Windows NT® Server

Server Operating System

Microsoft Security Configuration Manager for Windows NT 4

White Paper

Abstract

This paper describes Microsoft® Security Configuration Manager for Windows NT 4.0 Service Pack 4. Microsoft Security Configuration Manager is a Microsoft Management Console (MMC) snap-in tool designed to reduce costs associated with security configuration and analysis of the Windows NT® operating system. The Security Configuration Manager allows you to configure security for a Windows NT-based system, and then perform periodic analysis of the system to ensure that the configuration remains intact.

CONTENTS

INTRODUCTION

This paper describes Microsoft® Security Configuration Manager, a Microsoft Management Console (MMC) tool designed to reduce costs associated with security configuration and analysis of Windows NT-based systems.

The Microsoft Management Console is a Windows-based multiple-document interface (MDI) application that makes extensive use of Internet technologies. MMC is a core part of Microsoft's management strategy, and is designed to provide a single host for all management tools, facilitate task delegation, and lower total cost of ownership for enterprise users of the Windows® and Windows NT® operating systems. MMC itself does not supply any management behavior, but instead provides a common environment for *snap-ins*, which define the actual management behavior. Snap-ins are administrative components integrated into a common host—the MMC interface.

Security Configuration Manager is a snap-in component for MMC that is designed to provide a central repository for security-related administrative tasks. With Security Configuration Manager, you will be able to use a common tool to configure and analyze security on one or more Windows NT machines in your network.

Why Security Configuration Manager is Necessary

The current version of Microsoft Windows NT network operating system has excellent security features built into the system. A single sign-on to the Windows NT domain allows user access to resources anywhere in the corporate network. The system provides tools for security policy and account management, and the Windows NT Domain model is flexible and can support a wide range of network configurations.

From the administrator’s point of view, Windows NT provides a number of graphical tools that can be used individually to configure various aspects of system security. However, these tools are not centralized—an administrator may need to open three or four applications to configure security for one computer. Using these applications is therefore considered costly and cumbersome by many security conscious customers. In addition, security configuration can be complex.

While Windows NT provides adequate (if somewhat inconvenient) configuration tools, it lacks powerful tools for security analysis. The only tool provided that can be used to monitor security is Event Viewer, and it was not designed for performing corporate-level audit analysis. There are third-party tools for such analysis; however, those tools either lack enterprise-level features or are not comprehensive.

Security Configuration Manager is intended to answer the need for a central security configuration tool, and will provide the framework for enterprise-level analysis functionality. Most importantly, it will reduce security-related administration costs by defining a single point where the entire system’s security can be viewed, analyzed, and adjusted as necessary. The goal is to provide a *comprehensive, flexible, extensible and simple* tool for configuring and analyzing system security.

Security Configuration Manager Design Goals

The process of configuring security in a Windows NT-based network can be complex and detailed in terms of the system components involved and the level of change that may be required. Therefore, Security Configuration Manager is designed to allow you to perform *configuration at a macro level*. In other words, Security Configuration Manager allows you to define a number of configuration settings and have them applied as one. With this tool, configuration tasks can be grouped and automated; they no longer require numerous, iterative key presses and repeat visits to a number of different applications to configure a group of machines.

Note that Security Configuration Manager is not designed to replace system tools that address different aspects of system security—such as User Manager, Server Manager, Access Control List (ACL) Editor, and so forth. Rather its goal is to complement them by defining an engine that can interpret a standard configuration file and perform the required operations automatically. Administrators can continue to use existing tools to change individual security settings whenever necessary.

To address the security analysis gap in security administration in Windows NT, Security Configuration Manager provides *analysis at a micro level.* All security relevant system parameters which can be configured, can also be analyzed for deviations from some baseline configuration.

Security Configuration Manager Features

Security Configuration Manager is designed to be comprehensive, flexible, extendible, and simple.

Comprehensiveness

Unlike other operating system features, security is a characteristic of the system as a whole. Almost every component of the system is responsible for some aspect of system security. Therefore, questions such as “Is my computer secure?” or “Is my network secure?” are extremely difficult to answer. Typically, a system administrator must examine many different system components and use many tools in an attempt to answer these questions. Microsoft’s goal is to have Security Configuration Manager be the resource for answering security-related questions, whether they are general (such as those listed above) or very specific. To provide comprehensive security administration and information, Security Configuration Manager allows you to configure and analyze all of the following:

- **Account Policies** – You can use the tool to set access policy, including domain or local password policies and domain or local account lockout policies.
- **Local Policies** – You can configure local audit policy, user rights assignment and various security relevant system parameters which were previously managed by locating and setting certain registry values.
- **Restricted Groups** – You can control group memberships for built-in groups such as Administrators, Server Operators, Backup Operators, Power Users, and so forth, as well any other specific group that you would like to configure.

This should not be used as a general membership management tool—only to control membership of specific groups that have sensitive capabilities assigned to them.

- **System Services** – You can configure startup and security aspects for the different services installed on a system, such as Alerter, Messenger and so forth.
- **System Registry** – You can use the tool set to set the security on system registry keys.
- **System Store** – You can use the tool set to set the security for local file system objects.

Flexibility

Security Configuration Manager allows you to create and edit Security Configuration Files that contain settings for each of the security areas outlined above. These configuration files are text based files which can be easily distributed with tools such as Microsoft Systems Management Server to configure or analyze system security.

Security Configuration Manager also includes a set of predefined security configuration files which can be customized for your environment. These predefined security configuration files define three levels of security beyond the default “out of box” security settings.

