOCK_UIA event.
4. The user identification authority prompts the user for authorization information.
5. After the user identification authority authenticates the user, it calls SESReturnEventStatus() to pass the authentication results back to logon shell services.
6. Logon shell services invokes the system logon driver with the status of the previous user identification authority. Steps (2) through (6) are repeated until the system logon driver indicates that no more UIAs need to be notified. The system logon driver determines the final user authentication status.

7. Logon shell services unblocks the `SESWaitEvent()` call for the system logon authority with an `SES_EVENT_UNLOCK_SLA` event and provides the user authentication status.
8. Logon shell services calls the client logon driver to obtain the identity of a client logon authority.
9. Client logon driver returns the identity of a client logon authority.
10. Logon shell services unblocks the `SESWaitEvent()` call for the specified client logon authority with an `SES_EVENT_UNLOCK_CLA` event.
11. After the client logon authority processes the event, the client logon authority calls `SESReturnEventStatus()`. Steps (8) through (11) are repeated until the client logon driver indicates that no more CLAs should be notified.
12. Logon shell services unblocks the `SESWaitEvent` call for PSS with a `SES_EVENT_UNLOCK` event.
13. Personal shell services calls `SESReturnEventStatus()`.
14. Logon shell services unblocks the `SESStartEvent()` thread completing the `UNLOCK` event.

Event Data

UserToken - SES subject token
 UserTokenLen - length of SES subject token

Note: The user name that will be used for authentication by UIAs or for user identification authority selection by the system logon driver will be the currently logged on user.

Event Data Usage

SESStartEvent

UserToken (input) - User token or null if the token has not been obtained.
 UserTokenLen (input) - User token length or 0 if the token has not been obtained.

SESWaitEvent

UserToken (output) - User token or null if the token has not been obtained.
 UserTokenLen (output) - User token length or 0 if the token has not been obtained.

SESReturnEventStatus

UserToken (input) - User token if the UIA obtained a previously undefined token.
 UserTokenLen (input) - User token length if the UIA obtained a previously undefined token.

System Logon Driver Interface

The system logon driver will receive as input Event, Event ID, EventStatus, Unlock Event Data (user token).

SLDQueryUIA (input), the event status may be null, or it may contain SES_STATUS_GUEST_USER, or the event status will contain the status of the previous user identification authority (SES_STATUS_USER_AUTHENTICATED, SES_STATUS_NOT_APPLICABLE, SES_STATUS_NOT_AVAILABLE, SES_STATUS_USER_UNAUTHENTICATED, SES_STATUS_GUEST_USER).

The unlock event data may contain null data or a token.

SLDQueryUIA (output), the system logon driver may either return an authority ID of the next user identification authority to call, or a return code to indicate there are no more UIAs. If no more UIAs are left, the event status field will contain the final status (SES_STATUS_USER_AUTHENTICATED, SES_STATUS_USER_UNAUTHENTICATED, SES_STATUS_GUEST_USER). The system logon driver final event status will be propagated to the system logon authority.

Client Logon Driver Interface

The client logon driver will receive as input Event, Event ID, EventStatus, Unlock Event Data (user token).

CLDQueryCLA (input), the event status may be null, or it may contain SES_STATUS_GUEST_USER. The Unlock Event Data may contain null data or a password.

CLDQueryCLA (output), the client logon driver may either return an authority ID of the next client logon authority to call, or a return code to indicate there are no more CLAs. The event status may be null or it may contain SES_STATUS_GUEST_USER. The client logon driver event status will be propagated to the CLAs.

Event Definitions

SES_EVENT_UNLOCK_UIA

A user identification authority receives notification of this event when being asked to authenticate a user for unlock. If the token (password) has not been obtained, the user identification authority may prompt the user for authentication data.

After the user identification authority has performed authentication, the results are returned to logon shell services in the event status field via an `SESReturnEventStatus` call.

The event status may be one of the following:

- `SES_STATUS_USER_AUTHENTICATED` - The user identification authority authenticated the user. The user name and user token fields are filled in.
- `SES_STATUS_USER_UNAUTHENTICATED` - The user identification authority tried, but failed the authentication of the user. User name and user token are undefined.
- `SES_STATUS_NOT_APPLICABLE` - The authentication services aren't applicable for the specified user.
- `SES_STATUS_NOT_AVAILABLE` - The authentication services aren't available (for example the authentication server is not operational).
- `SES_STATUS_ID_ONLY` - The user identification authority only identified the user, but did not authenticate the user. User name is filled in.
- `SES_STATUS_GUEST_USER` - The user identification authority may not return this status unless the current system logon user is a guest.

`SES_EVENT_UNLOCK_CLA`

A client logon authority gets this event to be informed that the user is unlocking the local system logon session. If the current local system logon user is a guest, the client logon authority may wish to re-authenticate the user.

The event status is ignored.

`SES_EVENT_UNLOCK_SLA`

The system logon authority will receive this event after the UIAs and system logon driver have been given the chance to authenticate the user. The event status field contains the final authentication status from the system logon driver. Although the user identification authority (and system logon driver) determine if a user has been authenticated, ultimately, the system logon authority will determine if the user is granted access to the system.

If the user was not authenticated, the event status should be set to `SES_STATUS_EVENT_FAILURE`. The `SESStartEvent` is returned to immediately with the failure status.

SES_EVENT_UNLOCK

Personal shell services gets notified with this event when the user has been authenticated by the UIAs and approved by the system logon authority. Personal shell services must remove the lock up bit map at this time.

13.8.3 Logoff, Shutdown

The `SES_EVENT_LOGOFF` and `SES_EVENT_SHUTDOWN` events are similar. Logoff is used as the example in this section, with differences noted for shutdown where appropriate.

These events are called to start a logoff or a shutdown procedure. Any process may call these events.

Authorities participating in these events should call `SESWaitEvent` with the event `SES_EVENT_LOGOFF` or `SES_EVENT_SHUTDOWN`.

Event Flow

1. `SESStartEvent` is called to initiate the event. The process initiating the event is blocked until event completion.
2. Logon shell services unblocks the `SESWaitEvent()` call for the system logon authority to query whether the user is authorized to perform the specified event.
3. The system logon authority calls `SESReturnEventStatus()` to indicate whether the user is authorized to perform the specified event.
4. Logon shell services calls the client logon driver to obtain the identity of a client logon authority.
5. The client logon driver returns the identity of a client logon authority.
6. Logon shell services unblocks the `SESWaitEvent()` call for the specified client logon authority with the `SES_EVENT_LOGOFF_CLA` event.
7. After the client logon authority processes the event (for example, logs the user off of a remote server), the client logon authority calls `SESReturnEventStatus()` to indicate that logon shell services can continue.
8. Logon shell services repeats steps (4) through (7) for subsequent CLAs until the client logon driver indicates that no more CLAs need to be notified.
9. Logon shell services unblocks the `SESWaitEvent()` call for the system logon authority.

10. The system logon authority processes this event and calls `SESReturnEventStatus`. For a logoff, the system logon authority may terminate all processes running on behalf of the current user. For a shutdown, the system logon authority may perform a system shutdown and display a shutdown message.
11. Logon shell services unblocks the `SESWaitEvent()` call for the personal shell services.
12. If `RESTARTUSERSHELL=YES`, personal shell services notifies `SESShell` to terminate the user shell. Personal shell services then calls `SESReturnEventStatus()`.
13. Logon shell services associates PM and the user shell (if active) with the logoff state security context. Logon shell services then unblocks the application that started the event (which indicates that the event is complete).

Event Data

There is no event specific data for this event.

Event Data Usage

Client Logon Driver Interface

The client logon driver will receive as input Event, Event ID.

The client logon driver will return an authority ID of the next client logon authority to call, or a return code to indicate there are no more CLAs.

Event Definitions

SES_EVENT_LOGOFF_CLA

SES_EVENT_SHUTDOWN_CLA

A client logon authority gets this message when the user is logging off or shutting down the system. For a logoff or shutdown event, the client logon authority should detach the user from remote resources.

SES_EVENT_LOGOFF_SLA

SES_EVENT_SHUTDOWN_SLA

For a LOGOFF event the SLA should terminate all processes running on behalf of the local user. For a shutdown event, the system logon authority should perform the actual system shutdown when receiving this message. The system logon authority is responsible at this time to display a pending shutdown message.

SES_EVENT_LOGOFF_QUERY

SES_EVENT_SHUTDOWN_QUERY

The system logon authority receives this message to approve the pending event. If the user is to be queried for confirmation of the event, the system logon authority should perform the query after receiving this message.

SES_EVENT_LOGOFF

SES_EVENT_SHUTDOWN

Personal shell services gets notified with this event when the user is about to be logged off or the system is shutting down. Shutdown implies a logoff. Personal shell services is responsible for shutting down the user shell in this case. Personal shell services should manage the screen at this point and display the logoff bit map.

13.8.4 Lock

The SES_EVENT_LOCK event is called to start a lock procedure. Any process may call these events.

Authorities participating in these events should call SESWaitEvent with the event SES_EVENT_LOCK.

Event Flow

1. SESStartEvent is called to initiate the event. The process initiating the event is blocked until event completion.
2. Logon shell services unblocks the SESWaitEvent() call for the system logon authority to query whether the user is authorized to perform the specified event.
3. The system logon authority calls SESReturnEventStatus() to indicate whether the user is authorized to perform the specified event.
4. Logon shell services calls the client logon driver to obtain the identity of a client logon authority.

5. The client logon driver returns the identity of a client logon authority.
6. Logon shell services unblocks the `SESWaitEvent()` call for the specified client logon authority with the `SES_EVENT_LOCK_CLA` event.
7. After the client logon authority processes the event (for example logs the user off of a remote server), the client logon authority calls `SESReturnEventStatus()` to indicate that logon shell services can continue.
8. Logon shell services repeats steps (4) through (7) for subsequent CLAs until the client logon driver indicates that no more CLAs need to be notified.
9. Logon shell services unblocks the `SESWaitEvent()` call for the system logon authority.
10. The system logon authority processes this event and calls `SESReturnEventStatus`.
11. Logon shell services unblocks the `SESWaitEvent()` call for the personal shell services.
12. Personal shell services processes this event and calls `SESReturnEventStatus()`.
13. Logon shell services unblocks the application that started the event (which indicates that the event is complete).

Event Data

There is no event specific data for this event.

Event Data Usage

Client Logon Driver Interface

The client logon driver will receive as input Event, Event ID.

The client logon driver will return an authority ID of the next client logon authority to call, or a return code to indicate there are no more CLAs.

Event Definitions

SES_EVENT_LOCK_CLA

A client logon authority gets this message when the user is locking the system.

SES_EVENT_LOCK_SLA

A system logon authority gets this message when the user is locking the system.

SES_EVENT_LOCK_QUERY

The system logon authority receives this message to approve the pending event. If the user is to be queried for confirmation of the event, the system logon authority should perform the query after receiving this message.

SES_EVENT_LOCK

Personal shell services gets notified of this event when the system is about to go into a locked state. Personal shell services should manage the screen at this point and display the locked bit map.

13.8.5 Change Password

The `SES_EVENT_CHANGE_PASSWORD` event is provided to facilitate password synchronization among multiple cooperating components.

This event supports of the following situations:

- A user changes her/his own password. In this case, the user making the change and the target user are the same (and the name of the user can be obtained from the security context), so the user would typically be prompted only for the old password and the new password (twice for confirmation).
- A system administrator changes the password for another user. In this case, the user making the change is different from the target user, so the system administrator would typically be prompted for the target user's name, the system administrator's password (not the target user's old password), and the target user's new password (twice for confirmation).

Authorities participating in this event should call `SESWaitEvent` with the event `SES_EVENT_CHANGE_PASSWORD`.

Event Flow

1. `SESStartEvent` is called to initiate the event, with the current and new passwords.
2. Logon shell services calls the password validation driver to validate the new password.

3. The password validation driver validates the password and returns the status to logon shell services.
4. Logon shell services calls the system logon driver to obtain the identity of a user identification authority.
5. The system logon driver returns the identity of a user identification authority.
6. Logon shell services unblocks the `SESWaitEvent()` call for the specified user identification authority.
7. After the user identification authority processes the event, the user identification authority calls `SESReturnEventStatus()` to indicate that logon shell services can continue.
8. Logon shell services repeats steps (4) through (7) for subsequent UIAs until the system logon driver indicates that no more UIAs need to be notified.
9. Logon shell services calls the client logon driver to obtain the identity of a client logon authority.
10. The client logon driver returns the identity of a client logon authority.
11. LSS unblocks the `SESWaitEvent()` call for the specified client logon authority.
12. After the client logon authority processes the event, the client logon authority calls `SESReturnEventStatus()` to indicate that logon shell services can continue.
13. Logon shell services repeats steps (9) through (12) for subsequent CLAs until the client logon driver indicates that no more CLAs need to be notified.
14. Logon shell services unblocks the application that started the event completing the flow.

Event Data

ChangerName	- name of user making the password change (standard SES user name format)
ChangerNameLen	- length of ChangerName
ChangerPassword	- password of user making the password change
ChangerPasswordLen	- length of ChangerPassword
UserName	- name of target user
UserNameLen	- length of UserName
UserPassword	- new password for target user
UserPasswordLen	- length of UserPassword

Event Data Usage

SESStartEvent

ChangerName (input) - The name of the user making the password change.
ChangerNameLen (input) - The name length of the user making the password change.
ChangerPassword (input) - The password of the user making the password change.
ChangerPasswordLen (input) - The password length of the user making the password change.
UserName (input) - The target user name.

Note: This name will match the ChangerName for a user changing their own password.

UserNameLen (input) - The target user name length.
UserPasswordSecret (input) - The target user's new password.
UserPasswordLen (input) - The target user's new password length

SESWaitEvent

ChangerName (output) - The name of the user making the password change.
ChangerNameLen (output) - The name length of the user making the password change.
ChangerPassword (output) - The password of the user making the password change.
ChangerPasswordLen (output) - The password length of the user making the password change.
UserName (output) - The target user name.

Note: This name will match the ChangerName for a user changing their own password.

UserNameLen (output) - The target user name length.
UserPassword (output) - The target user's new password.
UserPasswordLen (output) - The target user's new password length

SESReturnEventStatus

N/A

System Logon Driver Interface

The system logon driver will receive as input Event, Event ID, Change Password Event Data (ChangerName Name, ChangerPassword, UserName, UserPassword).

The system logon driver will return an authority ID of the next user identification authority to call, or a return code to indicate there are no more UIAs.

Client Logon Driver Interface

The client logon driver will receive as input Event, Event ID, Change Password Event Data (ChangerName Name, ChangerPassword, UserName, UserPassword).

The client logon driver will return an authority ID of the next client logon authority to call, or a return code to indicate there are no more CLAs.

Password Validation Driver Interface

The password validation driver will receive as input Change Password Event Data (ChangerName Name, ChangerPassword, UserName, UserPassword).

The password validation driver will return a return code of SES_NO_ERROR or SES_EVENT_FAILURE.

Event Definitions

SES_EVENT_CHANGE_PASSWORD

The change password event is received by UIAs and CLAs waiting on this event.

The event status is ignored.

13.8.6 Create User Profile, Delete User Profile

The SES_EVENT_CREATE_PROFILE and SES_EVENT_DELETE_PROFILE events are similar so they are discussed as one. The text CREATE will be used in this discussion.

Create user profile is called when a new user is to be added to the system. SCAs waiting on this event may perform any new user processing as necessary.

Delete user profile is called when a user is to be deleted from system. SCAs waiting on this event may perform any delete processing as necessary.

Authorities participating in this event should call SESWaitEvent with the event SES_EVENT_CREATE_PROFILE or SES_EVENT_DELETE_PROFILE.

Event Flow

1. SESStartEvent is called to initiate the event.
2. Logon shell services calls the system logon authority to notify of the event.

3. System logon authority processes the event, calls `SESReturnEventStatus`.
4. Logon shell services calls the system logon driver to obtain the identity of a user identification authority.
5. The system logon driver returns the identity of a user identification authority.
6. Logon shell services unblocks the `SESWaitEvent()` call for the specified user identification authority.
7. After the user identification authority processes the event, the user identification authority calls `SESReturnEventStatus()` to indicate that logon shell services can continue.
8. Logon shell services repeats steps (4) through (7) for subsequent UIAs until the system logon driver indicates that no more UIAs need to be notified.
9. Logon shell services calls the client logon driver to obtain the identity of a client logon authority.
10. The client logon driver returns the identity of a client logon authority.
11. Logon shell services unblocks the `SESWaitEvent()` call for the specified client logon authority.
12. After the client logon authority processes the event, the client logon authority calls `SESReturnEventStatus()` to indicate that logon shell services can continue.
13. Logon shell services repeats steps (9) through (12) for subsequent CLAs until the client logon driver indicates that no more CLAs need to be notified.
14. Logon shell services unblocks the application that started the event completing the flow.

Event Data

`UserName` - name of user associated with new profile (standard SES `UserName` format)
`UserNameLen` - length of user name

Event Data Usage

`SESStartEvent`

`UserName` (input) - Name of new user.
`UserNameLen` (input) - Name length of new user.

`SESWaitEvent`

UserName (output) - Name of new user.
UserNameLen (output) - Name length of new user.

SESReturnEventStatus

N/A

System Logon Driver Interface

The system logon driver will receive as input Event, Event ID, Create/Delete User Profile Event Data (UserName).

The system logon driver will return an authority ID of the next user identification authority to call, or a return code to indicate there are no more UIAs.

Client Logon Driver Interface

The client logon driver will receive as input Event, Event ID, Create/Delete User Profile Event Data (UserName).

The client logon driver will return an authority ID of the next client logon authority to call, or a return code to indicate there are no more CLAs.

Event Definitions

SES_EVENT_CREATE_PROFILE

SES_EVENT_DELETE_PROFILE

The system logon authority, all UIAs, and all CLAs registered to receive these events will be notified when these events are invoked (and can then take the appropriate action, for example creating/deleting user profiles in their own databases).

13.8.7 Identification and Authentication

The SES_EVENT_IA event may be invoked when a user needs to be authenticated for purposes other than local system logon/unlock, for example when a second signature is required for a banking transaction.

The identification and authentication event has no affect on the state of the local system logon session.

Authorities participating in this event should call `SESWaitEvent` with the event `SES_EVENT_IA`.

Event Flow

1. Event is initiated by `SESStartEvent`.
2. Logon shell services calls the system logon driver to obtain the identity of a user identification authority.
3. System logon driver returns the identity of a user identification authority.
4. Logon shell services unblocks the `SESWaitEvent()` call of the specified user identification authority.
5. After the user identification authority authenticates the user, it calls `SESReturnEventStatus()` to pass the authentication results back to logon shell services.
6. Logon shell services invokes the system logon driver with the status of the previous user identification authority. Steps 2 through 6 are repeated until the system logon driver indicates that no more UIAs need to be notified. The system logon driver determines the final user authentication status.
7. Logon shell services unblocks the application that called `SESStartEvent()` indicating that the event is complete. The return code will indicate if the user was authenticated or not authenticated.

Event Data

`UserName` - SES user name
`UserNameLen` - SES user name length.
`UserToken` - SES token
`UserTokenLen` - SES token length

Event Data Usage

SESStartEvent

`UserName` (input) - User name or null.
`UserNameLength` (input) - User name length or null.
`UserToken` (input) - User token or null.
`UserTokenLen` (input) - User token length or null.

SESWaitEvent

UserName (output) - User name or null if the user has not been identified.
 UserNameLength (output) - User name length or null if the user has not been identified.
 UserToken (output) - User token or null if the token has not been obtained.
 UserTokenLen (output) - User token length or null if the token has not been obtained.

SESReturnEventStatus

UserName (input) - User name or null if user has not been identified.
 UserNameLen (input) - User name length or null if user has not been identified.
 UserToken (input) - User token or null if token has not been obtained.
 UserTokenLen (input) - User token length or null if token has not been obtained.

System Logon Driver Interface

The system logon driver will receive as input Event, Event ID, EventStatus, Logon Event Data (for example, user name, user token).

SLDQueryUIA (input), the IA event data may contain the user name and Password. The event status may contain the status of the previous user identification authority if it is not the first call.

SLDQueryUIA (output), the system logon driver may either return an authority ID of the next user identification authority to call, or a return code to indicate there are no more UIAs. If no more UIAs are left, the event status field will contain the final status (SES_STATUS_USER_AUTHENTICATED, SES_STATUS_USER_UNAUTHENTICATED).

Event Definitions

SES_EVENT_IA

A user identification authority receives notification of this event when being asked to identify and/or authenticate a user for an identification and authentication. If the user has not been identified yet, the user identification authority may prompt the user for identification and authentication data. If the user is already identified, the user identification authority may just authenticate or elect to do nothing.

After the user identification authority has performed identification and/or authentication, the results are returned to logon shell services in the event status field via an SESReturnEventStatus call.