The architecture is sufficiently flexible to support new security areas as the system evolves.

Extendibility

Security Configuration Manager is architected to be extendible. You can add extensions as new areas of security configuration, or as new attributes within an existing area. Since the configuration information is stored in a standard .inf file format, it can be easily extended without affecting backward compatibility.

Additionally, *system services* is a currently defined area that has been architected to be extendible within itself. It permits any service writer to implement a *Security Configuration Attachment* that can configure security settings for a particular system service, as well as perform any analysis that may be required. Different Windows NT-based systems can be configured to run different sets of services. Also, Microsoft expects that independent software vendors (ISVs) who develop services will want to add their service’s security configuration and analysis to this overall security framework.

Simplicity

Because *Security Configuration Manager* is designed to reduce costs associated with administering a network, it is vital that the tool be easy to learn and use. Security Configuration Manager contains no complicated options—only a simple uniform graphical user interface (GUI) for defining configuration files and viewing security analysis data. The interface uses the standardized context menus and views supported by Microsoft Management Console. There are no superfluous graphics or statistics, only a simple tabular view of the information with visual cues to flag security problems. In addition, Security Configuration Manager contains a command-line utility to allow administrators to run configuration and analysis as part of a script. Either the command line tool or the GUI can be used to perform a configuration or an analysis, although the GUI is needed to edit configuration files and view analysis results graphically.

The next section of this document provides a more in-depth overview of the *Security Configuration Manager*, its architecture, and how it fits into Windows NT.

The primary objective of Security Configuration Manager is to make it easier for customers to secure their Windows NT-based systems. Security Configuration Manager accomplishes this by allowing administrators to define all security relevant system parameters in a single location. Once a security configuration has been defined, the tool can be used to apply that configuration and detect deviations from that configuration. As mentioned previously, Security Configuration Manager also includes several predefined security configuration files which can be customized for site specific security and application requirements.

Security Configuration Areas

Security configuration for a system is subdivided into *security areas*. Microsoft has identified several security areas; however, new areas can be added in the future to support enhanced system functionality without breaking backward compatibility with existing configuration files. The currently supported security areas are:

Area	*Configurable Items*
Account Policies	- Password Policy - Lockout Policy
Local Policies	- Audit Policy - User Rights and Privilege Assignment - Security Options (Registry Values)
Event Log	- Settings for System, Application, and Security Logs
Restricted Groups	- Group membership
System Services	- Startup Modes and Access Control Lists for all system services
Registry	- Access Control Lists for Registry Keys
File System	- Access Control Lists for Folders and Files

Figure 1: Security Configuration areas and the types of items which are configurable in each area

Security Configuration Manager User Interface

The *Security Configuration Manager* GUI is provided as a Microsoft Management Console (MMC) snap-in. The graphical interface supports the following administrative functions:

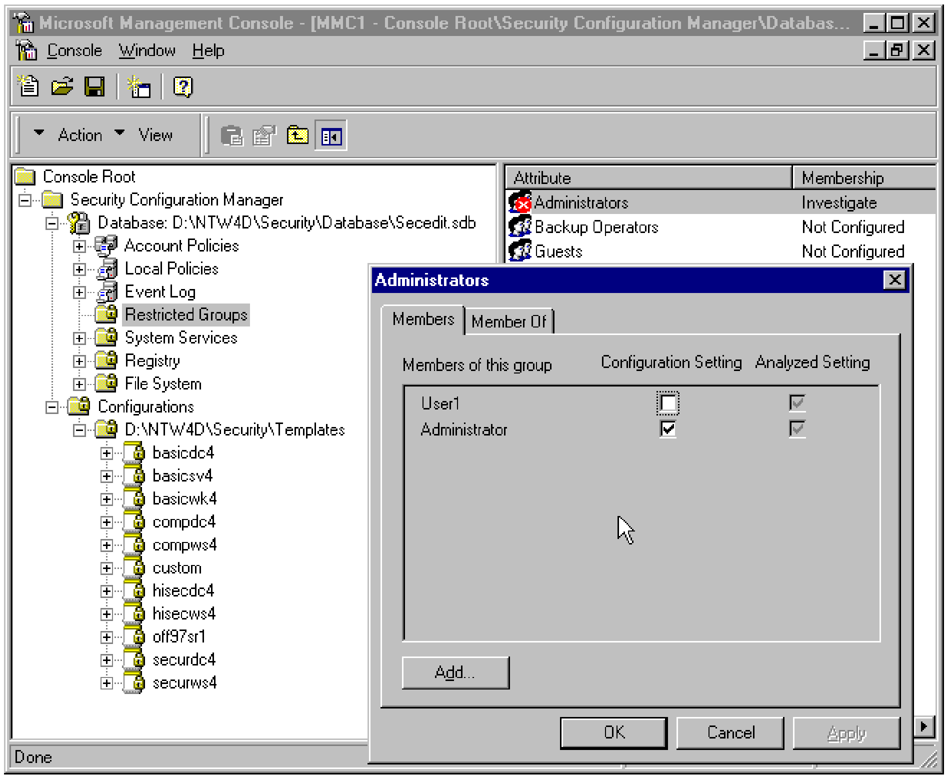
- ***Defining Security Configuration files***—The tool includes a GUI-based editor that enumerates all of the security areas described above and allows the administrator to define security settings for each parameter in each area. The configuration files are ultimately saved as text-based .inf files.
- ***Configuring system security***—Configuration operations are ultimately performed using a database. To configure a Windows NT-based system, use the Security Configuration Manager context menus to:
  - Select a Database
  - Import configuration file(s) as necessary
  - Configure the system

  Import operations can append to or overwrite database information that has been previously imported. Appending (which is the default) allows different configuration files to be combined into a single database for configuration.
- ***Analyzing system's security***— Similarly, all analysis operations are performed against a database. To analyze a system's security, use the context menus to:
  - Select a Database
  - Import configuration file(s) as necessary
  - Analyze the system

  The configuration file(s) that have been imported into the database define the *baseline* for the analysis.
- ***View Security Analysis data***—Analysis results are stored back into the same database that contains the baseline configuration information. The baseline settings are presented alongside the current system settings, and color, fonts, and icons are used to highlight differences between the baseline configuration and the actual system settings. If desired, you can modify the baseline configuration in lieu of the analysis results. The modified configuration information may also be exported into a configuration file for subsequent use.

Graphical User Interface

Figure 2 shows the GUI after an analysis has been performed against a database named secedit.sdb. Before performing the analysis, configuration file information would have been imported into the database to define the baseline for the analysis:



Attribute	Membership
Administrators	Investigate
Backup Operators	Not Configured
Guests	Not Configured

Members of this group	Configuration Setting	Analyzed Setting
User1	☐	✓
Administrator	✓	✓

Figure 2. Security Configuration Manager *Snap-in Graphical User Interface*

Highlighted is the fact that membership of the administrators group on the system is different from the membership defined in the baseline configuration. Investigating further reveals that the baseline configuration suggests that only the administrator should be a member of the administrators group, while the actual system settings includes User1 in the administrators group. If desired, the baseline configuration can be updated to include User1, or the system can be reconfigured to remove User1 from the administrators group.

This snapshot also reveals the predefined configuration files that are included with the Security Configuration Manager. These are listed under the *Configurations* node from whence they can be edited.

Secedit Command Line Tool

Security Configuration Manager also includes a command line tool (secedit.exe) for applying configuration files and performing analyses. Typing secedit with no command line arguments will expose the syntax for the command line tool. As an example:

```
secedit /configure /cfg securws4.inf /db secedit.sdb /areas REGKEYS FILESTORE
```

- Imports the securws4.inf configuration file into the secedit.sdb database
- Applies only the file system and registry security settings specified in the

securws4.inf configuration file to the Windows NT-based system where the program is run.

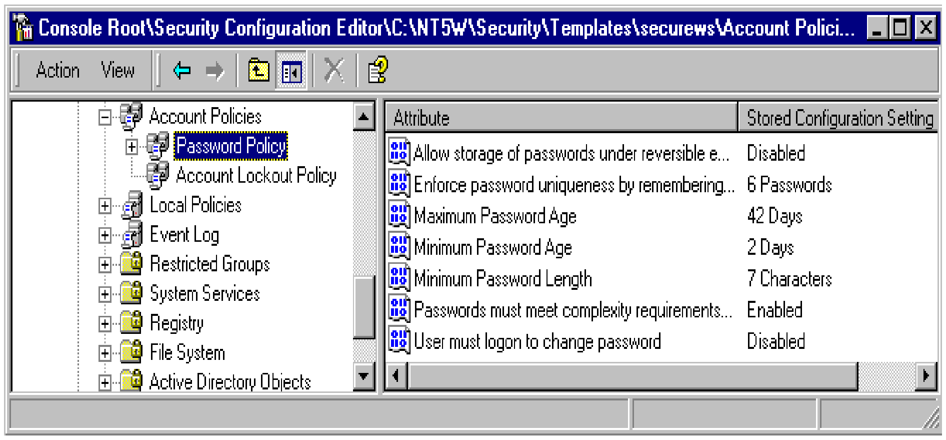
The command line tool also supports a quiet mode of operation and is useful for applying configuration files to many systems using distributed systems management tools such as Microsoft Systems Management Server.

Note that the GUI configures all security areas, while the command line tool is capable of configuring specific security areas.

This section describes how to use the Security Configuration Manager to configure various security aspects of a Windows NT 4-based system. Note that this tool relies entirely on the security features that are already in Windows NT—it does not alter the security capabilities of the system.

Account Policies

The Account Policies security area contains **Password** and **Lockout** Policy settings normally configured through the user manager:



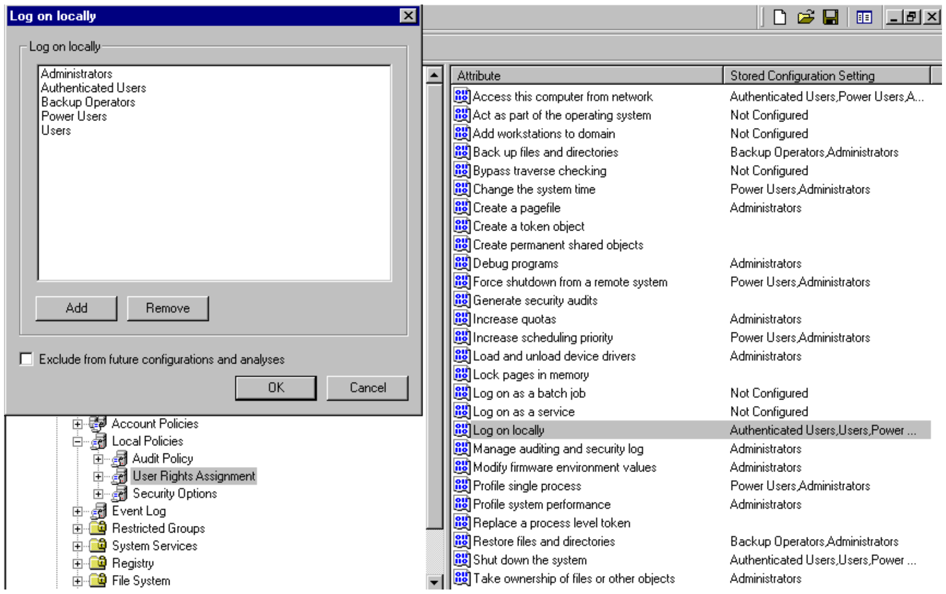
Attribute	Stored Configuration Setting
Allow storage of passwords under reversible e...	Disabled
Enforce password uniqueness by remembering...	6 Passwords
Maximum Password Age	42 Days
Minimum Password Age	2 Days
Minimum Password Length	7 Characters
Passwords must meet complexity requirements...	Enabled
User must logon to change password	Disabled