The event status may be one of the following:

- `SES_STATUS_USER_AUTHENTICATED` - The user identification authority authenticated the user. The user name and user token fields are filled in.
- `SES_STATUS_USER_UNAUTHENTICATED` - The user identification authority tried, but failed the authentication of the user. User name and user token are undefined.
- `SES_STATUS_NOT_APPLICABLE` - The authentication services aren't applicable for the specified user.
- `SES_STATUS_NOT_AVAILABLE` - The authentication services aren't available (for example the authentication server is not operational).
- `SES_STATUS_ID_ONLY` - The user identification authority only identified the user, but did not authenticate the user. User name is filled in.

13.8.8 Send Security Context

The `SES_EVENT_SEND_SECURITY_CONTEXT` event is called to send a message, with the effective security context of the sending thread, to a security control authority. The send security context event is triggered by a call to `SESSendSecurityContext()`.

Event Flow

1. An security control authority calls `SESWaitEvent` specifying `SES_EVENT_SEND_SECURITY_CONTEXT`.
2. A program calls `SESSendSecurityContext()` with the authority ID of the target SCA.
3. Logon shell services unblocks the `SESWaitEvent()` call of the target security control authority. (Note: If the security control authority has server process authority, the security control authority will be associated with the effective security context of the calling thread when it unblocks).
4. The security control authority calls `SESReturnEventStatus()`.
5. Logon shell services unblocks the `SESSendSecurityContext()` thread completing the `SEND SECURITY CONTEXT` event.

Event Data

`Message` - Pointer to message buffer.
`MessageLength` - Length of message
`Authority ID` - Target SCA authority ID

Event Data Usage

SESWaitEvent

Message (input) - Pointer to message buffer.
MessageLength (output) - Length of message.
Authority ID - N/A

SESReturnEventStatus

Message (input) - Pointer to return message buffer.
MessageLength (input) - Length of return message.
Authority ID - N/A

Event Definitions

See Chapter 12, “Security Context Services (SCS)” on page 161 for a complete description of this event.

13.8.9 Process Creation

The `SES_EVENT_PROCESS_CREATION` event is triggered by security enabling services when a new security control authority (or a process that is not associated with the local system logon session) is created. Only the system logon authority may wait on the process creation event. The system logon authority receives the program name from the event data.

The system logon authority may set (create if necessary) the client/agent user/group/process handles and agent process authority flag in the maximum and effective security contexts, and set/clear EUF/EGF/EPF in the effective security context.

Note: For most processes started by users, the expected policy is that the child process will inherit its client user/group/process handles from the parent process. However, for processes that are not started by a user (for example, jobs scheduled to run periodically by a system administrator), the child process may not inherit its client user/group/process handles from the parent process.

Event Flow

1. The system logon authority calls `SESWaitEvent()` specifying `SES_EVENT_PROCESS_CREATION`.
2. Security enabling services generates a `SES_EVENT_PROCESS_CREATION` event when a new authority process is created.
3. Logon shell services unblocks the system logon authority thread waiting on the `SES_EVENT_PROCESS_CREATION` event.

4. The system logon authority may set (create if necessary) the client/agent user/group/process handles, and set/clear the EUF, EGF, EPF, and APA flags.
5. The system logon authority calls `SESReturnEventStatus()` with the security context of the requesting authority.
6. Logon shell services unblocks the security enabling services thread completing the `SES_EVENT_PROCESS_CREATION` event.

Event Data

`ProgramName` - Path name of program file for newly created process.

`CUH` - Client User Handle

`AUH` - Agent User Handle

`CGH` - Client Group Handle

`AGH` - Agent Group Handle

`CPH` - Client Process Handle

`APH` - Agent Process Handle

`SCS` - Security Context Status

Event Data Usage

`SESWaitEvent`

`ProgramName` (output) - Path name of program file for newly created process.

`CUH` (output) - Inherited from parent's `CUH`.

`AUH` (output) - Inherited from parent's `CUH` or `AUH` as specified by parent's PAF.

`CGH` (output) - Inherited from parent's `CUH`.

`AGH` (output) - Inherited from parent's `CUH` or `AUH` as specified by parent's PAF.

`CPH` (output) - Inherited from parent's `CUH`.

`APH` (output) - Inherited from parent's `CUH` or `AUH` as specified by parent's PAF.

`SCS` (output) - Specified in `SECURE.SYS` or inherited from parent's `SCS` as specified by parent's PAF.

`SESReturnEventStatus`

`ProgramName` - N/A

`CUH` (input) - Specified by SLA.

`AUH` (input) - Specified by SLA.

`CGH` (input) - Specified by SLA.

`AGH` (input) - Specified by SLA.

`CPH` (input) - Specified by SLA.

`APH` (input) - Specified by SLA.

`SCS` (input) - Specified by SLA (only EUF, EGF, EPF, and APA flags may be set/cleared).

Event Definitions

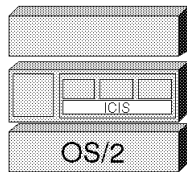
Refer to Chapter 12, “Security Context Services (SCS)” on page 161 for a complete description of this event.

13.8.10 Trusted Path

The trusted path callout to the installable security subsystem security kernel is triggered by the local system user typing Ctrl-Alt-Del (or Ctrl-Alt-NumLock-NumLock). The installable security subsystem security kernel can work with the ISS security daemon to control keyboard/mouse input, thereby providing a trusted path for the local system user to invoke installable security subsystem security services (for example, logon).

1. CONFIG.SYS contains the following line: SET TRUSTEDPATH=YES.
2. The installable security subsystem security kernel calls DevHlp_Security with the function code DHSEC_SETIMPORT and provides the address of its TRUSTEDPATHCONTROL function.
3. The installable security subsystem security daemon (which is providing the trusted path services in this example) calls a private API supported by the installable security subsystem security kernel and is blocked in the installable security subsystem security kernel.
4. A Ctrl-Alt-Del is hit. The keyboard device driver invokes the installable security subsystem security kernel TRUSTEDPATHCONTROL function.
5. TRUSTEDPATHCONTROL unblocks the installable security subsystem security daemon's trusted path thread and returns an instruction to the keyboard device driver not to reboot.
6. The installable security subsystem trusted path application turns off keyboard monitors (via the SESControlKBDMonitors() API) and then invokes its trusted path logic (for example, initiates a logon event).
7. After the trusted path logic processing is complete, the installable security subsystem trusted path application turns keyboard monitors back on and then calls back into the installable security subsystem security kernel to wait for another Ctrl-Alt-Del.

Chapter 14. Installation, Configuration, Initialization Support



The following sections discuss the system design for the installation, configuration, and initialization support.

14.1 Installation

The security enabling services facility makes use of some of the OS/2 components (OS2KRNL, DOSCALL1.DLL, PMWP.DLL, NWIAPI.DLL, MOUSE.SYS, KBD01.SYS, KBD02.SYS). Because security enabling services requires that these components have been modified to include the necessary enhancements to support security enabling services, a pre-requisite fixpak level will be required for some releases of OS/2.

For example, for OS/2 Version 2.11, security enabling services is supported from fixpak XR_B100. This fixpak was the first fixpak to include the necessary base code modifications for supporting security enabling services.

The security enabling services installation process will copy several security enabling services files (security enabling services device driver, security enabling services daemon programs, security enabling services dynamic link libraries, etc.) to the appropriate drive/directories. For OS/2 Version 2.11, these security specific files are available in fixpak XR_BSES.

Note: The security enabling services installation process will not make the changes to CONFIG.SYS that are necessary to enable security enabling services. When a customer installs a security product (ISS) that requires security enabling services, the installable security subsystem installation process must include the modifications to CONFIG.SYS (and SECURE.SYS) to enable security enabling services.

14.2 Configuration

The following sections describe the necessary modifications to CONFIG.SYS and SECURE.SYS to enable security enabling services, an installable security subsystem, and other security-related components.

14.2.1 CONFIG.SYS

The required modifications for CONFIG.SYS and the backup \OS2\INSTALL\CONFIG.SYS file are as follows. The example statements must appear first in these system configuration files to properly initialize the security enabling components.

```
REM ***** BEGIN SECURITY SUBSYSTEM REQUIREMENTS *****
REM
REM The additions/modifications to CONFIG.SYS to install SES should
REM be inserted at the beginning of CONFIG.SYS. The order of the
REM statements within this block should be preserved.
REM In this example, 'C:' is the boot drive.

REM To activate the SES device driver:
BASEDEV=SESDD32.SYS

REM To activate the ISS device driver:
BASEDEV=ISSDD32.SYS

REM To activate the SES daemon (must be the first CALL= statement):
CALL=C:\OS2\SECURITY\SES\SESSTART.EXE C:\OS2\SECURITY\SES\SESDMON.EXE

REM The PROTSHELL argument is a standard OS/2 CONFIG.SYS entry. It will
REM appear in CONFIG.SYS whether SES is enabled or not. To enable SES,
REM the original PROTSHELL statement must be replaced with the following:
PROTSHELL=C:\OS2\SECURITY\SES\SESSHLL.EXE

REM The RUNWORKPLACE environment variable is a standard OS/2 CONFIG.SYS
REM entry. It will appear in CONFIG.SYS whether SES is enabled or not.
REM To enable SES, the original RUNWORKPLACE statement must be replaced
REM with the following:
SET RUNWORKPLACE=C:\OS2\SECURITY\SES\PSSDMON.EXE

REM The SESDBPATH environment variable specifies the location of SES.LOG,
REM SECURE.SYS, and the SLD/CLD/PVD DLLs. SESDBPATH must be specified.
SET SESDBPATH=C:\OS2\SECURITY\SESDB

REM The AUTOQUEST environment variable enables/disables auto-guest logon.
SET AUTOQUEST=NO
```



```

REM The GUESTNAME environment variable defines the guest user id.
SET GUESTNAME=GUEST

REM The TRUSTEDPATH environment enables/disables trusted path detection.
SET TRUSTEDPATH=NO

REM The USERSHELL environment variable defines the user shell.
SET USERSHELL=C:\OS2\PMHELL.EXE

REM The RESTARTUSERSHELL environment variable specifies the user shell
REM restart option.
SET RESTARTUSERSHELL=YES

REM The BACKGROUNDBITMAP environment variable specifies what bitmap is to
REM be displayed when no local system user is currently logged on or when
REM the local system user interface is locked.
SET BACKGROUNDBITMAP=C:\BITMAP\CUSTOMER.BMP

REM ***** END SECURITY SUBSYSTEM REQUIREMENTS *****

```

14.2.2 SECURE.SYS

SECURE.SYS contains all security context authority (SCA) programs that need to be registered with security enabling services. A security control authority is a program/process that has any of the security enabling services authority flags set.

Note: Only a system administrator should have access to the SECURE.SYS file (like any other file that is part of the trusted computing base, for example CONFIG.SYS, device drivers, file system handlers, OS/2 kernel). An installable security subsystem must enforce this policy or the security of the system is exposed.

Each security control authority (except agent process authority) is specified in SECURE.SYS with a unique authority tag (/SCA="xxxx"). During initialization, each security control authority will be assigned a unique authority ID as specified by the authority tag in SECURE.SYS. Multiple cooperating SCA programs/processes can have the same authority ID to act as a single security control authority. This is accomplished by associating the cooperating programs/processes with the same authority tag (/SCA="xxxx") in SECURE.SYS.

The SECURE.SYS file is written in the following format:

- Comment lines must begin with a semi-colon in column 1.

- Command lines must begin in column 1 and appear as follows ("X:" indicates the boot drive):

```
X:\Path...\Filename.Ext /SCA='Auth Tag' [/APA] [/SPA] [/RLA] ...
... [/SLA] [/ACA] [/CLA] [/UIA] [/PSS] [/SES] ...
... [/START] [/PROPAGATE=YES/NO] [/LOCALUSER=YES/NO]
```

- If a line begins improperly with a blank, /, or any other invalid file specification character, the line is ignored.
- The tags have the following definitions:

/SCA="Auth Tag"	Identifies the authority tag associated with a program/process. The /SCA= tag can be any printable ASCII string with a maximum of 8 characters. The /SCA= tag is used to identify an security control authority program/process (except an agent process authority) and is also used to enable multiple programs/processes to act as a single security control authority. All security control authority programs/processes (except an agent process authority) need to define a /SCA= tag. All SCA programs/processes that need to act as a single SCA must define the same /SCA= tag.
/ACA	Identifies the program/process as an ACA.
/APA	Identifies the program/process as an APA.
/CLA	Identifies the program/process as a CLA.
/PSS	Identifies the program/process as the PSS daemon.
/RLA	Identifies the program/process as an RLA.
/SES	Identifies the program/process as the SES daemon.
/SLA	Identifies the program/process as the SLA.
/SPA	Identifies the program/process as an SPA.
/UIA	Identifies the program/process as a UIA.
/LOCALUSER=YES NO	Indicates whether the program/process can communicate with the local system logon user through PM/WPS interfaces or not.

	The default is inherited from the parent process, but this flag overrides the default.
/PROPAGATE=YES NO	Allows the program/process to specify the inheritance policy for a child process. The default is no (for example, the program/process can not specify the inheritance policy for a child process).
/START	Indicates that the program should be started by security enabling services during initialization. The default is that security enabling services will not start the program.

Note: Programs with /START are started after PM is initialized.

- A fully qualified file name (drive, path, and file name with extension) must be specified otherwise the line is ignored.
- Only files with .COM or .EXE file extension can be specified otherwise the line is ignored.
- If no options are specified, the line is ignored since the authority for the program is indeterminate.
- If an invalid or unknown option is specified, the line is ignored.
- If one type of option appears multiple times on a line, the line is ignored.
- At least one program must specify the SES option. If not, SES does not initialize. This is an unrecoverable condition.
- At least one program must specify the PSS option. If not, SES does not initialize. This is an unrecoverable condition.
- At least one program must specify the SLA option. If not, SES does not initialize. This is an unrecoverable condition.
- A maximum of 244 unique /SCA= tags can be specified in SECURE.SYS.
- A fully qualified file name may appear only once. Lines with the same fully qualified file name are considered duplicate lines. Only the last duplicate line is accepted. All other duplicate lines are ignored.

Following is an example of a SECURE.SYS file:

SES definition

```
C:\OS2\SECURITY\SES\SESDMON.EXE /SCA='SES' /SES
```

PSS definition

```
C:\OS2\SECURITY\SES\PSSDMON.EXE /SCA='PSS' /PSS
```

```
C:\OS2\SECURITY\SES\SESSHLL.EXE /SCA='PSS' /PSS
```

SLA definition

```
C:\OS2\SECURITY\SES\SLA.EXE /SCA='SLA' /SLA /START
```

Single program defined with multiple authorities

```
C:\OS2\SECURITY\SES\XXX.EXE /SCA='XXX' /UIA /CLA /SPA
```

Set of programs defined as the same SCA

```
C:\OS2\SECURITY\SES\YYY1.EXE /SCA='YYY' /CLA /LOCALUSER=YES
```

```
C:\OS2\SECURITY\SES\YYY2.EXE /SCA='YYY' /SPA /LOCALUSER=NO
```

APA definition (authority tag is not specified)

```
C:\OS2\SECURITY\SES\ZZZ.EXE /APA /PROPAGATE=YES
```

If a developer produced a program, for example SESMON, to monitor process-user associations under security enabling services, this program would be given the appropriate security control services status through the SECURE.SYS file. Whether or not SESMON would be able to monitor the associations would depend upon the authority it is given in the SECURE.SYS file.

For example, if SESMON.EXE is executed from the directory C:\SESMON, adding the following line to SECURE.SYS would give it system logon authority.

```
C:\SESMON\SESMON.EXE /SCA='sla' /SLA
```

14.3 Initialization

The initialization process is considered to have the following distinct phases:

Phase 1: In phase 1, the system has been powered on or rebooted. This phase can be interrupted and made potentially insecure should the user hit ALT-F1 during the system boot. This keystroke combination replaces the currently active system configuration files with copies of the files that existed at the time of OS/2 installation. These replacement files are CONFIG.SYS, OS2.INI and OS2SYS.INI and can be found in the \OS2\INSTALL subdirectory. For secure installations, these files must be

replaced with files that will provide a secure environment. To facilitate recovery, it is recommended that the system configuration established by these replacement files fulfill only the minimum functional requirements of the system. For example, these files could force the system into a state that requires a system administrator to logon.

Assuming that the process is not interrupted, initialization continues loading device drivers called out in CONFIG.SYS, including any installable security subsystem device drivers. Next, it reaches the point when programs listed in CONFIG.SYS with RUN= and CALL= statements are executed.

- Phase 2: During phase 2, the SES daemon is started by SESSTART (which is the first CALL= or RUN= statement in CONFIG.SYS), the security enabling services daemon waits to receive notification that PM is enabled from SESSHELL.
- Phase 3: During phase 3, the program specified on the PROTSHELL statement of CONFIG.SYS (SESSHELL) is executed by the system initialization process. SESSHELL creates the system message queue for PM and notifies the security enabling services daemon that PM is enabled. SESSHELL then reads the SECURE.SYS file and starts all security control authority programs (SLA, UIA, CLA, etc.) with the /START option specified.
- Phase 4: In phase 4, the system is now in a steady state, ready to process security enabling services events or respond to security enabling services APIs/KPIs. For example, the personal shell services daemon is waiting for keyboard/mouse activity to initiate a logon event.

An installable security subsystem typically includes several security daemons (SLA, UIA, ACA, etc.) and a security kernel (device driver). These components must be initialized to enforce security policy prior to allowing a user to logon.

- The installable security subsystem security kernel (device driver) is defined in CONFIG.SYS and is initialized when the device driver is loaded during the processing of CONFIG.SYS.
- An access control authority calls the SESCreateHandleNotify() and SESDeleteHandleNotify() APIs to register for notification of the creation/deletion of security enabling services subject handles.
- All other security daemons call the SESRegisterDaemon() API to register for notification of security enabling services events via the

SESWaitEvent() API. This allows the daemon to participate in SES events, for example the system logon authority can be notified when processes are created to provide trusted program support, UIAs can be notified when a user needs to be authenticated so that they can provide the authentication services, etc.

The following system initialization example shows how/when key security relevant processes are started (as specified in CONFIG.SYS and SECURE.SYS).

SAMPLE CONFIG.SYS STATEMENTS FOR SES and ISS

```
-----  
call=c:\os2\security\ses\sesstart.exe  
call=c:\os2\security\ses\sla_0.exe  
protshell=c:\os2\security\ses\sesshell.exe  
set runworkplace=c:\os2\security\ses\pssdmon.exe  
set usershell=c:\os2\pmshell.exe
```

SAMPLE SECURE.SYS STATEMENTS FOR SES and ISS

```
-----  
c:\os2\security\ses\sesdmon.exe /sca='ses' /ses  
c:\os2\security\ses\pssdmon.exe /sca='pss' /pss  
c:\os2\security\ses\sesshell.exe /sca='pss' /pss  
c:\os2\security\ses\sla_1.exe /sca='sla' /sla  
c:\os2\security\ses\sla_2.exe /sca='sla' /sla /start
```

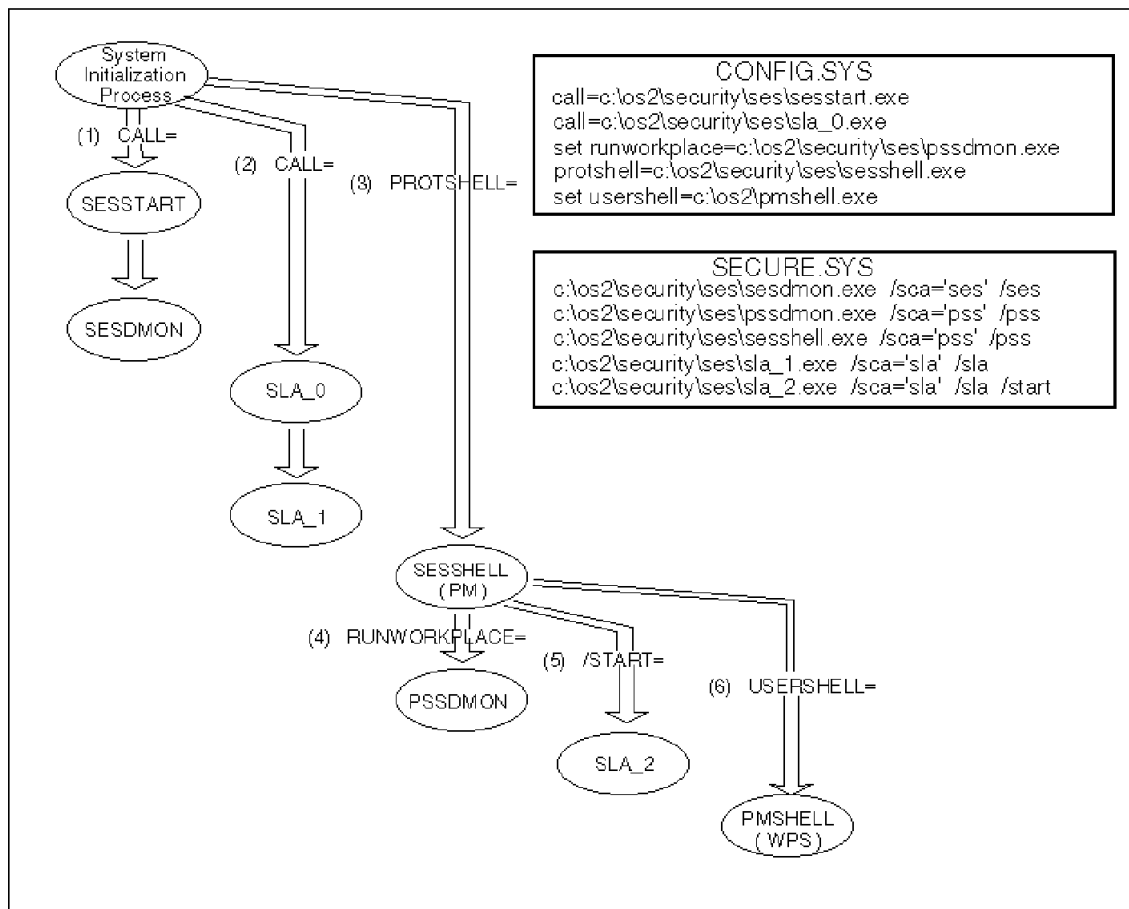


Figure 34. ICIS System Design - Example of SES and ISS Initialization

Part 3. Appendices

Chapter 15. KPI and API Calls

About This Part

Part 1, “Developer’s Guide” on page 1 and Part 2, “Design Notes” on page 79 provided the background information and detail of security enabling services (SES) and how to build a installable security subsystem (ISS). This section provides information on the actual API calls provided, the error codes, and some details of known customer requirements for workstation security products.

15.1 Chapter Breakdown

This appendices section contains details of all the Security Enabling Services APIs that are currently provided by IBM for each of the Security Enabling Services components.

The chapters in this part are:

- Appendix A, “Security Kernel Services KPI Details”
This chapter provides details of the security kernel services KPI calls.
- Appendix B, “Security Context Services (SCS) API Details”
This chapter provides details of the security control services API calls.
- Appendix C, “System Logon Driver API Details”
This chapter provides details of the system logon driver API calls.
- Appendix D, “Client Logon Driver API Details”
This chapter provides details of the client logon driver API calls.
- Appendix E, “Password Validation Driver API Details”
This chapter provides details of the password validation driver API calls.
- Appendix F, “Logon Shell Services API Details”
This chapter provides details of the logon shell services API calls.
- Appendix G, “Security Enabling Services Error Codes”
This chapter provides details of the security enabling services error codes.
- Appendix H, “Customer Thoughts on Security Products”

This chapter provides information on some of the customer requirements for security products that have been passed to IBM by our large accounts.

Appendix A. Security Kernel Services KPI Details

The Kernel Programming Interfaces for the Security Enabling Services are key to it's strength. IBM customers believe that the general publication of these interfaces may jeopardize the security of the system. Because of this, the distribution of this information will be done via a controlled process. If you need this documentation, please send a request to:

IBM Manager of OS/2 Security
Mail Stop 9171
11400 Burnet Rd.
Austin, Texas 78758

Appendix B. Security Context Services (SCS) API Details

This section details the functions contained in the security control services API.

B.1 SESControlProcessCreation

APIRET SESControlProcessCreation(ULONG ulActionCode)

This function enables the system logon authority to inhibit the creation of all new processes on the system and conversely releases the system to begin creating new processes again.

Parameters:

ulActionCode (ULONG) - Input

Indicates whether the function inhibits the creation of new processes or release the system to permit new process creation.

ulActionCode

Value	Definition
0	directs the function to release the system to once again permit process creation
1	directs the function to perform inhibit process creation

Returns

Standard OS/2 API return codes.

0 NO_ERROR

SES API return codes.

35367 SES_INVALID_PARAMETER
35517 SES_INVALID_AUTHORITY

Remarks

This call requires system logon authority or remote logon authority.

B.2 SESSetupHandleNotify

APIRET APIENTRY SESSetupHandleNotify(PSUBJECTINFO pSubjectInfo)

This function allows a process/thread to register for notification of handle creation.

Parameters:

pSubjectInfo - Output

The address of subject information. The buffer is defined as:

Handle - (HSUBJECT) output

The SES handle that has been created.

Instance - (HSUBJECT) output

Not used.

Name - (INFOENTRY) output

Subject name associated with the handle.

Token - (INFOENTRY) output

Not used.

Source - (ULONG) output

Contains the authority ID of the SCA that created the handle.

Returns

Standard OS/2 API return codes.

0 NO_ERROR

SES API return codes.

35517 SES_INVALID_AUTHORITY
35391 SES_BUFFER_OVERFLOW

Remarks

This call requires access control authority. This call is held captive in the security enabling services device driver until a handle has been created.

Upon return, the access control authority can take whatever action is appropriate, for example add the user's credentials (for example user ID,

group ID(s), etc.) and associated subject handle to a cache of active users/handles on the system. The call must then be re-issued to continue receiving notifications. Any handle creations done while the security control authority is processing the notification are queued up until the call is re-issued.

B.3 SESSetupInstanceHandle

APIRET APIENTRY SESSetupInstanceHandle(PHSUBJECT pSubjectHandle)

This function creates an instance handle for the specified subject handle.

Parameters:

pSubjectHandle - (PHSUBJECT) input/output

Address of a subject handle. On input, this is the subject handle for which an instance will be created. On output, the subject handle contains the value of the new instance handle.

Returns

Standard OS/2 API return codes.

0	NO_ERROR
---	----------

SES API return codes.

35286	SES_INVALID_HANDLE
35367	SES_INVALID_PARAMETER
35517	SES_INVALID_AUTHORITY

Remarks

This call requires client logon authority

The created instance handle is deleted by security control services automatically when all processes/threads using the handle have terminated.

B.4 SESSubjectHandle

APIRET APIENTRY SESSubjectHandle(PSUBJECTINFO pSubjectInfo)

This function creates a subject handle for the specified subject information (name and token).

Parameters:

pSubjectInfo - (PSUBJECTINFO) input/output

Address of subject information. The structure is defined in Table 8.

Field	Length
Handle	DWORD
Instance	DWORD
Length of Name	DWORD
Pointer to Name	DWORD
Length of Token	DWORD
Pointer to Token	DWORD
Source	DWORD

Table 8. Address of Subject Information

All lengths are maximum 32 bytes (including any null terminating character).

Handle	Output field. Subject Handle created for specified Name/Token. The Subject Handle is a unique number (per system boot) assigned by SES.
Instance	Set to -1 to indicated that this handle is not an instance handle.
Length of name	Input field. Length of subject name. Cannot be zero.
Pointer to name	Input field. Pointer to subject name. Cannot be null.
Length of token	Input field. Length of subject token.
Pointer to token	Input field. Pointer to subject token.
Source	Output field. Source of authority will be set to the AuthorityID of caller (SLA or RLA). See SESQuerySubjectInfo() for AuthorityID values.

The C Language declaration of the structure is the following:

```
typedef struct _SUBJECTINFO
{
    HSUBJECT    Handle;           /* Subject Handle      */
    HSUBJECT    Instance;        /* Instance Handle     */
    PVOID       Name;            /* Subject Name        */
    ULONG       NameLen;         /* Subject Name Length */
    PVOID       Token;           /* Subject Token       */
    ULONG       TokenLen;        /* Subject Token Length */
    ULONG       Source;          /* Source of Authority */
} SUBJECTINFO, *PSUBJECTINFO;
```

Returns

Standard OS/2 API return codes.

0 NO_ERROR

SES API return codes.

35367 SES_INVALID_PARAMETER
35395 SES_PROTECTION_VIOLATION
35517 SES_INVALID_AUTHORITY

Remarks

This call requires SLA or RLA authority.

No check is made of the user information against user information already stored by security control services. A new handle is returned for each call. Security control services does not guarantee the uniqueness of the user information associated with a handle. For example, one specific name can be associated with more than one handle.

B.5 SESDeleteHandleNotify

APIRET APIENTRY SESDeleteHandleNotify(PHSUBJECT pSubjectHandle)

This functions allows a process/thread to register for notification of handle deletion.

Parameters:

pSubjectHandle - (PHSUBJECT) output

Pointer to the SES handle that has been deleted.

Returns

Standard OS/2 API return codes.

0 NO_ERROR

SES API return codes.

35517 SES_INVALID_AUTHORITY
35311 SES_GENERAL_FAILURE

Remarks

This call requires access control authority. It is held captive in the security enabling services device driver until a handle has been deleted.

Upon return, the access control authority can take whatever action is appropriate, for example remove the user's credentials (for example user ID, group ID(s), etc.) and associated subject handle from the cache of active users/handles on the system. The call must then be re-issued to continue receiving notifications. Any handle deletions made while the security control authority is processing the notification are queued up until the call is re-issued.

B.6 SESDeleteSubjectHandle

APIRET APIENTRY SESDeleteSubjectHandle(HSUBJECT SubjectHandle)

This function deletes or removes the specified subject handle from use.

Parameters:

SubjectHandle - (HSUBJECT) input

Value of the subject handle, see SESCreateSubjectHandle().

Returns

Standard OS/2 API return codes.

0 NO_ERROR

SES API return codes.

35286	SES_INVALID_HANDLE
35367	SES_INVALID_PARAMETER
35517	SES_INVALID_AUTHORITY

Remarks

This call requires system logon authority or remote logon authority. The caller must have the same authority ID as the process that created the handle (for example, can only delete the handle it created).

The handle is not actually deleted until all processes/threads using the handle have terminated.

B.7 SESKillProcess

APIRET SESKillProcess(PID idProcessID)

This function enables the hard kill of a process. A call to this API will kill the specified process without regard to the current state of the process. For example, a process may have output files open that will not be closed in an orderly manner and may become unusable if the process is killed via this API. If the process is executing in supervisor mode, it will die as it makes the transition from Ring-0 to Ring-3. If the process is currently executing in Ring 3, it will die immediately.

Parameter:

idProcessID(PID) - Input

The process ID of the process to be killed.

Returns

Standard OS/2 API return codes.

0	NO_ERROR
303	ERROR_INVALID_PROCID

Remarks

This call requires SLA or RLA authority.

B.8 SESlogIntegrityViol

```
APIRET SESlogIntegrityViol(ULONG Flag,
                           PCHAR pLogData)
```

This function writes the LogData string to the error log file (SES.LOG in the directory specified by the SESDBPATH environment variable) and optionally starts a shutdown event.

Note: The installable security subsystem should enforce an access control policy on the SES.LOG file for example untrusted processes should not be allowed to access it.

Parameters:

Flag (ULONG) - input

LOG_ONLY - only write the LogData to the error log
LOG_HALT - write the LogData to the error log and start a shutdown event

pLogData (PCHAR) - input

Null terminated string that will be recorded in the error log.

Returns

Standard OS/2 API return codes.

0 NO_ERROR

Remarks

No authority is required for this call.

B.9 SESQueryAuthorityID

```
APIRET APIENTRY SESQueryAuthorityID(PSZ pszAuthorityTag,
                                     PULONG pAuthorityID)
```

This function queries the authority ID corresponding to the specified Authority Tag.

Parameters:

pszAuthorityTag - (PSZ) input

Address of the authority tag. Length of the tag must be less than or equal to eight characters, without including any terminating null character.

pAuthorityID - (PULONG) output

Address of the authority ID associated with the specified Authority Tag. See SESQuerySubjectInfo() for possible returned authority ID.

Returns**Standard OS/2 API return codes.**

0 NO_ERROR

SES API return codes.

35395 SES_PROTECTION_VIOLATION
35524 SES_ID_NOT_FOUND
35525 SES_INVALID_TAG_LENGTH

Remarks

No authority is required for this call.

B.10 SESQueryContextStatus

```
APIRET APIENTRY SESQueryContextStatus(PID     pid,  
                                       PULONG  pContextStatus)
```

This function queries the context status of the specified process.

Parameters:**pid - (PID) input**

Process ID:

pid == 0 : Context Status in Effective Security Context of current
 PID/TID is returned
pid <> 0 : Context Status in Maximum Security Context of specified
 PID is returned

pContextStatus - (PULONG) output

Address of context status. The context status consists of the following data:

Authority ID	Authority ID is returned in the 0x??----- format with the value in ?? from 0 to 255 as follows: AuthorityID : Security Context Authority ----- 0 : none 1 : SES 2 : PSS 3 : reserved 4 : SLA 5-15 : reserved 16-255 : any other SCA
Status Flags	Status flags are returned in the 0x--?????? format where ?????? represents the returned flags. See SESSetContextStatus() for the values defined for status flags.

Returns

Standard OS/2 API return codes.

0 NO_ERROR

SES API return codes.

35395 SES_PROTECTION_VIOLATION
35518 SES_PROCESS_NONEXISTENT

Remarks

No authority is required for this call.

B.11 SESQueryProcessInfo

APIRET	SESQueryProcessInfo(ULONG	ulActionCode,
	HSUBJECT	CUH,
	PULONG	pu1ProcessCount,
	PVOID	pProcessBuf)

This function enables the caller to obtain information about currently running processes.

Parameters:

ulActionCode(ULONG) - INPUT

- `== 0: (QPI_QUERYSIZE)` Query the size of buffer required to retrieve all information on active processes at time of call.
- `== 1: (QPI_QUERYINFO)` Return the process info in the buffer provided (pProcessBuf)

CUH(HSUBJECT) - INPUT

- `== 0:` return process information for all running processes
- `<> 0:` if a non-zero CUH. is specified, the API will return process information for all running processes that are associated with the specified client user handle

pulProcessCount (PULONG) - INPUT/OUTPUT

Address of variable to receive/specify the processes count.

output: for Action (QPI_QUERYSIZE) this will return the number of processes found meeting the criteria specified by CUH. This value should be used to allocate the buffer pProcessBuf.

input: for Action (QPI_QUERYINFO) this should specify the number of processes which the buffer pProcessBuf can hold. (Should be the same value returned by QPI_QUERYSIZE)

pProcessBuf (PVOID) - INPUT

For Action QPI_QUERYSIZE: This should be NULL

For Action QPI_QUERYINFO: This is the address of the buffer to receive information. The buffer must be large enough to receive the information initially requested in the QPI_QUERYSIZE call.

The buffer will contain the following structure for each process:

```
typedef struct _PROCESSBUF
{
```

```

        PID    ProcessID;
        CHAR    ProcessName(cchmaxpath);
    } PROCESSBUF, *PPROCESSBUF;

```

Returns

Standard OS/2 API return codes.

0 NO_ERROR

SES API return codes.

35367 SES_INVALID_PARAMETER
 35391 SES_BUFFER_OVERFLOW
 35517 SES_INVALID_AUTHORITY
 35935 SES_PROTECTION_VIOLATION

Remarks

The purpose of the API is to provide the system logon authority with information necessary to enforce a policy at logoff, and shutdown time. For example, at logoff the system logon authority may want to enforce a policy where all processes running on behalf of the current logged on user are killed. At shutdown, the policy may include stopping all process explicitly.

We recommend that the installable security subsystem use this API during logoff and shutdown to obtain a list of processes the user is currently executing so that this information can be displayed to the user to confirm whether the user wants to logoff/shutdown. See the logon shell services system design section for specific details.

The API SESQuerySubjectInfo() can be used to determine the client user handle for the current logged on user, or the system logon authority can save this value from the Logon event.

Usage

The API must be called twice to retrieve the requested information. The purpose of the first call is to determine the buffer size. An action of QPI_QUERYSIZE should be specified on the first call. The value returned in ProcessCount should be used to allocate the proper buffer size for the second call. The second call should specify an action of QPI_QUERYINFO, and provide an address for pProcessBuf. The caller

must make sure that client user handle is the same value on both calls. This will not be checked by security enabling services. The information returned will be for the value of client user handle on the QPI_QUERYINFO call. If this parameter was not the same as specified on the QPI_QUERYSIZE call, you risk a buffer overflow.

To make sure that no new processes are started while the event is in progress, the system logon authority must call the SESControlProcessCreation() API before querying for process information. This is important not only to make sure that the information between the QUERYSIZE and QUERYINFO calls is consistent, but also to make sure that the user does not start any new processes the system logon authority is not aware of during a logoff, or shutdown event.

B.12 SESQuerySecurityContext

```
APIRET APIENTRY SESQuerySecurityContext(PID          pid,
                                         PSECURITYCONTEXT pSecurityContext)
```

This function queries the security context of the specified process.

Parameters:

pid - (PID) input

Process ID:

pid == 0 : Effective Security Context of current PID/TID is returned
 pid <> 0 : Maximum Security Context of specified PID is returned

pSecurityContext - (PSECURITYCONTEXT) output

Address of the security context. See SESSetSecurityContext() for definition of security context. See SESQueryContextStatus() for definition of context status field of security context. See SESSetContextStatus() for definition of status flags in context status.

Returns

Standard OS/2 API return codes.

0 NO_ERROR

SES API return codes.

35395	SES_PROTECTION_VIOLATION
35518	SES_PROCESS_NONEXISTENT

Remarks

No authority is required for this call.

B.13 SESQuerySubjectHandle

```
APIRET APIENTRY SESQuerySubjectHandle(PID      pid,
                                       ULONG     TargetSubject,
                                       PHSUBJECT pSubjectHandle)
```

This function queries the subject handle for the target subject of the specified process.

Parameters:

pid - (PID) input

Process ID:

pid == 0: Specified handle in Effective Security Context of current
PID/TID is returned
pid <> 0: Specified handle in Maximum Security Context of specified
PID is returned

TargetSubject - (ULONG) input

0	EFFECTIVE_PROCESS
1	EFFECTIVE_USER
2	EFFECTIVE_GROUP
4	CLIENT_PROCESS
5	CLIENT_USER
6	CLIENT_GROUP
8	AGENT_PROCESS
9	AGENT_USER
10	AGENT_GROUP

pSubjectHandle - (PHSUBJECT) output

Address of the subject handle. See SESCreateSubjectHandle().

Returns

Standard OS/2 API return codes.

0 NO_ERROR

SES API return codes.

35367 SES_INVALID_PARAMETER
35395 SES_PROTECTION_VIOLATION
35518 SES_PROCESS_NONEXISTENT

Remarks

No authority is required for this call.

A subject handle value of 0 indicates a superuser or system process. It should imply full access to all objects.

A subject handle value of -1 indicates an unauthenticated user. It should imply only public/guest access to objects.

B.14 SESQuerySubjectHandleInfo

APIRET APIENTRY SESQuerySubjectHandleInfo(HSUBJECT SubjectHandle,
PSUBJECTINFO pSubjectInfo)

This function queries the subject info for the specified subject handle (original or instance).

Parameters:

SubjectHandle - (HSUBJECT) input

Value of the subject handle. See SESCreateSubjectHandle().

pSubjectInfo - (PSUBJECTINFO) output

Address of the subject info. See SESQuerySubjectInfo().

Returns

Standard OS/2 API return codes.

0 NO_ERROR

SES API return codes.

35286	SES_INVALID_HANDLE
35367	SES_INVALID_PARAMETER
35391	SES_BUFFER_OVERFLOW
35395	SES_PROTECTION_VIOLATION

Remarks

No authority is required for this call. However, only a caller with user identification authority or client logon authority will receive the token in the subject information.

B.15 SESQuerySubjectInfo

```
APIRET APIENTRY SESQuerySubjectInfo(PID          pid,
                                     ULONG         TargetSubject,
                                     PSUBJECTINFO pSubjectInfo)
```

This function queries the subject info for the target subject of the specified process.

Parameters:

pid - (PID) input

Process ID. The target security context (maximum or effective) is specified by the PID.

pid == -1: Maximum Security Context of current logged on user is returned
 pid == 0: Effective Security Context of current PID/TID is returned
 pid <> -1,0: Maximum Security Context of specified PID is returned

TargetSubject - (ULONG) input

0	EFFECTIVE_PROCESS
1	EFFECTIVE_USER
2	EFFECTIVE_GROUP
4	CLIENT_PROCESS
5	CLIENT_USER
6	CLIENT_GROUP
8	AGENT_PROCESS
9	AGENT_USER
10	AGENT_GROUP

For a query of the current logged on user (PID = -1), the CLIENT_USER information is returned (TargetSubject is ignored). SES_NO_CURRENT_USER is returned if no user is currently logged on.

pSubjectInfo - (PSUBJECTINFO) output

Address of SubjectInfo structure. See SESSubjectHandle() for the format of SubjectInfo and for the C declarations of PSUBJECTINFO and SUBJECTINFO. All lengths are maximum 32 bytes (including any NULL terminating character). The name and token fields in the SubjectInfo structure need not be null-terminated. The corresponding length fields must be used to determine the length.