Figure 3: Account Policy Security Area

Note that configuring a Domain Controller’s Account policy will impact all Domain Controllers as password and lockout policy is a domain-wide setting enforced by all Domain Controllers. Configuring Password and Lockout policy on a Workstation or Server impacts only the local password and lockout policy for that workstation or server.

Local Policies

Local Policies also includes policy settings that are typically managed from the User Manager. These include **Audit** policies such as *Audit File and Object Access* as well as **User Rights** policies such as *Access this Computer from the Network* or *Log on Locally*. In the case of User Rights, Security Configuration Manager allows the administrator to specify for each user right, exactly which users, local groups or global groups should be granted that right:



Administrators
Authenticated Users
Backup Operators
Power Users
Users

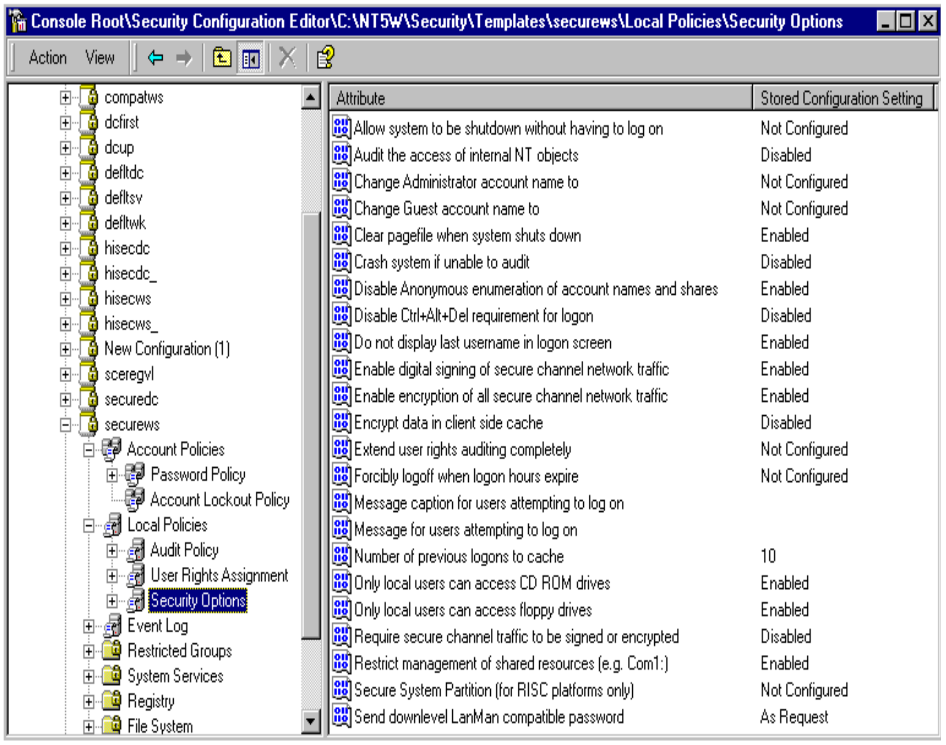
☐ Exclude from future configurations and analyses

Attribute	Stored Configuration Setting
Access this computer from network	Authenticated Users,Power Users,A...
Act as part of the operating system	Not Configured
Add workstations to domain	Not Configured
Back up files and directories	Backup Operators,Administrators
Bypass traverse checking	Not Configured
Change the system time	Power Users,Administrators
Create a pagefile	Administrators
Create a token object	
Create permanent shared objects	
Debug programs	Administrators
Force shutdown from a remote system	Power Users,Administrators
Generate security audits	
Increase quotas	Administrators
Increase scheduling priority	Power Users,Administrators
Load and unload device drivers	Administrators
Lock pages in memory	
Log on as a batch job	Not Configured
Log on as a service	Not Configured
Log on locally	Authenticated Users,Users,Power ...
Manage auditing and security log	Administrators
Modify firmware environment values	Administrators
Profile single process	Power Users,Administrators
Profile system performance	Administrators
Replace a process level token	
Restore files and directories	Backup Operators,Administrators
Shut down the system	Authenticated Users,Users,Power ...
Take ownership of files or other objects	Administrators

Figure 4: Local Policies—User Rights and Privileges

When the system is configured, Security Configuration Manager ensures that each user or group has only the rights defined in the configuration file.

Additionally, Local Policies also allows the administrator to configure **Security Options** which consist of well known security relevant system parameters, many of which are, normally configured by setting registry values using tools like Regedt32.exe:



Attribute	Stored Configuration Setting
Allow system to be shutdown without having to log on	Not Configured
Audit the access of internal NT objects	Disabled
Change Administrator account name to	Not Configured
Change Guest account name to	Not Configured
Clear pagefile when system shuts down	Enabled
Crash system if unable to audit	Disabled
Disable Anonymous enumeration of account names and shares	Enabled
Disable Ctrl+Alt+Del requirement for logon	Disabled
Do not display last username in logon screen	Enabled
Enable digital signing of secure channel network traffic	Enabled
Enable encryption of all secure channel network traffic	Enabled
Encrypt data in client side cache	Disabled
Extend user rights auditing completely	Not Configured
Forcibly logoff when logon hours expire	Not Configured
Message caption for users attempting to log on	
Message for users attempting to log on	
Number of previous logons to cache	10
Only local users can access CD ROM drives	Enabled
Only local users can access floppy drives	Enabled
Require secure channel traffic to be signed or encrypted	Disabled
Restrict management of shared resources (e.g. Com1:)	Enabled
Secure System Partition (for RISC platforms only)	Not Configured
Send downlevel LanMan compatible password	As Request

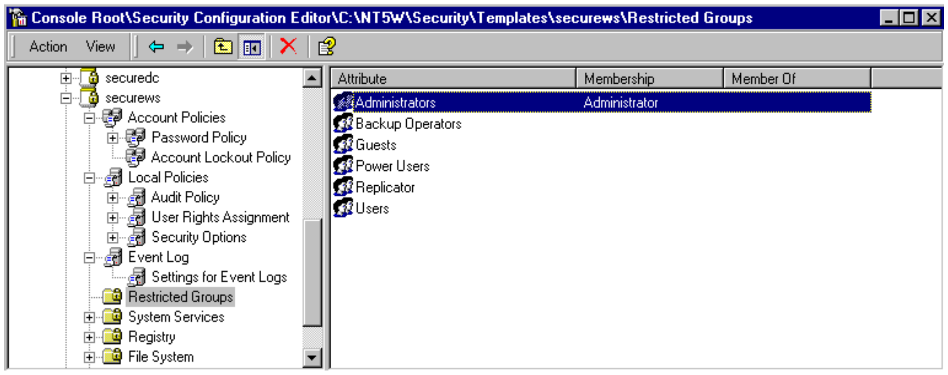
Figure 5: Local Policies—Security Options.

Restricted Group Management

Restricted Group Management allows you to manage the membership of built-in groups that have certain predefined capabilities These groups include local groups such as Administrators, Power Users, Print Operators, Server Operators, and so forth, as well as global groups such as Domain Administrators.

You can also add groups that you consider sensitive and/or privileged to the Restricted Group Management list, along with their membership information. Note that restricted groups is not intended to define all group memberships, just those that are considered to be security sensitive.

In addition to group membership, the area tracks and controls reverse membership of each restricted group in the **MemberOf** column. This column shows other groups to which the restricted group can belong. For example, you might say that Domain Temps can only be a *member of* the local Users group. The *member of* feature is extremely beneficial on Windows NT 5-based systems which support nested groups and allows administrators to prevent a specific group from achieving privileges through various levels of inadvertent nesting.


Console Root\Security Configuration Editor\C:\NT5W\Security\Templates\securews\Restricted Groups

Action View

- securedc
- securews
  - Account Policies
    - Password Policy
    - Account Lockout Policy
  - Local Policies
    - Audit Policy
    - User Rights Assignment
    - Security Options
  - Event Log
    - Settings for Event Logs
  - Restricted Groups
  - System Services
  - Registry
  - File System

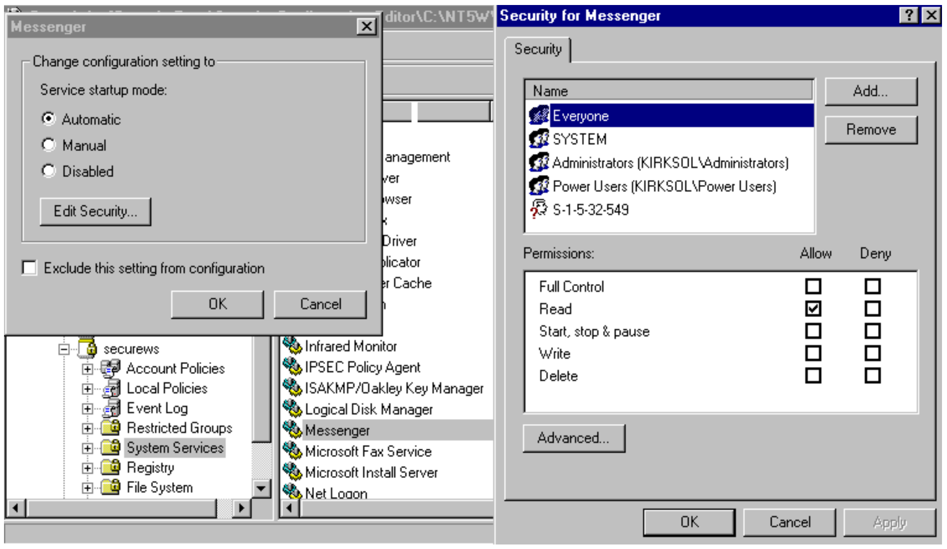
Attribute	Membership	Member Of
Administrators	Administrator	
Backup Operators		
Guests		
Power Users		
Replicator		
Users		

Figure 6. Configuring Restricted Groups

The configuration engine ensures that group memberships are set as specified in the configuration file. Groups and users not specified in the configuration are removed from the restricted group. Groups and users specified in the configuration are added if they do not exist. In addition, the reverse membership configuration option ensures that each restricted group is a member of only those groups specified in the **MemberOf** column.