Handle	Output field. Subject handle for specified TargetSubject.
Instance	<p>Output field. It returns a -1 to indicate the returned handle is an original subject handle or returns another value to indicate the returned handle is an instance handle.</p> <p>If the returned value is not -1, the caller may call SESQuerySubjectHandleInfo to track the chain of instance handles back to the original subject handle (but all handles in this chain will have the same name, token, and source).</p>
Length of name	Input/Output field. On input this should be set to the length of the name buffer (recommend using a maximum buffer size of 32). On output this returns the actual length of the name.
Pointer to name	Input/Output field. On input this should contain a valid pointer to a buffer of the size specified by length. On output the buffer contains the subject name.
Length of token	Input/Output field. On input this should be set to the length of the tToken buffer (recommend using a maximum buffer size of 32). On output this returns the actual length of the token (if caller has the authority to access token information).
Pointer to token	Input/Output field. On input this should contain a valid pointer to a buffer of the size specified by length. On output the buffer contains the token (if caller has the authority to access token information).
Source	<p>Output field. Source of authority for creation of a Subject handle or authentication of the user associated with a subject handle.</p> <p>The field may contain the authority ID of the SLA/RLA that created the handle or the UIA that authenticated the user</p>

associated with the handle, or the field may contain an authentication rule number for the the authentication of the user associated with the handle.

AuthorityID : Security Context Authority

0 : none
 4 : SLA
 16-255 : any other SCA
 256-MAX : authentication rule number

Returns

Standard OS/2 API return codes.

0 NO_ERROR

SES API return codes.

35367 SES_INVALID_PARAMETER
 35391 SES_BUFFER_OVERFLOW
 35395 SES_PROTECTION_VIOLATION
 35518 SES_PROCESS_NONEXISTENT
 35526 SES_NO_CURRENT_USER

Remarks

No authority is required for this call. However, only a caller with user identification authority or client logon authority will receive the token in the subject information.

B.16 SESReleaseSubjectHandle

APIRET APIENTRY SESReleaseSubjectHandle(ULONG TargetSubject,
 HSUBJECT SubjectHandle)

This function releases the subject handle specified by the target subject (in the effective security context of the calling thread of an server process authority) from the SPA's reserved set of handles that it is allowed to set. When a handle is released by the server process authority, the handle can then be deleted if there are no other processes/threads referencing the handle.

Parameters:

TargetSubject - (ULONG) input

0	EFFECTIVE_PROCESS
1	EFFECTIVE_USER
2	EFFECTIVE_GROUP

SubjectHandle - (HSUBJECT) input

Value of the subject handle to be deleted from the SPA's list of allowed handles.

Returns

Standard OS/2 API return codes.

0	NO_ERROR
---	----------

SES API return codes.

35286	SES_INVALID_HANDLE
35367	SES_INVALID_PARAMETER
35517	SES_INVALID_AUTHORITY

Remarks

This call requires server process authority.

B.17 SESReserveSubjectHandle

APIRET APIENTRY SESReserveSubjectHandle(ULONG TargetSubject)

This function reserves the subject handle specified by the target subject (in the effective security context of the calling thread of an server process authority) in the list of handles that the server process authority is allowed to set. While a handle is reserved for use by an server process authority, it cannot be deleted even when no other processes/threads are referencing the handle.

Parameters:

TargetSubject - (ULONG) input

0	EFFECTIVE_PROCESS
1	EFFECTIVE_USER
2	EFFECTIVE_GROUP
3	EFFECTIVE_USER_AND_GROUP

Returns

Standard OS/2 API return codes.

0	NO_ERROR
---	----------

SES API return codes.

35367	SES_INVALID_PARAMETER
35517	SES_INVALID_AUTHORITY

Remarks

This call requires SPA authority.

B.18 SESResetThreadContext

APIRET APIENTRY SESResetThreadContext(ULONG TargetContext)

This function resets the calling thread's effective security context according to the specified TargetContext.

Parameters:

TargetContext - (ULONG) input

PROCESS_CONTEXT

This function resets the thread's effective security context to be the same as the effective security context maintained by thread 1 of the process.

THREAD_CONTEXT

This function creates a new (private) effective security context for the thread by copying the maximum security context.

See `SESSetSecurityContext()` for the format and C declarations of the security context structure.

Returns

Standard OS/2 API return codes.

0 NO_ERROR

SES API return codes.

35367 SES_INVALID_PARAMETER
35395 SES_PROTECTION_VIOLATION

Remarks

No authority is required for this call.

Once a thread has created a private effective context, it can make changes without affecting other threads of the process.

B.19 SESSendSecurityContext

```
APIRET APIENTRY SESSendSecurityContext(PULONG pMessageLength,  
                                       PVOID pMessage,  
                                       ULONG Timeout,  
                                       ULONG AuthorityID)
```

This function enables a process/thread to send its effective security context (with a message) to the security control authority specified by the authority ID. The sending process/thread is blocked until either the message has been delivered to the security control authority and it has responded, or the specified timeout value has been exceeded.

Parameters:

MessageLength - (PULONG) input/output

Length of message text without including any terminating null character. The length is a maximum of 512 bytes. If message length is 0, no message is specified and pointer to message is NULL.

pMessage - (PVOID) input/output

Address of message text. Message can be binary.

Timeout - (ULONG) input

Number of milliseconds that the sending process/thread wants to wait for the target SCA to accept the message. If the Timeout is exceeded before the target SCA receives the message, the message is deleted and the sending process/thread is unblocked.

Authority ID - (ULONG) input

Authority ID of the target SCA.

Returns**Standard OS/2 API return codes.**

0 NO_ERROR

SES API return codes.

35521 SES_INVALID_EVENT_DATA
 35527 SES_EVENT_FAILURE
 35533 SES_EVENT_INVALID
 35534 SES_EVENT_TIMED_OUT

Remarks

No authority is required for this call. This API uses logon shell services, so the system must be in an initialized state or logon shell services will return SES_INVALID_EVENT. The personal shell services process is responsible for initializing logon shell services by starting the event SES_EVENT_INIT. The process targeted to receive the message must be registered with logon shell services, and waiting on the event SES_EVENT_SEND_SECURITY_CONTEXT.

B.20 SESSetContextStatus

APIRET APIENTRY SESSetContextStatus(ULONG ContextStatus)

This function sets the context status in the effective security context of the calling thread.

Parameters:**ContextStatus - (ULONG) input**

The status flags in ContextStatus are defined as follows:

```

/* SES authority flag definitions: */

#define APA_FLAG 0x00000001 /* Agent Process Authority */
#define SPA_FLAG 0x00000002 /* Server Process Authority */
#define RLA_FLAG 0x00000004 /* Remote Logon Authority */
#define SLA_FLAG 0x00000008 /* System Logon Authority */
#define ACA_FLAG 0x00000100 /* Access Control Authority */
#define CLA_FLAG 0x00000200 /* Client Logon Authority */
#define UIA_FLAG 0x00000400 /* User Identification Authority */
#define PSS_FLAG 0x00004000 /* Protected Shell Services Authority */
#define SES_FLAG 0x00008000 /* Security Enabling Services Authority */

/* SES state flag definitions: */

#define EUF_FLAG 0x00010000 /* Effective User Flag indicates state */
/* of Effective User Handle: */
/* 0: Client User Handle */
/* 1: Agent User Handle */
#define EGF_FLAG 0x00020000 /* Effective Group Flag indicates state */
/* of Effective Group Handle: */
/* 0: Client Group Handle */
/* 1: Agent Group Handle */
#define EPF_FLAG 0x00040000 /* Effective Process Flag indicates state */
/* of Effective Process Handle: */
/* 0: Client Process Handle */
/* 1: Agent Process Handle */
#define PAF_FLAG 0x00080000 /* Propagate Authority Flag */
#define LUF_FLAG 0x00100000 /* Local (PM) User Flag */

```

Changes to status flags are dependent upon the effective authority of the caller.

Returns

Standard OS/2 API return codes.

0 NO_ERROR

SES API return codes.

35367 SES_INVALID_PARAMETER

Remarks

No authority is required for the call itself; however, the values which can be set are restricted by the authority flags in the maximum security context of the calling process.

The `SESSetContextStatus()` API can only be used to modify the effective security context of a process/thread. All changes made to the effective security context of the calling thread are granted based on the contents of the maximum security context of the calling process. If the status/authority flag being requested is set in the maximum security context, then the request is granted by security enabling services.

Note: The default security context model is a process model. All threads share the same security context, so if one thread makes a change (using the `SESSetContextStatus()` API) then all others will see the same change.

If a thread of a process wants to be isolated from changes made by other threads, or wants to make a change without affecting other threads, it can create a private copy of its effective security context with `SESResetThreadContext()` before making any changes with `SESSetContextStatus()`.

B.21 SESSetSecurityContext

APIRET APIENTRY SESSetSecurityContext(PSECURITYCONTEXT pSecurityContext)

This function sets the security context in the effective security context of the calling thread.

Parameters:

pSecurityContext - (PSECURITYCONTEXT) input

Address of the security context structure.

The effective security context for a thread has the following information:

Field	Length
Client User Handle	DWORD
Agent User Handle	DWORD
Client Group Handle	DWORD
Agent Group Handle	DWORD

Field	Length
Client Process Handle	DWORD
Agent Process Handle	DWORD
Security Context Status	DWORD

Table 9. Effective Security Context Structure

The C Language declaration of the structure is the following:

```
typedef ULONG HSUBJECT;      /* HSUBJECT is a unique 32-bit handle */

typedef HSUBJECT *PHSUBJECT; /* PHSUBJECT is a pointer to HSUBJECT */

typedef struct _SECURITYCONTEXT
{
    HSUBJECT    CUH;          /* Client User Handle */
    HSUBJECT    AUH;          /* Agent User Handle */
    HSUBJECT    CGH;          /* Client Group Handle */
    HSUBJECT    AGH;          /* Agent Group Handle */
    HSUBJECT    CPH;          /* Client Process Handle */
    HSUBJECT    APH;          /* Agent Process Handle */
    ULONG       ContextStatus; /* Security Context Status */
} SECURITYCONTEXT;

typedef SECURITYCONTEXT *PSECURITYCONTEXT;
```

Changes to the security context are dependent upon the effective authority of the caller.

Returns

Standard OS/2 API return codes.

```
0          NO_ERROR
```

SES API return codes.

```
35286     SES_INVALID_HANDLE
35367     SES_INVALID_PARAMETER
35395     SES_PROTECTION_VIOLATION
35517     SES_INVALID_AUTHORITY
```

Remarks

Restrictions on this API are as for `SESSetSubjectHandle` for handle changes, and `SESSetContextStatus` for context status changes.

B.22 `SESSetSubjectHandle`

```
APIRET APIENTRY SESSetSubjectHandle(ULONG    TargetSubject,
                                     HSUBJECT SubjectHandle)
```

This function sets the subject handle specified by the `TargetSubject` in the effective security context of the calling thread.

Parameters:

`TargetSubject` - (ULONG) input

- 0 EFFECTIVE_PROCESS
- 1 EFFECTIVE_USER
- 2 EFFECTIVE_GROUP
- 3 EFFECTIVE_USER_AND_GROUP
- 4 CLIENT_PROCESS
- 5 CLIENT_USER
- 6 CLIENT_GROUP
- 7 CLIENT_USER_AND_GROUP
- 8 AGENT_PROCESS
- 9 AGENT_USER
- 10 AGENT_GROUP
- 11 AGENT_USER_AND_GROUP
- 12 BOTH_CLIENT_AGENT_PROCESS
- 13 BOTH_CLIENT_AGENT_USER
- 14 BOTH_CLIENT_AGENT_GROUP
- 15 BOTH_CLIENT_AGENT_USER_AND_GROUP

`SubjectHandle` - (HSUBJECT) input

Value of the subject handle, see `SESCreateSubjectHandle()`.

Returns

Standard OS/2 API return codes.

- 0 NO_ERROR

SES API return codes.

35286	SES_INVALID_HANDLE
35367	SES_INVALID_PARAMETER
35517	SES_INVALID_AUTHORITY

Remarks

The caller must have one of the following authorities, with the restrictions listed:

- | | |
|------------|---|
| SLA | A system logon authority can set any valid handle, including handles 0 and -1. |
| RLA | A remote logon authority can only set handles that it creates and handle -1. |
| SPA | A server process authority can only set the client user/group/process handles equal to the handles in its list of allowed handles or in its maximum security context. |

For a caller with multiple authorities, the call succeeds if it would be possible under any one of the authorities. For example, a caller with server process authority and remote logon authority may set any handle it creates or that is in its list of allowed handles.

B.23 SESSetSubjectInfo

```
APIRET APIENTRY SESSetSubjectInfo(ULONG TargetSubject,
                                   PSUBJECTINFO pSubjectInfo)
```

This function creates/sets/deletes the SubjectHandle specified by TargetSubject in the effective security context of the calling thread.

Parameters:

TargetSubject - (ULONG) input

4	CLIENT_PROCESS
5	CLIENT_USER
6	CLIENT_GROUP
7	CLIENT_USER_AND_GROUP
8	AGENT_PROCESS
9	AGENT_USER
10	AGENT_GROUP
11	AGENT_USER_AND_GROUP
12	BOTH_CLIENT_AGENT_PROCESS
13	BOTH_CLIENT_AGENT_USER

```

14  BOTH_CLIENT_AGENT_GROUP
15  BOTH_CLIENT_AGENT_USER_AND_GROUP

```

pSubjectInfo - (PSUBJECTINFO) input/output

Address of SubjectInfo structure. See SESSubjectInfo() for the format of SubjectInfo and for the C declarations of PSUBJECTINFO and SUBJECTINFO. All lengths are maximum 32 bytes (including any NULL terminating character).

Handle	Output field. Subject Handle created for specified Name/Token.
Instance	Set to -1 to indicated that this handle is not an instance handle.
Length of name	Input field. Length of subject name. Cannot be zero.
Pointer to name	Input field. Pointer to subject name. Cannot be null.
Length of token	Input field. Length of subject token.
Pointer to token	Input field. Pointer to subject token.
Source	Output field. Source of authority will be set to the authority ID of caller (SLA or RLA). See SESQuerySubjectInfo() for authority ID values.

Returns

Standard OS/2 API return codes.

```

0      NO_ERROR

```

SES API return codes.

```

35367  SES_INVALID_PARAMETER
35395  SES_PROTECTION_VIOLATION
35517  SES_INVALID_AUTHORITY

```

Remarks

This call requires system logon authority or remote logon authority. The handle is not actually deleted until it is no longer referenced by any process/thread.

Appendix C. System Logon Driver API Details

This section details the functions contained in the system logon driver (SLD) API.

C.1 SLDInit

APIRET APIENTRY SLDInit(VOID)

This function allows the system logon driver to do whatever initialization is necessary. The default system logon driver provided with security enabling services uses this opportunity to cache the list of UIAs specified in SECURE.SYS.

Parameters:

None.

Returns

NO_ERROR

Remarks

This call must be issued before any call to the SLDQueryUIA API.

C.2 SLDQueryUIA

APIRET APIENTRY SLDQueryUIA(ULONG ulEventVector,
 ULONG ulEventID,
 PSEVENTDATA pEventData,
 PAUTH pAuthorityID,
 PULONG pulEventStatus)

This function retrieves the authority ID of the next user identification authority that should be invoked to identify/authenticate a user for the event specified by EventVector and EventID (for logon, unlock, etc.). The authority ID of the user identification authority and event status is returned to the security enabling services daemon and a return code is set.

Parameters

ulEventVector(ULONG) - input

A 32-bit vector with each bit representing a particular system event. This field is set by the security enabling services daemon. The following events are processed by the system logon driver:

- SES_EVENT_LOGON
- SES_EVENT_UNLOCK
- SES_EVENT_CHANGE_PASSWORD
- SES_EVENT_CREATE_PROFILE
- SES_EVENT_DELETE_PROFILE
- SES_EVENT_IA

ulEventID(ULONG) - input

This field is maintained by the security enabling services daemon. It identifies the process that is requesting the user identification authority ID.

pEventData(PSESEVENTDATA) - input/output

A pointer to an event data structure. The address of the event data structure is initially set by the security enabling services daemon. The fields of the event data structure can be set by the security enabling services daemon and modified by the system logon driver. This pointer can address one of the following event data structures:

- SESLOGON
- SESUNLOCK
- SESCCHANGEPASSWORD
- SESCREATEPROFILE
- SESEDELETEPROFILE
- SEZIA

pAuthorityID(PAUTH) - output

This is a pointer to the user identification authority ID of the user identification authority to be queried by SES (authority ID range 4 to 255).

pulEventStatus(PULONG) - input/output

This is a pointer to the event status field. This field is set by a user identification authority and passed back to the security enabling services daemon, which in turn passes it to the system logon driver. Because this API returns the ID of the user identification authority that is run next, the authentication status is not available on the first call to the system logon driver. Table 10 on page 273 and Table 11 on page 273 show the input and output event status that is possible on a call to the system logon driver.

Status on Input	Logon	Unlock	Chng Psswr	Profile	I&A
SES_STATUS_USER_AUTHENTICATED	N	N	N	N	N
SES_STATUS_USER_UNAUTHENTICATED	N	N	N	N	N
SES_STATUS_NOT_APPLICABLE	N	N	N	N	N
SES_STATUS_NOT_AVAILABLE	N	N	N	N	N
SES_STATUS_ID_ONLY	Y	N	N	N	N
SES_STATUS_GUEST_USER	Y	Y	N	N	N

Table 10. SES Status on Input

Status on Output	Logon	Unlock	Chng Pass-word	Profile	I&A
SES_STATUS_USER_AUTHENTICATED	Y	Y	N	N	Y
SES_STATUS_USER_UNAUTHENTICATED	Y	Y	N	N	Y
SES_STATUS_NOT_APPLICABLE	N	N	N	N	N
SES_STATUS_NOT_AVAILABLE	N	N	N	N	N
SES_STATUS_ID_ONLY	Y	N	N	N	N
SES_STATUS_GUEST_USER	Y	Y	N	N	N

Table 11. SES Status on Output

On subsequent calls to SLDQueryUIA the rest of the UIAs, if any, are returned. In this case, on entry to the system logon driver, the event status of the previous user identification authority is received from the security enabling services daemon.

Returns

NO_ERROR
 ERROR_NO_MORE_ITEMS
 ERROR_INVALID_EVENT

Remarks

When the last user identification authority is retrieved a subsequent SLDQueryUIA() call will result in a return code of ERROR_NO_MORE_ITEMS.

Appendix D. Client Logon Driver API Details

This section details the functions contained in the client logon driver (CLD) API.

D.1 CLDInit

APIRET APIENTRY CLDInit(VOID)

This function allows the client logon driver to do whatever initialization is necessary. The default client logon driver provided with security enabling services uses this opportunity to cache the list of CLAs specified in SECURE.SYS.

Parameters:

None.

Returns

NO_ERROR

Remarks

This call must be issued before any call to the CLDQueryCLA API.

D.2 CLDQueryCLA

APIRET APIENTRY CLDQueryCLA(ULONG	ulEventVector,
ULONG	ulEventID,
PSEVENTDATA	pEventData,
PAUTH	pAuthorityID,
PULONG	pulEventStatus)

This function retrieves the authority ID of the next client logon authority that should be invoked for single signon support for the event specified by EventVector and EventID (for logon, unlock, etc.). The authority ID of the client logon authority and event status is returned to the security enabling services daemon and a return code is set.

Parameters:

ulEventVector(ULONG) - Input

A 32-bit vector with each bit representing a particular system event. The following events are processed by the client logon driver:

- SES_EVENT_LOGON
- SES_EVENT_UNLOCK
- SES_EVENT_LOGOFF
- SES_EVENT_LOCK
- SES_EVENT_SHUTDOWN
- SES_EVENT_CHANGE_PASSWORD
- SES_EVENT_CREATE_PROFILE
- SES_EVENT_DELETE_PROFILE

ulEventID(ULONG) - Output

This field is maintained by the security enabling services daemon. It identifies the process that is requesting the client logon authority ID.

pEventData(PSESEVENTDATA) - Input/output

A pointer to an event data structure. This pointer can address one of the following event data structures:

- SESLOGON
- SESUNLOCK
- SESLOGOFF
- SESLOCK
- SESSHUTDOWN
- SESCCHANGEPASSWORD
- SESCREATEPROFILE
- SESDELETEPROFILE

pAuthorityID(PAUTH) - Output

This is a pointer to the client logon authority ID of the client logon authority.

pulEventStatus(PULONG) - Input/output

This is a pointer to the event status field. This field can contain one of the following values:

- NULL
- SES_STATUS_AUTOGUEST
- SES_STATUS_GUEST_USER

Returns

- NO_ERROR
- ERROR_NO_MORE_ITEMS
- ERROR_INVALID_EVENT

Remarks

When the last client logon authority is retrieved a subsequent CLDQueryCLA() call will result in a return code of ERROR_NO_MORE_ITEMS.

Appendix E. Password Validation Driver API Details

This section details the functions contained in the password validation driver (PVD) API.

E.1 PVDValidatePassword

APIRET APIENTRY PVDValidatePassword(PSECHANGEPASSWORD pChangePassword)

The security enabling services daemon calls this function whenever a password must be validated. The password validation driver provided by an independent software vendor or customer can apply any password policy to the password and then pass back a return code indicating that the password is valid or invalid. The default OS/2 password validation driver does no password validation at all it simply returns to the security enabling services daemon with a return code of NO_ERROR.

Parameters:

pChangePassword(PSECHANGEPASSWORD) - Input

This field contains the address of the password information structure.

```
typedef struct _SECHANGEPASSWORD
{
    ULONG    ChangerNameLen;
    CHAR     ChangerName[MAX_USER_NAME];
    ULONG    ChangerPasswordLen;
    CHAR     ChangerPassword[MAX_TOKEN];
    ULONG    UserNameLen;
    CHAR     UserName[MAX_USER_NAME];
    ULONG    UserPasswordLen;
    CHAR     UserPassword[MAX_TOKEN];
    ULONG    SystemLen;
    CHAR     System[MAX_SYSTEM];
} SECHANGEPASSWORD, *PSECHANGEPASSWORD;
```

Returns

NO_ERROR
ERROR_INVALID_PASSWORD

Remarks
None.

Appendix F. Logon Shell Services API Details

This section details the functions contained in the logon shell services (LSS) API.

F.1 SESSControlKBDMonitors

APIRET SESSControlKBDMonitors(ULONG ulActionCode,
PULONG pulStatus)

This function allows the system logon authority to inhibit the routing of keyboard input to keyboard monitor programs.

Parameters:

ulActionCode (ULONG) - Input

ulActionCode	Definition
-----	-----
0 (CKM_OFF)	Turn monitors off
1 (CKM_ON)	Turn monitors on
2 (CKM_QUERY)	Query current status (on/off)

pulStatus (PULONG) - Output

Address of variable to receive current monitor state.

pulStatus	Definition
-----	-----
0 (off)	monitors are not receiving keystrokes
1 (on)	monitors are receiving keystrokes

Returns

Standard OS/2 API return codes.

0 NO_ERROR

SES API return codes.

35367 SES_INVALID_PARAMETER
35517 SES_INVALID_AUTHORITY
35535 SES_KBD_MOUSE_ERROR

Remarks

This call requires system logon authority.

F.2 SESInactivityNotify

APIRET APIENTRY SESInactivityNotify(ULONG timeout)

Used to notify a process of keyboard and mouse inactivity.

Parameters:**Timeout (ULONG) - Input**

Timeout in minutes.

Returns**Standard OS/2 API return codes.**

0	NO_ERROR
110	ERROR_OPEN_FAILED

SES API return codes.

35517	SES_INVALID_AUTHORITY
35535	SES_KBD_MOUSE_ERROR

Remarks

This call requires system logon authority. The calling thread will be blocked in Ring 0. The address provided at initialization will be checked at 30 second intervals. The smallest timeout period available to the user will be 1 minute and the largest 99 minutes. If there is no activity detected within the timeout period, this thread will be unblocked. If there is any activity detected the timeout value will be reset to the original value. A timeout of value zero will mean a timeout of infinity and therefore a lockup will never occur.

F.3 SESRegisterDaemon

```
APIRET APIENTRY SESRegisterDaemon(PULONG pDaemonNumber,  
                                   ULONG EventList)
```

The SESRegisterDaemon API is used to enable an security control authority to wait on logon shell services events.

Parameters:

pDaemonNumber - (PULONG) output

The DaemonNumber must be passed as an input parameter to any subsequent WaitEvent or ReturnStatus calls.

EventList - (ULONG) input

The EventList identifies the events the security control authority will service. The EventList can be any combination of logon shell services events for which the security control authority has proper authority.

```
SES_EVENT_LOGON  
SES_EVENT_LOGOFF  
SES_EVENT_SHUTDOWN  
SES_EVENT_LOCK  
SES_EVENT_UNLOCK  
SES_EVENT_CHANGE_PASSWORD  
SES_EVENT_CREATE_PROFILE  
SES_EVENT_DELETE_PROFILE  
SES_EVENT_IA  
SES_EVENT_SEND_SECURITY_CONTEXT  
SES_EVENT_PROCESS_CREATION
```

Returns

Standard OS/2 API return codes.

```
0      NO_ERROR
```

SES API return codes.

```
35517  SES_INVALID_AUTHORITY  
35533  SES_EVENT_INVALID
```

Remarks

A security control authority must obtain a DaemonNumber via the SESRegisterDaemon before waiting on any LSS event. The EventList parameter identifies the events the security control authority will service. The DaemonNumber is passed as input to the WaitEvent, ReturnEventStatus, and ReturnWaitEvent calls. There must be a unique DaemonNumber for each instance of a WaitEvent. If multiple threads will be concurrently waiting on events, each thread should request a DaemonNumber via this API.

The system logon authority may specify a security context to be used as the logoff state security context in this API by first creating/setting the appropriate security context in its own effective security context before calling this API. If the subject handles in the SLA's effective security context are zero when this API is called, then the logoff state security context will be the default:

```
client user handle      = -1 (unauthenticated user)
agent user handle       = -1
client group handle     = -1
agent group handle      = -1
client process handle   = -1
agent process handle    = -1
security context status = 0 (all state/authority flags=0)
```

If the subject handles in the SLA's effective security context are non-zero when this API is called, security enabling services will assign the subject handles in the SLA's effective security context to the logoff state maximum and effective security contexts. The system logon authority may also specify the agent process authority flag in the logoff state maximum and effective security contexts and EGF/EPF/EUF in the logoff state effective security context.

Specified by SLA	==>	Logoff State Security Context

CUH=U1	==>	Maximum CUH=U1, Effective CUH=U1
AUH=U2	==>	Maximum AUH=U2, Effective AUH=U2
CGH=G1	==>	Maximum CGH=G1, Effective CGH=G1
AGH=G2	==>	Maximum AGH=G2, Effective AGH=G2
CPH=P1	==>	Maximum CPH=P1, Effective CPH=P1
APH=P2	==>	Maximum APH=P2, Effective APH=P2

APA=X	==>	Maximum APA=X, Effective APA=X

APA=0:		
EGF=G	==>	Maximum EGF=0, Effective EGF=0
EPF=P	==>	Maximum EPF=0, Effective EPF=0

EUF=U	==>	Maximum EUF=0, Effective EUF=0

APA=1:		
EGF=G	==>	Maximum EGF=1, Effective EGF=G
EPF=P	==>	Maximum EPF=1, Effective EPF=P
EUF=U	==>	Maximum EUF=1, Effective EUF=U

Upon return from this API, the system logon authority should not delete the handles that it has assigned to the logoff state security context. Also, the system logon authority will probably need to reset the handles in its own security context.

F.4 SESReturnEventStatus

APIRET SESReturnEventStatus(PSESEVENT pEventInfo)

The SESReturnEventStatus API is used to return the status of an event.

Parameters:

pEventInfo - (PSESEVENT) Input/output

```
typedef struct _SESEVENT
{
    ULONG          DaemonNumber;
    ULONG          ReqPID;
    ULONG          ReqTID;
    SESSecurityContext SCH;
    ULONG          EventID;
    ULONG          Event;
    ULONG          EventStatus;
    SESEVENTDATA  EventData;
} SESEVENT, *PSESEVENT;
```

DaemonNumber - (ULONG) input

The DaemonID specified here must be a valid ID returned from a previous call to SESRegisterDaemon(). For SESReturnEventStatus() this should be the same DaemonID specified in the corresponding SESWaitEvent() call.

ReqPID - (ULONG) N/A

This parameter is not applicable to a return status call.

ReqTID - (ULONG) N/A

This parameter is not applicable to a return status call.

SCH - (SESSecurityContext) input

For events where it is appropriate, this structure contains the current security context of the process/thread that initiated the event.

For the logon event, the client user handle will be the handle reserved by security enabling services during identification and authorization. The system logon authority can use this parameter to return the client group/process handles for the local system logon user. All other fields are ignored!

For the process creation event, this structure is not used to return the handles/flags from the system logon authority. Fields are defined in the event-specific data for process creation to allow the system logon authority to return handles/flags for process creation.

For send security context and other events that a client logon authority can receive, a client logon authority may assign instance handles for the client process in this structure.

EventID - (ULONG) input

EventID is a unique ID assigned by security enabling services to track the instance of the event. The EventID allows security enabling services to associate a ReturnEventStatus with a specific instance of an event type.

Event - N/A**EventStatus - (ULONG) input**

The EventStatus parameter is used to pass the results of an event to logon shell services and/or other SCAs. The following table shows the possible EventStatus values that can be returned for the listed events. Events not listed do not require any EventStatus for a ReturnEventStatus.

Event State (SCA)	Event Status
SES_EVENT_LOGON_UIA (UIA)	SES_STATUS_USER_AUTHENTICATED SES_STATUS_USER_UNAUTHENTICATED SES_STATUS_CANCEL SES_STATUS_NOT_APPLICABLE SES_STATUS_NOT_AVAILABLE SES_STATUS_ID_ONLY SES_STATUS_GUEST_USER
SES_EVENT_LOGON_SLA (SLA)	SES_STATUS_SYSTEM_LOGON SES_STATUS_NO_ERROR SES_STATUS_GUEST_USER SES_STATUS_EVENT_FAILURE
SES_EVENT_LOGOFF_SLA (SLA)	SES_STATUS_NO_ERROR SES_STATUS_EVENT_FAILURE
SES_EVENT_LOGOFF_QUERY (SLA)	SES_STATUS_NO_ERROR SES_STATUS_EVENT_FAILURE
SES_EVENT_SHUTDOWN_SLA (SLA)	SES_STATUS_NO_ERROR SES_STATUS_EVENT_FAILURE
SES_EVENT_SHUTDOWN_QUERY (SLA)	SES_STATUS_NO_ERROR SES_STATUS_EVENT_FAILURE
SES_EVENT_LOCK_SLA (SLA)	SES_STATUS_NO_ERROR SES_STATUS_EVENT_FAILURE
SES_EVENT_LOCK_QUERY (SLA)	SES_STATUS_NO_ERROR SES_STATUS_EVENT_FAILURE
SES_EVENT_UNLOCK_UIA (UIA)	SES_STATUS_USER_AUTHENTICATED SES_STATUS_USER_UNAUTHENTICATED SES_STATUS_CANCEL SES_STATUS_NOT_APPLICABLE SES_STATUS_NOT_AVAILABLE SES_STATUS_GUEST_USER
SES_EVENT_UNLOCK_SLA (SLA)	SES_STATUS_NO_ERROR SES_STATUS_EVENT_FAILURE
SES_EVENT_IA (UIA)	SES_STATUS_USER_AUTHENTICATED SES_STATUS_USER_UNAUTHENTICATED SES_STATUS_NOT_APPLICABLE SES_STATUS_NOT_AVAILABLE SES_STATUS_ID_ONLY
SES_EVENT_PROCESS_CREATION (SLA)	SES_STATUS_NO_ERROR SES_STATUS_EVENT_FAILURE

EventData - (SESEVENTDATA) input/output

Event specific data. This structure is a union of event specific structures defined for each event.

Returns

Standard OS/2 API return codes.

0 NO_ERROR

SES API return codes.

35528 SES_INVALID_EVENTNUM
35529 SES_INVALID_DMONNUM
35535 SES_SCA_NOT_REGISTERED (only applicable for SES Daemon)

Remarks

The SESReturnEventStatus call is used to return the results of the event processing to logon shell services and/or subsequent SCAs. For example, if a user identification authority determined that a user entered an invalid token, the user identification authority would set the EventStatus parameter to indicate the user was unauthenticated.

F.5 SESReturnWaitEvent

APIRET APIENTRY SESReturnWaitEvent(PSESEVENT pEventInfo,
ULONG Timeout)

The SESReturnWaitEvent API is used to return the results of an event and re-wait on the same event instance (for example same EventID) in a single operation.

Parameters:

pEventInfo - (PSESEVENT) input/output

```
typedef struct _SESEVENT
{
    ULONG          DaemonNumber;
    ULONG          ReqPID;
    ULONG          ReqTID;
    SESSecurityContext SCH;
    ULONG          EventID;
    ULONG          Event;
```

```

        ULONG                EventStatus;
        SESEVENTDATA         EventData;
    } SESEVENT, *PSESEVENT;

```

DaemonNumber - (ULONG) input

The DaemonID specified here must be a valid ID returned from a previous call to SESRegisterDaemon().

ReqPID - (ULONG) output

Process ID of process which initiated this event.

ReqTID - (ULONG) output

Thread ID of process/thread which initiated this event.

SCH - (SESSecurityContext) output

SecurityContext of process/thread which initiated this event.

EventID - (ULONG) input

EventID is a unique ID assigned by SES to track the instance of the event. The EventID allows SES to associate a ReturnWaitEvent with a specific instance of an event type.

Event - (ULONG) output

The event parameter identifies the event type corresponding to this ReturnWaitEvent. Refer to SESWaitEvent Event parameter for a complete list of LSS events.

EventStatus - (ULONG) input/output

The EventStatus parameter is used to pass the results of an event to logon shell services and/or other SCAs and to receive status from logon shell services and/or previous SCAs. Refer to SESWaitEvent EventStatus for possible output values. Refer to SESReturnEventStatus EventStatus for possible input values.

EventData - (SESEVENTDATA) input/output

Event specific data. This structure is a union of event specific structures defined for each event.

Timeout - (ULONG) input

Timeout specifies, in milliseconds, how long to remain blocked waiting for an event. A value of negative one (-1) indicates an infinite wait.

Returns

Standard OS/2 API return codes.

0 NO_ERROR

SES API return codes.

35528 SES_INVALID_EVENTNUM
35529 SES_INVALID_DMONNUM
35534 SES_EVENT_TIMEOUT (only applicable for SES Daemon)
35535 SES_SCA_NOT_REGISTERED (only applicable for SES Daemon)

Remarks

The SESReturnWaitEvent call is used to return the results of the eventprocessing to logon shell services and/or subsequent SCAs and re-wait on the same event instance (for example, same EventID) in a single operation.

This API is used extensively by the SES logon shell services daemon to wait multiple times for the same event. This API cannot be used to wait on a new instance of an event or a new event type. For example, a thread that returns from a wait on a logon event cannot use this call to return status from the logon event and wait for a logoff event.

F.6 SESStartEvent

APIRET APIENTRY SESStartEvent(PSESSTARTEVENT pSES)

The SESStartEvent API is used to initiate an logon shell services event.

Parameters:

pSES - (PSESSTARTEVENT) input/output

```
typedef struct _SESSTARTEVENT
{
    ULONG          Event;
    ULONG          EventStatus
    SESEVENTDATA   EventData;
} SESSTARTEVENT, *PSESSTARTEVENT;
```

Event - (ULONG) input

The Event parameter defines the event that is being initiated. The following events are defined for SESStartEvent():


```

SES_EVENT_LOGON
SES_EVENT_LOGOFF
SES_EVENT_SHUTDOWN
SES_EVENT_LOCK
SES_EVENT_UNLOCK
SES_EVENT_CHANGE_PASSWORD
SES_EVENT_CREATE_PROFILE
SES_EVENT_DELETE_PROFILE
SES_EVENT_IA

```

EventStatus - (ULONG) input

The EventStatus parameter is only used to initiate a logon or unlock event. The following status values are defined:

```

SES_STATUS_NO_ERROR
SES_STATUS_ID_ONLY
SES_STATUS_GUEST_USER
SES_STATUS_AUTOGUEST (only applicable for PSS daemon)

```

EventData - (SESEVENTDATA) input/output

Event specific data. This structure is a union of event specific structures defined for each event.

Returns

Standard OS/2 API return codes.

```

0      NO_ERROR

```

SES API return codes.

```

35517  SES_INVALID_AUTHORITY
35521  SES_INVALID_EVENT_DATA
35527  SES_EVENT_FAILURE
35533  SES_EVENT_INVALID
35536  SES_CANCEL

```

Remarks

A process issues an SESStartEvent to initiate an logon shell services event. This call does not return until all authorized SCAs waiting on this event have executed and returned their status. It is not necessary to be defined as an security control authority or to register via the SESRegisterDaemon API in order to start an logon shell services event.

The logon, unlock, and identification and authorization events may pass in event specific data to initiate the event, for example, if the name and password of the user are already known to the calling process, it may pass these in on the SESStartEvent call so that the user identification authority doesn't try to get these from the user again. The change password and create/delete user profile events must pass in event specific data to initiate the event. Other events do not have corresponding event specific data.

Only a process with personal shell services authority may initiate a logon event with an EventStatus of SES_STATUS_AUTOGUEST.

A return code of SES_CANCEL is returned when a logon or unlock event is aborted. SES_CANCEL should be interpreted as a request initiated by the user to cancel the operation, it should not be interpreted as an unsuccessful logon/unlock attempt.

F.7 SESWaitEvent

```
APIRET APIENTRY SESWaitEvent(PSESEVENT pEventInfo,
                             ULONG      Timeout)
```

The SESWaitEvent API is used to wait on logon shell services events.

Parameters:

pEventInfo - (PSESEVENT) input/output

```
typedef struct _SESEVENT
{
    ULONG      DaemonNumber;
    ULONG      ReqPID;
    ULONG      ReqTID;
    SESSecurityContext SCH;
    ULONG      EventID;
    ULONG      Event;
    ULONG      EventStatus;
    SESEVENTDATA EventData;
} SESEVENT, *PSESEVENT;
```

DaemonNumber - (ULONG) input

The DaemonID specified here must be a valid ID returned from a previous call to SESRegisterDaemon().

ReqPID - (ULONG) output

Process ID of process which initiated this event.

ReqTID - (ULONG) output

Thread ID of process/thread which initiated this event.

SCH - (SESSecurityContext) output

SecurityContext of process/thread which initiated this event.

EventID - (ULONG) output

EventID is a unique ID assigned by SES to track the instance of the event. The EventID allows an security control authority receiving the event to differentiate between multiple notifications of the same event type.

Event - (ULONG) output

The Event parameter indicates which event caused the thread to unblock. Note that the events started via SESStartEvent may cause the thread to unblock in various states of the event processing. The security control authority must be able to distinguish the state of the event when it is unblocked.

For example, a single thread of an security control authority might be responding to events as both a user identification authority and client logon authority since a single event such as logon would unblock this thread for both user identification authority and client logon authority processing, the thread needs to be able to distinguish between these states. The following list defines the possible event states that an security control authority may encounter:

```
SES_EVENT_LOGON
SES_EVENT_LOGON_UIA
SES_EVENT_LOGON_SLA
SES_EVENT_LOGON_CLA
```

```
SES_EVENT_LOGOFF
SES_EVENT_LOGOFF_QUERY
SES_EVENT_LOGOFF_SLA
SES_EVENT_LOGOFF_CLA
```

```
SES_EVENT_SHUTDOWN
SES_EVENT_SHUTDOWN_QUERY
SES_EVENT_SHUTDOWN_SLA
SES_EVENT_SHUTDOWN_CLA
```

```
SES_EVENT_LOCK
SES_EVENT_LOCK_QUERY
SES_EVENT_LOCK_SLA
SES_EVENT_LOCK_CLA
```

SES_EVENT_UNLOCK
 SES_EVENT_UNLOCK_UIA
 SES_EVENT_UNLOCK_SLA
 SES_EVENT_UNLOCK_CLA

 SES_EVENT_CHANGE_PASSWORD

 SES_EVENT_CREATE_PROFILE

 SES_EVENT_DELETE_PROFILE

 SES_EVENT_IA

 SES_EVENT_SEND_SECURITY_CONTEXT

 SES_EVENT_PROCESS_CREATION

EventStatus - (ULONG) output

The EventStatus parameter indicates the results of the event. This parameter is mainly used by the SES daemon. The following table shows the possible EventStatus values returned by each security control authority for the various event states. Event states not listed do not return an EventStatus.