System Services Security

System Services include critical functionality such as network, file, and print services. Security Configuration Manager directly supports general settings for each system service. These general settings include the service startup settings and security on the service itself—for example, the ability to Start, Stop and Pause a service. Service security has always been supported in Windows NT 4, however there has not previously been a UI for configuring service security:


Messenger

Change configuration setting to

Service startup mode:

(● Automatic) ○ Manual ○ Disabled

Edit Security...

☐ Exclude this setting from configuration

OK Cancel

- securews
  - Account Policies
  - Local Policies
  - Event Log
  - Restricted Groups
  - System Services
  - Registry
  - File System

Infrared Monitor
IPSEC Policy Agent
ISAKMP/Oakley Key Manager
Logical Disk Manager
Messenger
Microsoft Fax Service
Microsoft Install Server
Net Logon

Security for Messenger

Security

Name
- Everyone
- SYSTEM
- Administrators (KIRKSOL\Administrators)
- Power Users (KIRKSOL\Power Users)
- S-1-5-32-549

Add... Remove

Permissions:	Allow	Deny
Full Control	☐	☐
Read	☑	☐
Start, stop & pause	☐	☐
Write	☐	☐
Delete	☐	☐

Advanced...

OK Cancel Apply

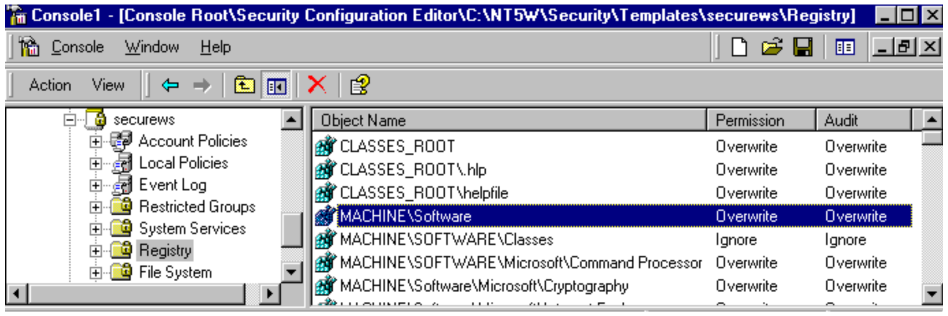
Figure 7. Configuring System Services Security

Because of the breadth and diversity of this area, *Security Configuration Manager*'s System Services area is architected to be extendable.

To extend this area of the Security Configuration Manager so that it can be used to configure specific settings for a new service, you need to create and attach a Security Configuration Manager Attachment (a DLL) as described in Appendix A.

File and Registry Security

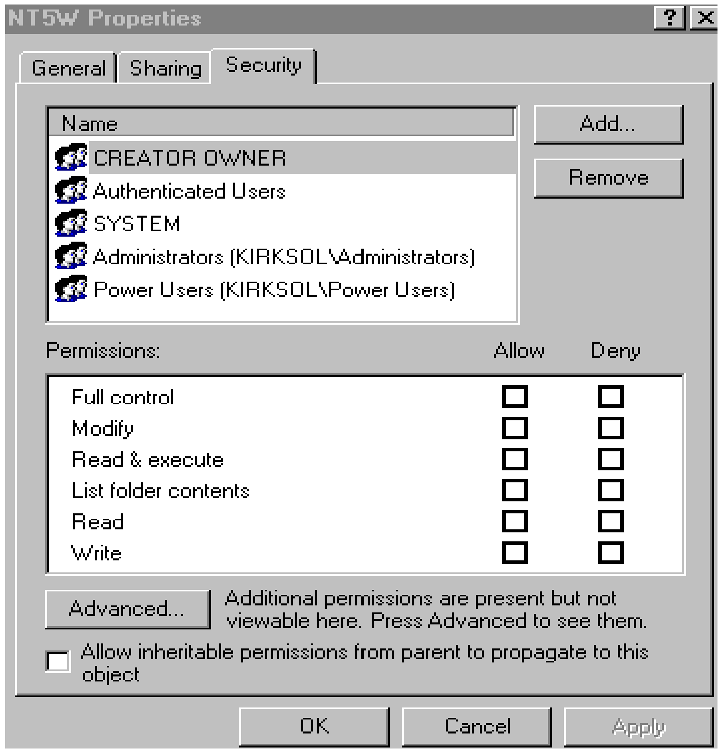
Access Control settings for File System and Registry Objects are typically configured using the ACL Editor in Windows NT Explorer and Regedt32.exe respectively. Security Configuration Manager also incorporates these important aspects of system security into the UI:



Object Name	Permission	Audit
CLASSES_ROOT	Overwrite	Overwrite
CLASSES_ROOT\.hlp	Overwrite	Overwrite
CLASSES_ROOT\helpfile	Overwrite	Overwrite
MACHINE\Software	Overwrite	Overwrite
MACHINE\SOFTWARE\Classes	Ignore	Ignore
MACHINE\SOFTWARE\Microsoft\Command Processor	Overwrite	Overwrite
MACHINE\Software\Microsoft\Cryptography	Overwrite	Overwrite