Event State (SES_EVENT_)	Event Status	Source SCA
LOGON (1)	SES_STATUS_NO_ERROR SES_STATUS_EVENT_FAILURE SES_STATUS_USER_AUTHENTICATED (2) SES_STATUS_USER_UNAUTHENTICATED (2) SES_STATUS_CANCEL SES_STATUS_NOT_APPLICABLE (2) SES_STATUS_NOT_AVAILABLE (2) SES_STATUS_ID_ONLY (2) SES_STATUS_GUEST_USER (2) SES_STATUS_AUTOGUEST	SLA SLA UIA UIA UIA UIA UIA UIA UIA PSS
LOGON_SLA	SES_STATUS_USER_AUTHENTICATED SES_STATUS_USER_UNAUTHENTICATED SES_STATUS_GUEST_USER SES_STATUS_AUTOGUEST	SES/SLD SES/SLD SES/SLD PSS
LOGON_CLA	SES_STATUS_NO_ERROR SES_STATUS_GUEST_USER SES_STATUS_AUTOGUEST	SES/CLD SES/CLD SES/CLD
LOGOFF (1)	SES_STATUS_NO_ERROR SES_STATUS_EVENT_FAIL	SLA SLA

Event State (SES_EVENT_)	Event Status	Source SCA
SHUTDOWN (1)	SES_STATUS_NO_ERROR SES_STATUS_EVENT_FAIL SES_STATUS_INTEGRITY_VIOLATION	SLA SLA SES
SHUTDOWN_QUERY	SES_STATUS_NO_ERROR SES_STATUS_INTEGRITY_VIOLATION	SES SES
SHUTDOWN_SLA	SES_STATUS_NO_ERROR SES_STATUS_INTEGRITY_VIOLATION	SES SES
SHUTDOWN_CLA	SES_STATUS_NO_ERROR SES_STATUS_INTEGRITY_VIOLATION	SES SES
LOCK (1)	SES_STATUS_NO_ERROR SES_STATUS_EVENT_FAIL	SLA SLA
UNLOCK (1)	SES_STATUS_NO_ERROR SES_STATUS_EVENT_FAIL SES_STATUS_USER_AUTHENTICATED (2) SES_STATUS_USER_UNAUTHENTICATED (2) SES_STATUS_NOT_APPLICABLE (2) SES_STATUS_NOT_AVAILABLE (2) SES_STATUS_GUEST_USER (2)	SLA SLA UIA UIA UIA UIA UIA
UNLOCK_SLA	SES_STATUS_USER_AUTHENTICATED SES_STATUS_USER_UNAUTHENTICATED SES_STATUS_GUEST_USER	SES/SLD SES/SLD SES/SLD
IA (1)	SES_STATUS_USER_AUTHENTICATED (2) SES_STATUS_USER_UNAUTHENTICATED (2) SES_STATUS_NOT_APPLICABLE (2) SES_STATUS_NOT_AVAILABLE (2) SES_STATUS_ID_ONLY	UIA UIA UIA UIA UIA
PROCESS_CREATION (1)	SES_STATUS_NO_ERROR SES_STATUS_EVENT_FAIL	SLA SLA

Table 12. Event Status and Sources

NOTES:

- (1) Only applicable for SES daemon.
- (2) Only propagated to the SLD.

EventData - (SESEVENTDATA) input/output

Event specific data. This structure is a union of event specific structures defined for each event.

Timeout - (ULONG) input

Timeout specifies in milliseconds how long to remain blocked waiting for an event. A value of negative one (-1) indicates an infinite wait.

Returns**Standard OS/2 API return codes.**

0 NO_ERROR

SES API return codes.

35529 SES_INVALID_DMONNUM
35534 SES_EVENT_TIMEOUT

Remarks

The events returned for an SESWaitEvent are more extensive than the list of events that can be initiated via the SESStartEvent. There are two reasons for this. First all, some events have phases (for example, SES_EVENT_LOGON_UIA, SES_EVENT_LOGON_SLA, etc.). Security enabling services determines which phase of logon is currently in progress.

The event parameter is updated by security enabling services to reflect the current phase. If a security control authority is a system logon authority and a user identification authority, and has a single thread waiting for a logon event, the security control authority is able to determine which phase of logon is in progress by examining the event parameter.

The second reason for having more wait events than start events is that there is no corresponding start event for send security context. This event is triggered as a result of a security control services call. The status returned to the security enabling services daemon from UIAs for logon, unlock, and I&A events is only used as input to the system logon driver. The status returned to the system logon authority from a logon and unlock event is passed in by the security enabling services daemon. However, it is generated by the system logon driver.

An EventStatus of SES_STATUS_INTEGRITY_VIOLATION during a shutdown event indicates an unrecoverable error. SCAs should not modify the EventStatus field and should not request any input from the user. The system logon authority should terminate all user processes and display a window indicating a security error has occurred. This

event is generated by a call to `SESlogIntegrityViol` with a `LOG_HALT` parameter.

Any security control authority with server program authority waiting on an logon shell services event will be associated with the context of the thread that started the event (or invoked the `SESSendSecurityContext` call) when it is unblocked. Receiving notification of events requires an security control authority to have the appropriate authority.

For example, an security control authority must have user identification authority to wait on the `SES_EVENT_LOGON_UIA` event. The following table specifies the required authority for each of the defined events. The security enabling services daemon waits on all events and receives control after each security control authority returns status.

Event State	SCA
SES_EVENT_LOGON	PSS
SES_EVENT_LOGON_UIA	UIA
SES_EVENT_LOGON_SLA	SLA
SES_EVENT_LOGON_CLA	CLA
SES_EVENT_LOGOFF	SLA
SES_EVENT_LOGOFF_SLA	SLA
SES_EVENT_LOGOFF_CLA	CLA
SES_EVENT_LOGOFF_QUERY	SLA
SES_EVENT_SHUTDOWN	PSS
SES_EVENT_SHUTDOWN_SLA	SLA
SES_EVENT_SHUTDOWN_CLA	CLA
SES_EVENT_SHUTDOWN_QUERY	SLA
SES_EVENT_LOCK	PSS
SES_EVENT_LOCK_SLA	SLA
SES_EVENT_LOCK_CLA	CLA
SES_EVENT_LOCK_QUERY	SLA
SES_EVENT_UNLOCK	PSS
SES_EVENT_UNLOCK_UIA	UIA
SES_EVENT_UNLOCK_SLA	SLA
SES_EVENT_UNLOCK_CLA	CLA
SES_EVENT_CHANGE_PASSWORD	UIA, CLA
SES_EVENT_CREATE_PROFILE	UIA, CLA, SLA
SES_EVENT_DELETE_PROFILE	UIA, CLA, SLA
SES_EVENT_IA	UIA
SES_EVENT_SEND_SECURITY_CONTEXT	Any
SES_EVENT_PROCESS_CREATION	SLA

Appendix G. Security Enabling Services Error Codes

The following list defines the security enabling services error codes which may be returned from security enabling services APIs or KPIs:

0	SES_NO_ERROR
35281	SES_INVALID_FUNCTION
35286	SES_INVALID_HANDLE
35288	SES_NOT_ENOUGH_MEMORY
35302	SES_BAD_COMMAND
35311	SES_GENERAL_FAILURE
35367	SES_INVALID_PARAMETER
35391	SES_BUFFER_OVERFLOW
35395	SES_PROTECTION_VIOLATION
35403	SES_INVALID_NAME
35516	SES_INVALID_SUBFUNCTION
35517	SES_INVALID_AUTHORITY
35518	SES_PROCESS_NONEXISTENT
35519	SES_THREAD_NONEXISTENT
35520	SES_SES_DISABLED
35521	SES_INVALID_EVENT_DATA
35522	SES_UIA_NONEXISTENT
35523	SES_NOTIF_DISABLED
35524	SES_ID_NOT_FOUND
35525	SES_INVALID_TAG_LENGTH
35526	SES_NO_CURRENT_USER
35527	SES_EVENT_FAILURE
35528	SES_INVALID_EVENTNUM
35529	SES_INVALID_DMONNUM
35530	SES_INVALID_MESSAGE LENG
35532	SES_UNBLOCKED

35533	SES_EVENT_INVALID
35534	SES_EVENT_TIMED_OUT
35535	SES_KBD_MOUSE_ERROR
35536	SES_CANCEL

Appendix H. Customer Thoughts on Security Products

IBM has spent time discussing OS/2 security with its customers. These discussions involved the requirements for security in an enterprise environment and what our customers believed would make a good security product for the OS/2 environment. This chapter will provide information on the feedback that was received from these meetings and may prove useful to ISVs who are developing security products for an enterprise environment.

The information that follows is only a guide based on the feedback that has been received from IBM customers, it is not intended as a comprehensive list of security product requirements.

There are two types of users that exist in enterprises, administrators and users. These people have different requirements when it comes to a security product.

- Administration

The security administrator has most of the requirements that exist in a secure environment. These can be divided into the following categories:

1. Functional requirements
2. Administration facilities

- Users

1. Easy to use
2. No knowledge that it exists
3. Help for problems

This appendix will concentrate on the security administrator and will look at the functional and administration requirements that have been mentioned by IBM customer. Not all of these requirements will be valid for all situations and indeed in many situations, a low function easy to use product may be preferable to a high function product, which take longer to administer or be more difficult to set up and use.

1. Functional Requirements

- Password control. There are various ways to implement password control. Under the administration facilities we will look more at the type of password control requests that have been looked for by customers.

- Interfaces to other authentication devices, for example, smart cards, and biometrics devices.
- Host interfaces to products like RACF, ACF2, VFP
- Access control that can be defined by the administrator
 - Boot protection. Prevent the user from booting using the diskette drive. Provide this support by software, or hardware adapter support.
 - Pre-boot authentication when attempting to boot from the hard disk.
 - Ability to enable or disable the ALT-F1 facility.
 - File level control. Prevent unauthorized access to specific files on the system.
 - Directory level control. Prevent unauthorized access to certain directories on the system.
 - Drive level control. Prevent unauthorized access to specific drives or partitions.
 - Restrict access to command line in OS/2, DOS, Windows and from within applications.
 - Restrict access to devices such as the following:
 - Diskette drive
 - Hard files
 - COM port
 - Parallel port
 - Keyboard (for example, ALT-CRTL-DEL)
 - Removable media (CD-ROM, tape, etc)
- Encryption. Encryption requirements varied from customer to customer. A selection of the requirements mentioned are the following:
 - File, directory, disk level encryption
 - Per user (for example all a certain user's files encrypted)
 - Key definition, public-key or symmetric cryptography
 - Supported encryption algorithms

- User database. The user database must be secure there should also be the option of the following:
 - Remote storage
 - Local storage
 - Both local and remote storage
 - Make use of another system's user database, for example RACF's user database

If it is not possible to use another system's user database, then database synchronization would be required. There should be a product default for synchronization time, and a customer defined option.
- Allow definition of privileges, such as, read, write, create, delete, execute, copy, move, permissions.
- Provide interfaces to other PC operating systems or other versions of products, such as, DOS, Windows NT, Win-95, OS/2, OS/2 with SES, MAC, AIX.
- User shell management (PDF type function) which has the following:
 - Definable setup
 - In-built default shell setup
 - Customer replaceable shell definition
 - User personalization

Allow the user personalization of items such as, data folders, screen colors, icons to be enabled or disabled.
- Provide a secure installation.
- Restrict working hours. This is a requirement to control out of hours access by users.
 - Default.
 - Define specific requirements per user.
 - Access permissions differ at different times.
- Single signon features. Provide the ability to sign on automatically to other systems through products like, LAN Server, Netware, CM/2, DB2/2, TCP/IP.
- Disaster recovery procedures.
- Diagnostic tools.

- Lockup. Provide a lockup facility that can be activated depending upon:
 - Time of day.
 - Inactivity.
 - User selected.
- Overheads. The impact of security should be kept to a minimum. Particular attention should be paid to, memory, disk space, performance, and hardware overheads.
- Certification compliance. Products should aim to provide functional compliance with acknowledged security certification standards, D2, C2, E2/FC2, B1 (through design). It could also be advantageous for products to gain certain certifications as part of the products medium or long term strategy.
- Dial-in options with the facility to:
 - Enable/disable dial-in
 - Receive automatic dial-back
 - Have reduced permissions
 - Have a definable dial-back number
- Support medialess workstations.
- Backup/Restore. Allow the user database to be backed in a secure, encrypted manner, to local or remote devices.
- Configurable logon panel. Allow the logon message, panel background or the whole panel to be replaced.
- Provide a definable screensaver option.
- National language support.
- Audit facilities. More detail on audit facilities will be provided as part of the administration facilities section.

2. Administration Facilities

- Installation.
 - Ease of installation using facilities such as, CID, NVDM
 - Automated SES/Non-SES install
 - Install SES version if SES enabled
 - Install for non-SES if no SES present

- Allow administrator selection of SES/non-SES override
 - Non-SES reduced function set
- Access permissions.
 - Ability to propagate access permissions
 - Provision of default access permissions for new files/directories
- Administration levels. The ability to provide an administration hierarchy would be useful.
 - System administrator
 - Area administrator
 - User group administrator

Product administration should provide options for administering the system.

- GUI interface
 - Administration script
 - Usable from other system (for example, MVS/RACF).
 - May be used to manage other systems.
 - Action recorder to produce script for GUI interface
- User Management Facilities.
 - User and group profiles.
 - User ID expiry (date/time).
 - No limit on number of user definitions.
 - Ability to read existing UPM definitions.
- Enforced password procedures.
 - Product default.
 - Administrator definable via user exit.
 - Administrator definable per character.
 - Number of characters.

(For example 5 chars) 1=alpha, 2=numeric, 3=alpha/numeric, 4=alpha/numeric, 5=spec characters (&)
 - Product allows for a minimum of 0, maximum of 16 as password length.

- Local or remote enforcement.
 - Definable
 - If no remote access check local (definable)
- Password Expiry.
 - Date.
 - Time period.
 - Definable.
 - Force.
 - Password reuse criteria.
 - Password synchronization. Synchronization should be performed with other systems and programs:
 - Local - Screensaver, keyboard lock, user logon
 - Remote - Host sessions, LAN
- Software license control.
 - Copy count
 - Installation restrictions
 - Interfaces to license management products
- Auditing. There are many facilities that could be useful with regard to auditing.
 - Pre-defined audit trails.
 - Customer defined audit trails.
 - Definable audit reporting. Audit reports should be able to be stored locally, remotely or both.
 - Audit log transmission. This should be definable in terms of the date, time etc. A product default should also be included.
 - Audit log size.
 - Default (disk space)
 - Max size definition
 - Wrap around option
 - Definable action at maximum size with a default of flag to administrator
 - Audit log analysis tools

- Report generators

As stated earlier this list is not complete, as it doesn't include some of the more obvious considerations such as, performance and data import and export. It should provide a general idea on the type of functions that customers are investigating.

Glossary

A

access. An interaction to obtain information from any source or to communicate.

access control. A security mechanism to protect access to programs or information, for example RACF.

access method. The technique or program used for moving information or communicating.

access time. The time from issuing a command to read or write to a file on disk until the physical read or write is actually carried out.

access object. Object of the rights administration on which subjects access. Access objects are generally files and interfaces.

administration object. Rights administration object which contains rules for several users. The administration object can be assigned to domain, user groups, users, workstations and trusted programs.

adapter card. A printed circuit card for attaching input/output devices to a computer.

address. In data communication, the unique code assigned to each device or workstation connected to a network.

AIX. Advanced Interactive eXecutive: IBM's version of UNIX.

algorithm. A set of rules to solve a problem in a number of steps.

API. Application Program Interface. Interface through which programs can communicate.

ASCII. American National Standard Code for information interchange: a coded character set used on personal computers.

authenticate. A process to verify the integrity of data or a message, or to verify the user of an information system or protected source.

authentication. Confirmation of a given identity, using password, smart card or ID token.

authentication server. Part of a trusted security base. Responsible for authenticating identities of clients. Maintains passwords and group membership information for users.

authorize. Granting someone the right to use a computer, application or database; is also used in connection with programs to grant complete or restricted access to an object, resource or function.

audit. Logging of user actions for audit purposes.

auditor. Role with regard to rights administration or log evaluation. DP auditor, responsible for auditing DP systems.

B

batch file. On a personal computer, a file having the extension .BAT, which contains a list of commands that are executed when the file is called.

bit map. (1) An area of memory or storage that contains the pixels representing an image, arranged in the sequence in which they are normally scanned, to display the image.
(2) A representation of an image by an array of bits.

BIOS. The area of the computer that controls incoming and outgoing signals.

boot. The process of starting up a personal computer.

boot drive. Logical drive from which the operating system is loaded. Generally it is the disk drive (C). It can, however, also be a floppy drive (A).

boot protection. Prevention of a system start from a medium other than the hard disk (or boot ROM). A system start with an operating system diskette is prevented.

bus. In a processor, a physical facility to transfer data; for example, ISA, MCA. Adapter cards are connected to a bus.

byte. A string that consist of a number of bits, treated as a unit, and representing a character.

C

CBC mode. Cypher Block Changing. DES encryption mode in which not only the key, but also the last encrypted 8 byte are used for encryption.

CCA. IBM Common Cryptographic Architecture; the IBM architecture for the cryptographic Application Programming Interface (API).

client/server system. A client/server system is a local network where PCs are connected to a server.

CD-ROM. Compact Disc Read Only Memory. A Compact disc specifically for storing data.

CD-ROM XA. Compact disk read-only-memory extended architecture. A partial implementation of CDI and DVI standards.

channel. A connection between a personal computer and one or more input/output devices.

checksum. The sum of a group of data, used for checking purposes.

cipher. A cryptographic system.

clip board. A temporary storage area used to pass information within a program or from a program to another.

CKDS. Cryptographic Key Data Set: a file containing cryptographic keys.

CMOS. A chip technology that requires little power, used to store vital configuration data of a PC.

COM. Serial interface for data communication. Is used to connect a modem, for example.

confidentiality of data. Protection against unauthorized reading. Can only be achieved by encryption.

configuration. (1) An arrangement of physical or logical devices that make up a (sub)system. (2) The manner in which the hardware and software of an information processing system are organized and interconnected.

control vector. In TSS: a 16-byte string that modifies the master key or a key-encrypting key to create another key that is used to encipher and decipher data or data keys.

controller. A device that controls the operation of input/output devices.

conventional memory. Random Access memory in a PC that DOS or OS uses as the first 640 K byte.

CRC. Cyclic Redundancy Check. Checksum which is not cryptographic.

cryptography. The principles and methods for encrypting plain text and decrypting cipher text to conceal its meaning.

CVC. Cryptographic checking of the contents of a magnetic stripe of a credit card (Mastercard).

CVV. Cryptographic checking of the contents of a magnetic stripe of a credit card (VISA).

D

data encrypting key. A key used to encipher, decipher or authenticate data.

data integrity. Data intactness. The intactness is checked by an integrity check (checksum method).

DEA. Data Encryption Algorithm. Method for encrypting data using a 64 block cipher that uses a 64 bit key, including 8 parity bits.

decipher. To convert encrypted text (cipher text) into the original text (plain text).

DES. Data Encryption Algorithm. Developed and published in 1977 by IBM. A US standard for encrypting data, available in two versions, full DES and commercial DES. Standard algorithm used by international banks. Symmetrical algorithm with a 56 bit long key. The CBC mode is regarded as secure.

device. A physical unit of a computer system, often used for input/output operations, which can be used in a logical order or have a logical address.

directory. A hierarchically structured logical area for storing files on a hard disk or diskette, which may include one or more sub-directories.

DOS. Disk Operating System: an operating system for personal computers.

domain. Organizational unit which is commonly managed. Also known as system.

E

EBCDIC. Extended Binary Coded Decimal Interchange Code: a coded character set used on main-frames.

emulator. Imitator.

encipher. To convert an original text (plain text) into encrypted form (cipher text).

encrypt. Synonym of encipher.

ESS. Establish Secure Session; a cryptographic means by which hardware components establish.

extended attribute. The OS/2 method of attaching additional information to a file object. Extended attributes can be used to store notes on file objects (for example, version, history), categorize file objects for example, file type, associations, describe the format of data contained in the file object, or append additional data to the file object. They are stored separately from the file object they are associated with and are managed by the file system attached to the file object.

extended data. User-defined information, including multimedia information, about Light Table folder objects. Such information goes beyond what is available in OS/2 standard data. Extended data includes user-defined columns, and may come either from a supported database or from extended attributes.

extension. In the name of a file, the three letters following the dot, which often indicates the type of file, for example, BAT in AUTOEXEC.BAT indicates batch file.

F

folder. A directory as represented on the OS/2 desktop.

font. The characters available for text with a given set of attributes.

G

generation. The number of copies away from the original.

graphical user interface (GUI). A type of computer interface consisting of a visual metaphor of a real world scene, often of a

desktop. Within that scene are icons representing objects, that the user can access and manipulate with a

graphic. Any pictorial representation of information.

graphics. Text or pictorial artwork created by a variety of means, such as electronic generated graphics software and the pressed onto the video-discs.

H

hertz. A measure of frequency equivalent to cycles per second.

host. A host is a "large" computer which acts as a "host" for terminals or workstation PCs with terminal function.

HPFS. High Performance File System. HPFS provides long file name support and fast access to very large disk volumes.

I

I&A. Identification and Authentication; see identification and authentication.

icon. A pictorial representation of a function that you can select to carry out this function.

identification. Identification of a DP user to the system with a user ID (Name or Personal-No).

ID token. Checkcard-sized special calculator with a keypad to generate a dynamic password according to the Challenge/ Response method.

interface. Hardware and/or software that links systems, programs, or devices.

I/O. Input/Output: pertaining to a device that performs input and/or output operations.

IPL. Initial Program Load; the initialization of a computer.

ITSEC. Information Technology Security Criteria. Criteria book issued by the EU, worked out by four member states - France, The Netherlands, UK and Germany. Evaluations to determine security are made on the basis of ITSEC. The products are awarded a certificate if they meet the requirements. F-C2, E2 means that the security functions of class F-C2 and the evaluation level E2 were reached. F-C2, E2 is the commercial standard.

image. A still picture or one frame.

interlace. The technique of using more than one vertical scan to reproduce a complete image. In television, a 2:1 interlace is used, giving two vertical scans per frame. One scan will be odd lines, the other will be even lines.

K

KB. kilobyte: 1024 bytes.

Kilohertz (kHz). Thousands of cycles per second.

KEK. Key Encrypting Key; a key used to encrypt, decrypt or authenticate keys for transmissions.

key. In computer security, a sequence of symbols used with a cryptographic algorithm for encrypting or decrypting data.

key administration. Administration program which administers the keys and restores them if they are destroyed.

key token. In TSS, a data structure that can contain a cryptographic key, a control vector, and other information related to the key.

KM. Master Key: the top level key in a hierarchy of key encrypting keys.

KMC. Key Management Center; a department for managing cryptographic keys.

L

local area network (LAN). A data network located on the user's premises in which serial transmission is used for direct data communication between workstations.

leading logon type. The type of logon that must be done before other logons can be used.

link. (1) A logical connection, (2) A physical connection, (3) An interconnection between data or programs.

LPT. Parallel interface to attach a printer, streamer, etc.

M

MAC. Message Authentication Code. Cryptographic checksum, based on the DES-MAC. The encryption result, the last 8 encrypted bytes, is the checksum.

migrate. (1) To move data from one storage media to another, (2) To change to a new operating environment.

module. Program module which takes over a specific function. Example: logging in a linear file on the server, or logging in a local ring buffer file. Modules can be swapped by the system administrator.

multi-tasking. A technique that allows several processes to appear to run simultaneously, even though the computer only has one CPU. This is achieved by sequentially switching the CPU between tasks.

multiplexer. A device that interleaves the transmission of several input signals over a connection such that the input signals can be recovered.

N

network. (1) A network of devices and software connected for information interchange, (2) An arrangement of nodes and connecting branches to interconnect computers, terminals and workstations.

node. In a network, a point at which one or more units are connected. Each node has a network address.

O

object. (1) Resource of the DP system, such as files, interfaces, networks, etc. (2) A visual component of a user interface that a user can work with to perform a task.

OEM. Equipment sold by another manufacturer.

off line encryption. File encryption in a separate work process.

on line encryption. Transparent encryption during the Read or Write process. On-line encryption in Safe Guard Professional can be both file and sector oriented.

orange book. Security standards from the US Department of Defense that specify different security levels.

P

panel. The set of information displayed on the screen of a display station.

password. In computer security, a string of characters used to gain access to a computer file or system, during sign-on or at a later time. A PIN can be considered as a password.

path. (1) In a network, any route between two nodes, (2) The route traversed by information exchanged between two network devices, (3) A

command in DOS related to the path through its (sub)directory structure to reach a file.

pause function. "Logoff" for a short work interruption. The screen is blanked, the keyboard can only be used for special entries and the work station is locked. To continue work, the user who triggered the pause, must log on again.

PCF. Programmed Cryptographic Facility: IBM program for enciphering and deciphering text and for key management.

pel. Picture element. The smallest building block that a screen or bit-mapped image can display. Pel and pixel can be used interchangeably.

pixel. A single point of an image, having a single pixel value.

pop-up. A window which appears on the screen to display text, graphics, messages, or documents.

PIN. Personal Identification Number: The secret number that a user must remember to gain access to a service, can be used in conjunction with an IBM Personal Security Card.

plain text. Non-encrypted data.

private key. In computer security, a secret key used to encrypt data.

protocol. Rules and agreements for communication between devices.

PSC. Personal Security Card: A standard smart card with a processor for executing DES-based cryptographic functions. It can hold more than 4000 bytes of data, including the characteristics of a signature for the purpose of verification. Incorporated. Currently supported by IBM.

public key. In computer security, a widely known key used to encrypt data, the encrypted

data must be decrypted with a related private key.

R

RAM. Random Access Memory: Memory where data can be written and read directly.

reuse. This is the recreation of the original status of a file, the main memory or the swap file after it has been deleted or after the user has logged off.

resolution. The ability of an image reproducing system to reproduce fine detail.

RGB. A method of processing color images according to their red, green, and blue color content. Colors can also be measured on an HIS color scale. Contrast with composite, Y/C, and YUV.

role. Role which the user plays, particularly with regard to rights administration. A role is assigned specific administrative rights, for example system administrator, auditor, accessory or simple user.

ROM. Read Only Memory: Memory to store programs or data permanently.

S

SAF. System Authorization Facility: a program that provides access to RACF.

SAPI. Interface for application call via TSS hardware using special verbs for executing security information.

scanner. A device which performs scanning.

schema. The data-definition part of a database table.

sealing. Sealing of data for purposes of the integrity check.

security. The protection of data, system operations and devices from accidental or intentional ruin, damage or exposure.

security architecture. IBM strategy and architecture for secure information systems.

security enabling services. Add-on in OS/2, which acts as interface between ISV products and OS/2.

security server. Server, which saves data important for the security of the security sub-system and carries out authentications.

Security officer. Role in rights administration. Executive responsible for DP security. Generally he is responsible for the initial installation or de-installation, as well as for key management.

server. On a LAN, a station that provides services to other stations; for example, file server, print server, and security server.

session. The period of time that a network connection lasts, including the establishment and release of the connection.

session key. Key valid for only one session. A session is the time between logon and logoff.

signature verification. In TSS an optional feature of the security interface unit for user verification through their signature, written with a signature verification pen and recognized by a signature verification module on the cryptographic adapter.

Single SignOn (SSO). First logon to a network. The Safe Guard Professional SSO function enables the automatic logon to other computers in the network, after the user has authenticated himself once. Passwords are automatically synchronized.

software configuration utility. One of the utilities distributed with the workstation security

services program for configuring security servers and device drivers.

sub-directory. A directory contained within another directory in the file system hierarchy.

subject. A subject is a user or process which accesses objects (files, etc.).

subsystem. A secondary or subordinate system, usually capable of operating independently.

system administrator. User in a DP system who plays an important administrative role. He is responsible for the administration of rights.

T

token. Bit string (combination of bits) to enable the execution of a specific operation.

token-ring. An IBM network with a ring topology that passes tokens from one attaching device to another.

trusted programs. Programs, which either do not have the user 's full access rights or whose rights go over and beyond these.

trusted workplace shell. Workplace which corresponds to the individual rights profile of the user. It cannot be changed by the user.

TSS. Transaction Security System: A series of cryptographic products for providing a secure workstation.

U

user. User in a DP system. In Safe Guard Professional a user generally does not have administration rights.

user ID. User identification, name for a user in the system.

W

wildcards. Placeholders for any number of other characters. An asterix (*) stands for a permitted set of any other characters. A question mark (?) stands for any other single character.

workstation. A terminal or microcomputer that often is connected to a main frame or a

network, at which the user can perform applications.

X

XGA. Extended graphics array. A high resolution display with a display matrix (pels) of 1,024 x 768 at 256 colors. XGA can also provide more colors with reduced resolution (640 x 480 at 65,536 colors).

List of Abbreviations

ACA	access control authority	DAC	discretionary access control
ACL	access control list	DBMS	database management system
AGH	agent group handle	DCE	distributed computing environment
APA	agent process authority	DEA	data encrypting algorithm
APH	agent process handle	DES	data encrypting standard
AUH	agent user handle	DIB	device independent bitmap
AIX	advanced interactive executive	DLL	dynamic link library
ANSI	American national standards institute	DMA	direct memory access
API	application programming interface	DOS	disk operating system
APAR	authorized program analysis report	DoD	department of defense (USA)
ASCII	American national standard code for information interchange	EBCDIC	extended binary coded decimal interchange code
AT	advanced technology	EC	engineering chance
AUH	agent user handle	EGF	effective group flag
BIOS	basic input output system	EISA	extended industry standard architecture
BGA	business graphics adapter (8514/A card)	EPF	effective process flag
BMP	bit-mapped graphics	ESS	establish secure session
CGH	client group handle	EUF	effective user flag
CKDS	cryptographic key data set	FAT	file allocation table
CLA	client logon authority	FSD	file system driver
CLD	client logon driver	GUI	graphical user interface
CMOS	complimentary metal oxide semiconductor	HPFS	high performance file system
CPH	client process handle	IBM	International Business Machines
CPU	central processor unit	ICIS	installation, configuration, initialization support
CTCPEC	Canadian trusted computer product evaluation criteria	ICRF	integrated cryptographic facility
CUH	client user handle	IFS	installable file system
CV	control vector	IMS	information management system
CVC	card verification code	IPL	initial program load
CVV	card verification value	IRQ	interrupt request
		ISA	industry standard architecture

I&A	identification and authentication	PDF	personal desktop facility
ISO	international organization for standardization	PDD	physical device driver
ISS	installable security subsystem	PIN	personal identification number
ISV	independent software vendors	PM	Presentation Manager (OS/2)
ITSEC	information technology security evaluation criteria (European)	POSIX	portable operating system interfaces for computer environments
KB	kilobyte	PSC	personal security card
Kbps	kilobits per second	PSS	personal shell services
KEK	key encrypting key	PS/1	personal System/1
KM	master key	PS/2	personal System/2
KMC	master key center	PVD	password validation driver
KPI	kernel programming onterface	RACF	resource access control facility
LAN	local area network	REXX	restructured extended executor language
LSB	least significant bit	RISC	reduced instruction set computer
LSS	logon shell services	RLA	remote logon authority
LUF	local user flag	RPQ	request for price quotation
MAC	Message authentication code	RSCS	remote spooling communications subsystem
MB	megabyte (1,048,576 bytes)	R/W	read/write
Mbps	megabits per second	SAF	system authorization facility
MBps	megabytes per second	SAPI	secured application programming interface
MC	micro channel	SCA	security context authority
MCA	micro channel architecture	SCS	security context services
MVDM	multiple virtual DOS machine	SDK	software developers kit
MVS	multiple virtual system	SES	security enabling services
NLS	national language support	SIU	security interface unit
OEM	other equipment manufacturer	SKS	security kernel services
OS/2	Operating System/2	SLA	system logon authority
PAF	propagate authority flag	SLD	system logon driver
PC	personal computer	SLE	session level Eecryption
PCF	programmed cryptographic facility	SNA	system network architecture
PC AT	personal computer advanced technology		
PC XT	personal computer extended technology		

SPA	server process authority	TSS	transaction security system
SPI	service provider interface	UIA	user identification authority
SRPI	server requester programming interface	UPA	unprivileged process authority
SVGA	super video graphics adapter	UPM	user profile management
SWF	security workstation feature	WAN	wide area network
TCB	trusted computing base	VDD	virtual device driver
TCSEC	trusted Computer system evaluation criteria (Orange book)	VDM	virtual DOS machine
TMK	terminal master key	VM	virtual machine
TSO	time sharing option	WOAM	work place shell object access
TSR	terminate and stay resident	WPS	workplace shell
		XGA	extended graphics array

Index

A

A Process Sends Its Security Context to an SCA Process 184

A User Logs on Remotely to an Application Server 180

Abbreviations 317

ACA 39, 40, 42, 71, 85, 161

ACA Daemon 149

ACA flag 117

ACA, SLA and UIA 71

access 309

Access Control 8, 87, 309

access method 309

access object 309

access rights 32

access time 309

Accountability 11, 15

Acronyms 317

Activity Detection 193

adapter card 309

address 309

administration object 309

Agent Group Handle 103

agent process 35

Agent Process Handle 103

agent user 35

Agent User Handle 102

AIX 309

algorithm 309

An SPA Process Acts as proxy 186

An Untrusted Process 183

APA 41, 43, 72, 85, 162

APA and SPA 72

APA flag 118

APA Process 149

API 24, 77, 86, 87, 88, 89, 309

SCS 178

API Calls

CLD API 275

CLDInit 275

API Calls (*continued*)

CLDQueryCLA 275

Error Codes 299

LSS API 281

PVD API 279

PVDValidatePassword 279

Return Codes 299

SCS API 241

SESControlKBDMonitors 281

SESControlProcessCreation 241

SESCreateHandleNotify 242

SESCreateInstanceHandle 243

SESCreateSubjectHandle 244

SESDeleteHandleNotify 245

SESDeleteSubjectHandle 246

SESIactivityNotify 282

SESKillProcess 247

SESlogIntegrityViol 248

SESQueryAuthorityID 248

SESQueryContextStatus 249

SESQueryProcessInfo 250

SESQuerySecurityContext 253

SESQuerySubjectHandle 254

SESQuerySubjectHandleInfo 255

SESQuerySubjectInfo 256

SESRegisterDaemon 283

SESReleaseSubjectHandle 258

SESReserveSubjectHandle 259

SESResetThreadContext 260

SESReturnEventStatus 285

SESReturnWaitEvent 288

SESSendSecurityContext 261

SESSetContextStatus 262

SESSetSecurityContext 264

SESSetSubjectHandle 266

SESSetSubjectInfo 267

SESStartEvent 290

SESWaitEvent 292

SLD API 271

SLDInit 271

SLDQueryUIA 271

- API for Audit 88
- API for DAC 87
- API for Logon 86
- API for Single Signon 89
- API for Trusted Program Support 89
- Application Development 91
- Application Management 105
- Architecture 95
- ASCII 309
- Association 100, 102
- Association of Security Context with OS/2 IPC 177
- Assurance 11, 17
- Audit 8, 16, 21, 88, 309
- Audit Support 156
- auditor 309
- authenticate 309
- Authentication 16, 309
- authentication server 309
- authority flag 39
- Authority Flags 117, 176
 - ACA flag 117
 - APA flag 118
 - CLA flag 118
 - RLA flag 119
 - SLA flag 119
 - SPA flag 120
 - UIA flag 121
- Authority ID 113
- authorize 309
- Auto guest logon state 52
- Auto-Guest Logon State 136
- AUTOQUEST 51, 56, 58, 62

B

- Background (detached) process 41
- Background (detached) program 41
- batch file 309
- Biometrics 8
- BIOS 309
- bit map 309
- boot 309
- boot drive 310

- boot protection 310
- Building an ISS 85
- bus 310
- byte 310

C

- C2 21, 27, 31
- Callgate Level Support 153
- Callouts 98, 99, 152, 153, 154, 155, 156, 198
 - Order of Events 154
- Callouts for Callgate Level Support 99, 153
- Callouts for Logon Shell Services Trusted Path Support 99, 155
- Callouts for Multiple Virtual DOS Machine Support 99, 154
- Callouts for Security Enabling Services API Audit Support 99, 156
- Callouts for Security Relevant OS/2 System Calls 98, 152
- Callouts to the ISS security kernel for LSS functions 198
- CBC mode 310
- CCA 310
- CD-ROM 310
- CD-ROM XA 310
- Certification 9, 10
- change password 44, 192
- channel 310
- checksum 310
- cipher 310
- CKDS 310
- CLA 40, 49, 74, 85, 163
- CLA Daemon 149
- CLA Daemons 133
- CLA flag 118
- CLD 47, 69, 75, 85, 89
- CLD API 196, 275
 - CLDInit 275
 - CLDQueryCLA 275
- CLD Dynamic Link Library 134
- CLDInit 275
- CLDQueryCLA 275
- Client Group Handle 103

- Client logon authority 49
- Client logon driver 47
- Client Logon Driver API 196
- client process 35
- Client Process Handle 103
- Client Processes 186
- client user 35
- Client User Handle 102
- client/server system 310
- clip board 310
- CMOS 310
- COM 310
- confidentiality of data 310
- CONFIG.SYS 51, 62, 143, 226
- configuration 63, 142, 226, 310
 - CONFIG.SYS 143, 226
 - SECURE.SYS 144, 227
- context 32
- Context Status Management 128, 130
- contexts 36
- Continuous Protection 10, 12
- control vector 310
- controller 310
- conventional memory 310
- CRC 310
- create user profile 44
- Creating an Untrusted Child Process 183
- credentials 32, 36
- cryptography 310
- Customer Requirements 7, 301
- CVC 310
- CVV 310

D

- DAC 31, 66, 71
- data encrypting key 311
- data integrity 311
- DCE 35, 44
- DEA 311
- decipher 311
- DEFAULT OPERATION 54, 56, 60
- Definition of a guest user 136
- Definition of Auto-guest 138

- Definition of Local System Logon 135
- Definition of Security Context 110
- delete user profile 44
- DES 311
- Design Guidelines 91
- device 311
- Device Helpers 156
- directory 311
- Discretionary Access Control 8, 13
- DLL 69, 86, 87, 88, 89
 - DLL for Audit 88
 - DLL for DAC 87
 - DLL for Logon 86
 - DLL for Single Signon 89
 - DLL for Trusted Program Support 89
- domain 311
- DOS 311
- dynamic link libraries 69

E

- EBCDIC 311
- Effective Group Flag 114
- Effective Process Flag 114
- Effective Security Context 110
- Effective User Flag 115
- EGF 175
- emulator 311
- encipher 311
- encrypt 311
- EPF 175
- Error Codes 299
- ESS 311
- EUF 175
- Event Flows 190
 - Change Password 192
 - Lock 191
 - Logoff 191
 - Logon 190
 - Shutdown 191
 - Unlock 190
- Event Order 154
- Explicit logon as a guest user 137
- Explicit logon state 52, 136

Export 156
extended attribute 311
extended data 311
extension 311

F

File System API 150
File System Open Callout 159
File System Router 150
File System Services 156
fixpak 62
Flags 108, 114, 117
 ACA flag 117
 APA flag 118
 Authority 176
 CLA flag 118
 Effective Group Flag 114
 Effective Process Flag 114
 Effective User Flag 115
 EGF 175
 EPF 175
 EUF 175
 Local User Flag 115
 LUF 175
 PAF 176
 Propagate Authority Flag 116
 RLA flag 119
 SLA flag 119
 SPA flag 120
 Status 174
 UIA flag 121
folder 311
font 311

G

generation 311
graphic 312
graphical user interface 311
graphics 312
group credentials 32
Guest Logon Support 51
Guest user 136

H

handle 23
Helpers 99, 100, 156
hertz 312
Hooks 31
host 312
HPFS 312

I

I/O 312
I/O Device API 150
IO 43, 44, 66, 312
ICIS 29, 62, 142, 225
 CONFIG.SYS 143, 226
 Configuration 142, 226
 Initialization 144, 230
 Installation 142, 225
 SECURE.SYS 144, 227
icon 312
ID token 312
Identification 11, 16, 312
Identification and Authentication 7, 43, 44, 66
image 312
Implementation 95
Inactivity Detection 194
Inheritance 122
Inheritance Default 122
Inheritance Options 124
Initialization 54, 63, 144, 151, 171, 230
Installable security subsystem 21, 65
installation 62, 142, 151, 225
Installation, configuration and initialization
 support 62
Installation, configuration, initialization
 support 23, 29, 142, 225
Instance Handles 106
Integrity Violation Log 129
Inter-Process Communication 129
interface 312
interlace 312
Interoperation of security context
 authorities 74

Interoperation of SES and ISS 147

IPC 177

IPL 312

ISS 9, 21, 65, 152, 156

ISS API 95, 149

ISS components 85, 149

ISS API 149

ISS Credential Cache 149

ISS Security Daemon 149

ISS Security Kernel 149

ISS Credential Cache 149

ISS Design Guidelines 91

ISS Security Daemon 149

ISS Security Daemon(s) 95

ISS Security Kernel 95, 127, 149, 198

ISS Security Kernel Initialization 158

ISS summary 77

ITSEC 312

K

KB 312

KEK 312

Kernel Callouts Imported from the ISS 152

Kernel level operating system services 31

Kernel Services Exported to the ISS 156

key 312

key administration 312

Key Components 131

key token 312

keyboard 50

Keyboard Activity Detection 193

Keyboard Device Drivers 135

Keyboard Inactivity Detection 194

Keyboard Support 141, 192

Keyboard/Mouse Activity Detection 193

Keyboard/Mouse Inactivity Detection 194

Kilohertz 312

KM 312

KMC 312

KPI 24, 77, 86, 87, 88, 89

KPI for Audit 88

KPI for DAC 87

KPI for Logon 86

KPI for Single Signon 89

KPI for Trusted Program Support 89

L

leading logon type 313

Least Privilege Operation 15

link 313

Loader 150

local area network 313

Local User Flag 115

Lock 44, 49, 51, 60, 61, 62, 191

Lock State 53

Logoff 44, 49, 51, 57, 58, 60, 191

Logoff State 53

logon 50, 51, 56, 57, 58, 59, 86, 190

Logon shell services 23, 28, 43, 130, 189

Logon Shell Services (LSS) Scenarios 198

Logon Shell Services Kernel Programming

Interface 198

logon/logoff 56

LPT 313

LSS 28, 43, 44, 189

AuthoritySource 195

Callout for LSS Functions 198

Change Password 192

CLD API 196

Event Flows 190

Event status 194

Keyboard Activity Detection 193

Keyboard Inactivity Detection 194

Keyboard Support 192

Lock 191

Logoff 191

Logon 190

LSS API 197

LSS KPI 198

Mouse Activity Detection 193

Mouse Inactivity Detection 194

Mouse Support 192

PVD API 196

Shutdown 191

SLD API 194

Trusted Path Support 155, 192

Unlock 190

- LSS API 140, 197, 281
 - Error Codes 299
 - Return Codes 299
 - SESControlKBDMonitors 281
 - SESIInactivityNotify 282
 - SESRegisterDaemon 283
 - SESReturnEventStatus 285
 - SESReturnWaitEvent 288
 - SESStartEvent 290
 - SESWaitEvent 292
- LSS event daemons 48
- LSS functions 198
- LSS keyboard/mouse device driver support 50
- LSS KPI 141, 198
- LSS policy DLLs 47
- LSS Programming Interfaces 140
- LSS Scenarios 198
 - Change Password 212
 - Create User Profile 215
 - Delete User Profile 215
 - Identification and authentication 217
 - Lock 210
 - Logoff 208
 - Logon 198
 - Process Creation 221
 - Send Security Context 220
 - Shutdown 208
 - Trusted Path 223
 - Unlock 204
- LUF 175