Figure 8: Configuring Registry ACLs

Since Security Configuration Manager was originally designed for Windows NT 5 which supports a dynamic ACL inheritance model, the Windows NT 5 ACL Inheritance infrastructure was also back-ported to Windows NT 4 and is available when the Security Configuration Manager is installed. This is most readily apparent when setting security through Windows NT Explorer or Security Configuration Manager after Security Configuration Manager is installed:



NT5W Properties

General | Sharing | Security

Name
CREATOR OWNER
Authenticated Users
SYSTEM
Administrators (KIRKSOL\Administrators)
Power Users (KIRKSOL\Power Users)

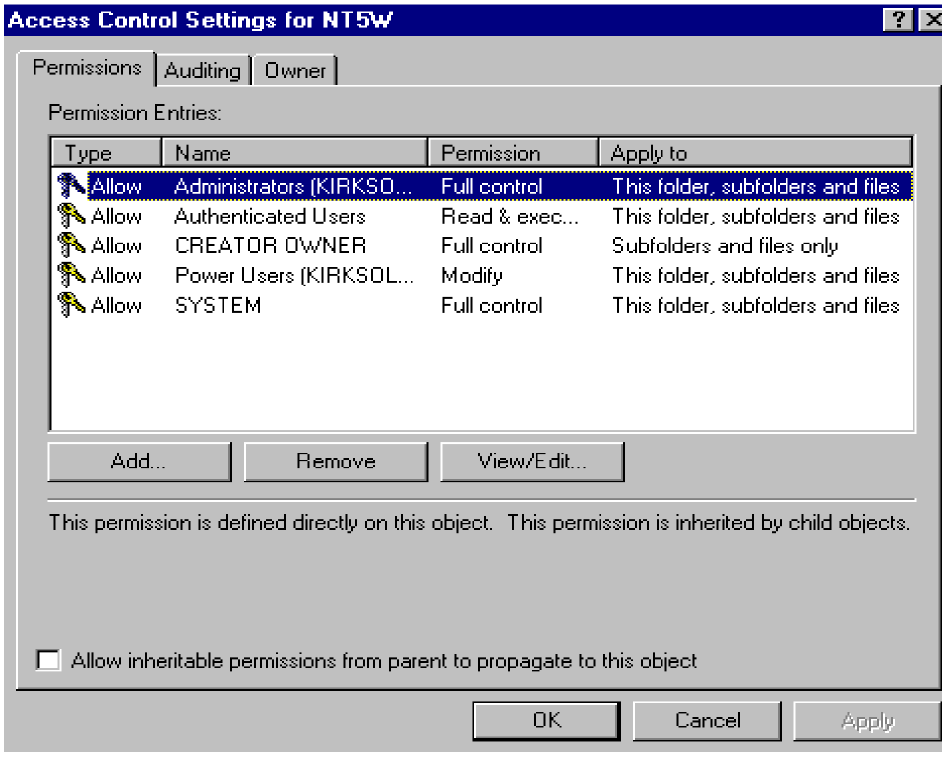
Permissions:	Allow	Deny
Full control	☐	☐
Modify	☐	☐
Read & execute	☐	☐
List folder contents	☐	☐
Read	☐	☐
Write	☐	☐

Additional permissions are present but not viewable here. Press Advanced to see them.

☐ Allow inheritable permissions from parent to propagate to this object

Figure9: The new ACL Editor

Clicking on the advanced tab in the new ACL Editor reveals the following dialog:


Access Control Settings for NT5W

Permission Entries:

Type	Name	Permission	Apply to
Allow	Administrators (KIRKSO...	Full control	This folder, subfolders and files
Allow	Authenticated Users	Read & exec...	This folder, subfolders and files
Allow	CREATOR OWNER	Full control	Subfolders and files only
Allow	Power Users (KIRKSOL...	Modify	This folder, subfolders and files
Allow	SYSTEM	Full control	This folder, subfolders and files

This permission is defined directly on this object. This permission is inherited by child objects.

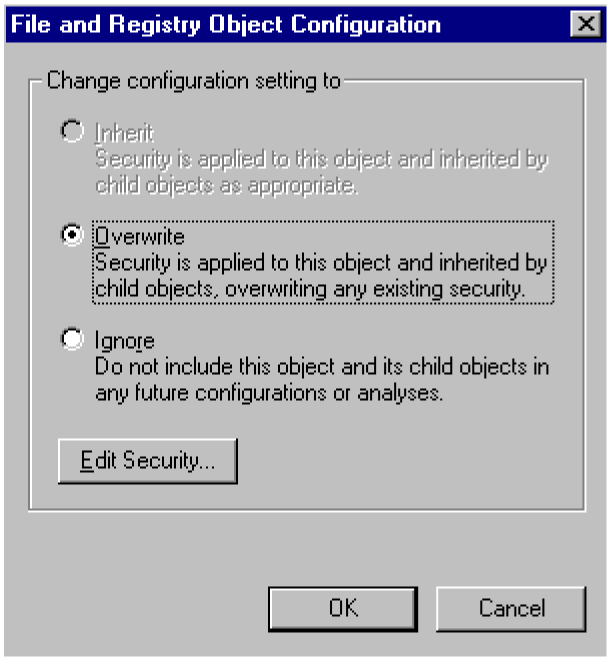
☐ Allow inheritable permissions from parent to propagate to this object

Figure 10: Advanced Tab of the new ACL Editor

Notable in these dialogs is the ability to control whether or not permissions for a given object are inherited from the parent object or not. This is specified by the checkbox labeled *Allow inheritable permissions from parent to propagate to this object*. If an object (for example, a Folder) does not allow inheritable permissions from parent to propagate, it is said to be protected, and changes to it's parent do not impact the object itself. However, if the object does allow inheritable permissions to propagate, changes to inheritable permissions defined on its parent will automatically (dynamically) be propagated by the system.