M

- Maximum Security Context 110
- migrate 313
- Modification 127
- module 313
- mouse 50
- Mouse Activity Detection 193
- Mouse Device Drivers 135
- Mouse Inactivity Detection 194
- Mouse Support 141, 192
- multi-tasking 313
- Multi-User Application Server 41

- Multiple concurrently active security applications 39
- Multiple concurrently active users 41
- multiplexer 313
- MVDM 154

N

- network 313
- node 313

O

- Object 14, 32, 313
- Object Reuse Protection 14
- OEM 313
- off line encryption 313
- on line encryption 313
- Operational Assurance 17
- Orange Book 10, 313
- Order of Events 154
- OS/2 Components 150
 - File System API 150
 - File System Router 150
 - I/O Device API 150
 - Loader 150
 - Per Task Data Area 150
 - Task Manager 150
 - Thread Control Block 150
- OS/2 System Calls 152
- OS/2 System Calls Supported 98
- Overview of key logon shell services components 46
- Overview of key LSS operations 51
- Overview of Keyboard/Mouse Support 192
- Overview of LSS Event Flows 190

P

- PAF 176
- panel 313
- parent process 70
- password 313
- Password Change 192
- Password validation driver 48

- Password Validation Driver API 196
- Passwords 7
- path 313
- pause function 314
- PCF 314
- pel 314
- Per Task Data Area 150
- Performance 151
- Physical Devices 7
- PIN 314
- pixel 314
- plain text 314
- pop-up 314
- POSIX 24, 35
- POSIX gid 35
- POSIX uid 35
- POSIX unmask 35
- Prerequisite Knowledge 4
- private key 314
- Privilege transition 42
- privileged 23
- Privileges and authorities 70
- process 24, 32
- Process creation 173
 - Agent Group Handle 173
 - Agent Process Handle 174
 - Agent User Handle 173
 - Authority Flags 176
 - Authority ID 174
 - Client Group Handle 173
 - Client Process Handle 174
 - Client User Handle 173
 - EGF 175
 - EPF 175
 - EUF 175
 - LUF 175
 - PAF 176
 - SECURE.SYS 174
 - Status 174
 - Status Flags 174
- process credentials 32
- Process Management 129
- Process-Status Association 108
- Process-User Association 102

- processes 24
- Programming interfaces 24, 76
- Propagate Authority Flag 116
- protocol 314
- PSC 314
- public key 314
- PVD 48, 69, 85
- PVD API 196, 279
 - PVDValidatePassword 279
- PVD Dynamic Link Library 134
- PVDValidatePassword 279

R

- RAM 314
- Reference Monitor 12
- Reserved Handles 105
- resolution 314
- Resource access control 21, 27, 87
- RESTARTUSERSHELL 51, 55, 59, 60, 62
- Return Codes 299
- reuse 314
- RGB 314
- RLA 40, 42, 74, 75, 85, 165
- RLA and CLA 74
- RLA Daemon 149
- RLA flag 119
- role 314
- ROM 314

S

- SAF 314
- SAPI 314
- SCA 70, 86, 87, 88, 89
- SCA for Audit 88
- SCA for DAC 87
- SCA for Logon 86
- SCA for Single Signon 89
- SCA for Trusted Program Support 89
- scanner 314
- Scenarios
 - A Process Sends Its Security Context to an SCA Process 184
 - A User Logs on Remotely to an Application Server 180

Scenarios *(continued)*

- An SPA Process Acts as Proxy for Its Client Processes 186
- An Untrusted Process Creates an Untrusted Child Process 183
- Change Password 212
- Create User Profile 215
- Delete User Profile 215
- File System Open Callout 159
- Identification and authentication 217
- ISS Security Kernel Initialization 158
- Lock 210
- Logoff 208
- Logon 198
- LSS 198
- Process Creation 221
- SCS 180
- Send Security Context 220
- Shutdown 208
- Trusted Path 223
- Unlock 204

schema 314

SCS 22, 28, 31, 37, 39, 161

- ACA 161
- APA 162
- CLA 163
- Initialization 171
- RLA 165
- Security Context Authority Roles 161
- SLA 166
- SPA 167
- UIA 170

SCS API 128, 178, 241

- Error Codes 299
- Return Codes 299
- SESControlProcessCreation 241
- SESCreateHandleNotify 242
- SESCreateInstanceHandle 243
- SESCreateSubjectHandle 244
- SESDeleteHandleNotify 245
- SESDeleteSubjectHandle 246
- SESKillProcess 247
- SESlogIntegrityViol 248
- SESQueryAuthorityID 248
- SESQueryContextStatus 249

SCS API *(continued)*

- SESQueryProcessInfo 250
- SESQuerySecurityContext 253
- SESQuerySubjectHandle 254
- SESQuerySubjectHandleInfo 255
- SESQuerySubjectInfo 256
- SESReleaseSubjectHandle 258
- SESReserveSubjectHandle 259
- SESResetThreadContext 260
- SESSendSecurityContext 261
- SESSetContextStatus 262
- SESSetSecurityContext 264
- SESSetSubjectHandle 266
- SESSetSubjectInfo 267

SCS Helpers 156

SCS KPI 129

SCS Programming Interfaces 128

SCS Scenarios 180

- A Process Sends Its Security Context to an SCA Process 184
- A User Logs on Remotely to an Application Server 180
- An SPA Process Acts as Proxy for Its Client Processes 186
- An Untrusted Process Creates an Untrusted Child Process 183

sealing 314

SECURE.SYS 62, 70, 125, 144, 149, 227

security 315

Security Application Components 148

- ACA Daemon 149
- APA Process 149
- CLA Daemon 149
- RLA Daemon 149
- SLA Daemon 149
- SPA Daemon 149
- UIA Daemon 149

security architecture 315

Security context 32, 70, 110, 149

- Agent Group Handle 173
- Agent Process Handle 174
- Agent User Handle 173
- Authority Flags 176
- Authority ID 174
- Client Group Handle 173

- Security context (*continued*)
 - Client Process Handle 174
 - Client User Handle 173
 - EGF 175
 - EPF 175
 - EUF 175
 - IPC 177
 - ISS Security Kernel 127
 - LUF 175
 - PAF 176
 - Process creation 173
 - SECURE.SYS 125, 174
 - SLA 125
 - Status 174
 - Status Flags 174
- Security context and trusted program support 21
- Security Context Association 100
- Security context at Process Creation 173
- Security context authorities (interoperation of) 74
- Security Context Authority Roles 161
- Security Context Creation 122
- Security Context inheritance 24
- Security Context Inherited from Parent Process 122
- Security Context Management 129, 130
- Security context services 22, 28, 31, 100, 161
- Security Context Services API 178, 241
- Security Context Status 113
- security credentials 39
- security daemon 77
- security daemons 68
- Security enabling services 21, 27, 315
- Security Event Router 98
- Security Features 85
- Security Functions 7
- Security Helpers 99
- Security Helpers for File System Services 100, 156
- Security Helpers for Security Context Services 100, 156
- Security Kernel 12, 68, 77
- Security kernel services 22, 28, 29, 96, 151
- Security Kernel Services KPI 156
- Security Kernel Services KPI Details 239
- Security officer 315
- Security Policy 10, 13
- Security policy administration tools 21
- Security relevant event interception and routing 31
- Security Requirements 7
- security server 315
- server 315
- SES 9, 21, 27
- SES and ISS communication 23
- SES API 86, 87, 88, 89, 95, 149
- SES API Audit Support 156
- SES Architecture Implementation 95
- SES Components 149
 - SECURE.SYS 149
 - Security Context 149
 - SES API 149
 - SES Daemons 149
 - SES Device Driver 149
 - SES KPI 149
 - Subject Information 149
- SES Daemons 95, 132, 149
- SES Development 81
- SES Device Driver 95, 133, 149
- SES Dynamic Link Library 134
- SES Event Management 141
- SES KPI 86, 87, 88, 89, 95, 149
- SES Overview 19, 27
- SESCtrlKBDMonitors 281
- SESCtrlProcessCreation 241
- SESCtrlHandleNotify 242
- SESCtrlInstanceHandle 243
- SESCtrlSubjectHandle 244
- SESDelHandleNotify 245
- SESDelSubjectHandle 246
- SESIactivityNotify 282
- SESKillProcess 247
- SESIlogIntegrityViol 248
- SESQueryAuthorityID 248
- SESQueryContextStatus 249
- SESQueryProcessInfo 250
- SESQuerySecurityContext 253

- SESQuerySubjectHandle 254
- SESQuerySubjectHandleInfo 255
- SESQuerySubjectInfo 256
- SESRegisterDaemon 283
- SESReleaseSubjectHandle 258
- SESReserveSubjectHandle 259
- SESResetThreadContext 260
- SESReturnEventStatus 285
- SESReturnWaitEvent 288
- SESSendSecurityContext 261
- SESSetContextStatus 262
- SESSetSecurityContext 264
- SESSetSubjectHandle 266
- SESSetSubjectInfo 267
- session 315
- session key 315
- SESStartEvent 290
- SESWaitEvent 292
- setgid 35
- setuid 35
- shutdown 44, 49, 191
- signature verification 315
- Single Signon 9, 86, 89, 315
- SKS 28, 29, 96, 151
 - Initialization 151
 - Installation 151
 - Performance 151
- SKS API 239
 - Error Codes 299
 - Return Codes 299
- SKS KPI 156
- SKS Scenarios 157
- SLA 39, 40, 49, 71, 85, 86, 125, 166
- SLA Daemon 133, 149
- SLA flag 119
- SLD 47, 69, 85
- SLD API 194, 271
 - AuthoritySource 195
 - Event status 194
 - SLDInit 271
 - SLDQueryUIA 271
- SLD Dynamic Link Library 134
- SLDInit 271
- SLDQueryUIA 271
- Smart Cards 7
- software configuration utility 315
- SPA 41, 43, 72, 85, 167
- SPA Daemon 149
- SPA flag 120
- State Flags 108, 114
 - Effective Group Flag 114
 - Effective Process Flag 114
 - Effective User Flag 115
 - Local User Flag 115
 - Propagate Authority Flag 116
- Status Flags 174
 - EGF 175
 - EPF 175
 - EUF 175
 - LUF 175
 - PAF 176
- sub-directory 315
- Subject 14, 32, 315
- Subject handle information 111
- Subject Handle Management 128, 130
- Subject handle summary 111
- Subject handles 23, 102, 110
 - Agent Group Handle 103
 - Agent Process Handle 103
 - Agent User Handle 102
 - Application Management 105
 - Client Group Handle 103
 - Client Process Handle 103
 - Client User Handle 102
 - Reserved Handles 105
- Subject Info Management 128, 130
- Subject Information 149
- subsystem 315
- system administrator 315
- System Architecture 18
- System Integrity 18
- system logon 70
- System logon authority 49
- System logon driver 47
- System Logon Driver API 194

T

- Task Manager 150
- Thread Control Block 150
- thread model 36
- threads 23, 36
- token 315
- token-ring 315
- trusted application 69
- Trusted Computing Base 9, 12
- Trusted Facility Management 18
- Trusted Path 16
- Trusted Path Support 192
- Trusted Path Support: 51
- Trusted process 41, 42
- trusted program 32, 41, 42
- Trusted Program Support 9, 89
- trusted programs 315
- Trusted Recovery 18
- trusted server process 42
- TRUSTEDPATH 51, 55, 57, 61
- TSS 315

U

- UIA 40, 47, 48, 71, 72, 85, 86, 170
- UIA Daemon 149
- UIA Daemons 133
- UIA flag 121
- umask 35
- unlock 44, 50, 51, 60, 61, 62, 190
- user 315
- user credentials 32
- user ID 315
- User identification and authentication (logon) 21
- User identification authority 48

W

- What are the typical components of an ISS 67
- What is an ISS 66
- What support does SES provide for an ISS 70
- wildcards 316
- workstation 316

X

- XGA 316



Printed in U.S.A.

S624-4668-00



Artwork Definitions

<u>id</u>	<u>File</u>	<u>Page</u>	<u>References</u>
ITSLOGO	4668SU	i	i

Table Definitions

<u>id</u>	<u>File</u>	<u>Page</u>	<u>References</u>
IATAB	4668CH8	86	86
DACTAB	4668CH8	87	87
AUDTAB	4668CH8	88	88
SINTAB	4668CH8	89	89
TRUTAB	4668CH8	89	89
SUGG	4668CH9	91	91
ROW5	4668C13A	178	178, 178
ODDROW	4668C13A	178	
HEAD5	4668C13A	178	179
XROW5	4668CH14	197	197, 197
XODDROW	4668CH14	197	197
XHEAD5	4668CH14	197	197

Figures

<u>id</u>	<u>File</u>	<u>Page</u>	<u>References</u>
8BIGPIC	4668CH4	20	1 19
8BIGP2	4668CH5	28	2 27
8SKS1	4668CH5	30	3 29
8SCS1	4668CH5	32	4
8SCS2	4668CH5	34	5
8SCS3	4668CH5	35	6
8SCS4	4668CH5	36	7
8SCS5	4668CH5	38	8
8LSS1	4668CH5		

/XRL/2

		46	9	46
8LSS2	4668CH5	54	10	53
8ISS1	4668CH6	66	11	65
8ISS2	4668CH6	67	12	67
8ISS3	4668CH6	73	13	
8ISS4	4668CH6	75	14	
8ISS6	4668CH6	77	15	
SESOVE	4668CH10	96	16	95
SKSARC	4668CH10	97	17	
SCSARC1	4668CH10	101	18	
SESARC2	4668CH10	104	19	
SCSARC3	4668CH10	107	20	
SCSARC4	4668CH10	109	21	
LSSARC1	4668CH10	132	22	
LSSARC2	4668CH10	139	23	
SESSYS	4668CH11	148	24	
SCSSYS	4668CH13	169	29	168
SCSSCN1	4668C13C	181	30	180
UPCPRO	4668C13C	183	31	183
SCSDDS	4668C13C	184	32	184
SPACP	4668C13C	186	33	186
ICISYS	4668CH15	233	34	

Headings			
<u>id</u>	<u>File</u>	<u>Page</u>	<u>References</u>
NOTICES	4668FM	xv	Special Notices ii
PT1	4668PT1	1	Part 1, Developer's Guide xvii, 81, 237
4668CH1	4668PT1	3	Chapter 1, Introduction to Security Enabling Services 4
4668CH2	4668CH2	7	Chapter 2, Security Requirements 4
CH1ORAN	4668CH2	10	2.1, Orange Book Security Criteria
CH1CONP	4668CH2	12	2.1.1, Continuous Protection
CH1TCB	4668CH2	12	2.1.1.1, Trusted Computing Base
CH1REFM	4668CH2	12	2.1.1.2, Reference Monitor and Security Kernel
4668CH4	4668CH4	19	Chapter 3, Security Enabling Services Overview 5
4668CH5	4668CH5	27	Chapter 4, Security Enabling Services 5
CH5PHM	4668CH5	33	4.3.1, Process, Handle and Thread Models
4668CH6	4668CH6	65	Chapter 5, Installable Security Subsystem 5
PT2	4668PT2	79	Part 2, Design Notes xvii, 4, 91, 237
4668CH7	4668PT2	81	Chapter 6, Introduction to SES Development 82
4668CH8	4668CH8	85	Chapter 7, Building an Installable Security Subsystem 82
CH3LOG	4668CH8	86	7.1, Logon
CH3RAC	4668CH8	87	7.2, Resource Access Control
C3AUDIT	4668CH8	88	7.3, Audit
CH3SSO	4668CH8	89	7.4, Single Signon
CH3TPS	4668CH8	89	7.5, Trusted Program Support
4668CH9	4668CH9	91	Chapter 8, Installable Security Subsystem Design Guidelines 82
4668C10	4668CH10	95	Chapter 9, SES Architecture Implementation 82
SCSHLP	4668CH10	100	9.2, Security Context Services 100
4668C11	4668CH11	147	Chapter 10, Interoperation of SES and ISS 82

4668C12	4668CH12	151	Chapter 11, Security Kernel Services (SKS) 82, 198
4668C13	4668CH13	161	Chapter 12, Security Context Services (SCS) 83, 91, 221, 223
CH8SCSA	4668CH13	161	12.1, Security Context Authority Roles
C8ACA	4668CH13	161	12.1.1, Access Control Authority (ACA)
C8APA	4668CH13	162	12.1.2, Agent Process Authority (APA)
CH8CLA	4668CH13	163	12.1.3, Client Logon Authority (CLA)
4668C14	4668CH14	189	Chapter 13, Logon Shell Services (LSS) 83, 113
4668C15	4668CH15	225	Chapter 14, Installation, Configuration, Initialization Support 83
PT3	4668PT3	235	Part 3, Appendices xvii
4668C16	4668PT3	237	Chapter 15, KPI and API Calls
SKSKAX	4668AX1	239	Appendix A, Security Kernel Services KPI Details 157, 198, 237
SCSAAX	4668AX2	241	Appendix B, Security Context Services (SCS) API Details 180, 237
SLDAAX	4668AX3	271	Appendix C, System Logon Driver API Details 195, 237
CLDAAX	4668AX3	275	Appendix D, Client Logon Driver API Details 196, 237
PVDAAX	4668AX3	279	Appendix E, Password Validation Driver API Details 196, 237
LSSAAX	4668AX3	281	Appendix F, Logon Shell Services API Details 197, 237
4668AX1	4668AX4	299	Appendix G, Security Enabling Services Error Codes 237
4668CTS	4668AX5	301	Appendix H, Customer Thoughts on Security Products 237

Tables

<u>id</u>	<u>File</u>	<u>Page</u>	<u>References</u>
CH6T1	4668CH6	68	1 68
AX2HAN	4668AX2	244	8 244
AX3T1	4668AX3	273	10 272
AX3T2	4668AX3	273	11 272

Processing Options

Runtime values:

Document fileid SG244668 SCRIPT
 Document type USERDOC
 Document style SDELIB
 Profile EDFPRF40
 Service Level 0022
 SCRIPT/VS Release 4.0.0
 Date 96.05.17
 Time 14:51:23
 Device 3820A
 Number of Passes 4
 Index YES
 SYSVAR D YES
 SYSVAR G INLINE
 SYSVAR V ITSCEVAL
 SYSVAR X YES

Formatting values used:

Annotation NO
 Cross reference listing YES
 Cross reference head prefix only NO
 Dialog LABEL
 Duplex YES
 DVCF conditions file (none)
 DVCF value 1 (none)
 DVCF value 2 (none)
 DVCF value 3 (none)
 DVCF value 4 (none)
 DVCF value 5 (none)
 DVCF value 6 (none)
 DVCF value 7 (none)
 DVCF value 8 (none)
 DVCF value 9 (none)
 Explode NO
 Figure list on new page YES
 Figure/table number separation YES
 Folio-by-chapter NO
 Head 0 body text Part
 Head 1 body text Chapter
 Head 1 appendix text Appendix
 Hyphenation NO
 Justification NO

/XRL/6

Language	ENGL
Keyboard	395
Layout	OFF
Leader dots	YES
Master index	(none)
Partial TOC (maximum level)	4
Partial TOC (new page after)	INLINE
Print example id's	NO
Print cross reference page numbers	YES
Process value	(none)
Punctuation move characters	,
Read cross-reference file	(none)
Running heading/footing rule	NONE
Show index entries	NO
Table of Contents (maximum level)	3
Table list on new page	YES
Title page (draft) alignment	RIGHT
Write cross-reference file	(none)

Imbed Trace

Page 0	4668SU
Page 0	4668VARS
Page 0	4668FM
Page i	4668EDNO
Page ii	4668ABST
Page xv	4668SPEC
Page xv	4668TMKS
Page xvi	4668PREF
Page xix	4668ACKS
Page xx	4668PT1
Page 5	4668CH2
Page 18	4668CH4
Page 25	4668CH5
Page 64	4668CH6
Page 78	4668PT2
Page 83	4668CH8
Page 90	4668CH9
Page 93	4668CH10
Page 145	4668CH11
Page 150	4668CH12
Page 159	4668CH13
Page 178	4668C13A
Page 180	4668C13C
Page 187	4668CH14
Page 223	4668CH15
Page 233	4668PT3
Page 238	4668AX1
Page 239	4668AX2
Page 269	4668AX3
Page 298	4668AX4
Page 300	4668AX5
Page 308	4668GLOS
Page 316	4668ABRV
Page 331	4668EVAL