The degree to which a permission is inheritable is defined by the Apply To dialog. For example, the Full Control permission for Administrators applies to This Folder, Subfolders, and Files. When this permission is applied, the ACL inheritance model will make sure that Administrators have Full Control on This Folder, and all Subfolders and Files which are not protected (that is which Allow inheritable permissions to propagate).

As shown in Figure 11 below, Security Configuration Manager allows administrators to override the normal behavior of the ACL inheritance model by specifying that all child objects of a given object should be reconfigured whether they are protected (do not allow inheritable permissions to propagate) or not. In fact, this is the only mode of operation that is supported in the Windows NT 4 version of Security Configuration Manager as it is consistent with the “Replace Permissions on Subdirectories and Files” check boxes in the original Windows NT 4 ACL Editor.



File and Registry Object Configuration

Change configuration setting to

( ) Inherit
Security is applied to this object and inherited by child objects as appropriate.

(●) Overwrite
Security is applied to this object and inherited by child objects, overwriting any existing security.

( ) Ignore
Do not include this object and its child objects in any future configurations or analyses.

Figure 11: Configuration Modes

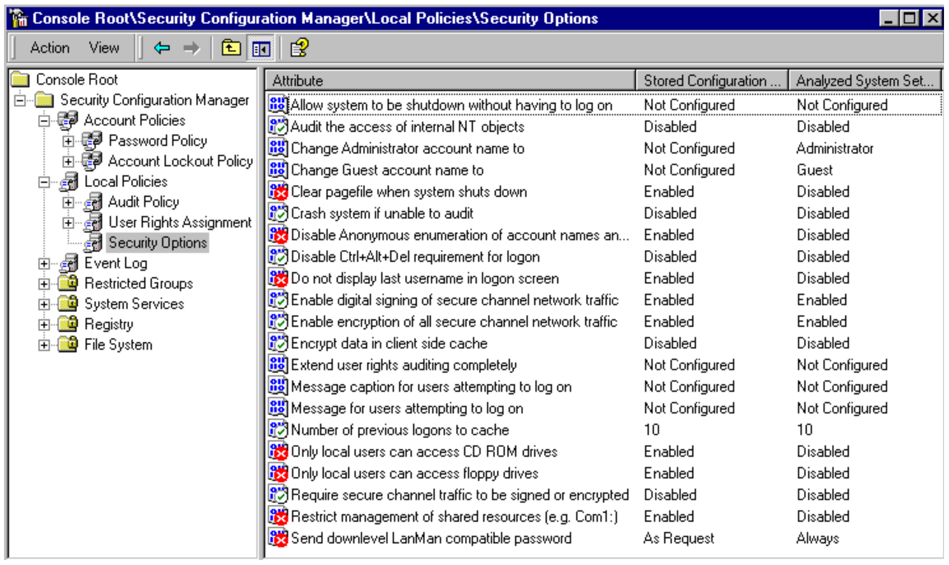
If there is a child object which should not be reconfigured by Security Configuration Manager, then that child object should be included in the configuration file and marked with Ignore.

Environment Variables for File System Security

In addition to the comments above which are applicable to both File System and Registry Objects, Security Configuration Manager also supports the use of environment variables for File System objects. Use of environment variables can make configuration files portable across systems which have installed Windows NT on different volumes.

This section describes how to use the Security Configuration Manager to analyze various security aspects of a Windows NT 4-based system. Security analysis is the other half of Security Configuration Manager's capabilities. Security Configuration Manager provides a graphical interface which allows you to view the analysis information collected from the system. Additionally, you can use the graphical user interface or the command-line utility, **secedit**, to collect analysis data from the system. This enables you to collect the data interactively or to schedule data collection as part of an off-hour batch processing script.

To promote ease of use (and to eliminate the steep learning curve normally associated with new administration tools), the GUI design of the analysis display has been kept simple and informational. Instead of complicated graphics or error alerts, it provides simple visual cues (icons and color) to identify differences between a provided baseline configuration and actual system settings:


Console Root\Security Configuration Manager\Local Policies\Security Options
Action
View

Console Root
- Security Configuration Manager
  - Account Policies
    - Password Policy
    - Account Lockout Policy
  - Local Policies
    - Audit Policy
    - User Rights Assignment
    - Security Options
  - Event Log
  - Restricted Groups
  - System Services
  - Registry
  - File System

Attribute	Stored Configuration ...	Analyzed System Set...
Allow system to be shutdown without having to log on	Not Configured	Not Configured
Audit the access of internal NT objects	Disabled	Disabled
Change Administrator account name to	Not Configured	Administrator
Change Guest account name to	Not Configured	Guest
Clear pagefile when system shuts down	Enabled	Disabled
Crash system if unable to audit	Disabled	Disabled
Disable Anonymous enumeration of account names an...	Enabled	Disabled
Disable Ctrl+Alt+Del requirement for logon	Disabled	Disabled
Do not display last username in logon screen	Enabled	Disabled
Enable digital signing of secure channel network traffic	Enabled	Enabled
Enable encryption of all secure channel network traffic	Enabled	Enabled
Encrypt data in client side cache	Disabled	Disabled
Extend user rights auditing completely	Not Configured	Not Configured
Message caption for users attempting to log on	Not Configured	Not Configured
Message for users attempting to log on	Not Configured	Not Configured
Number of previous logons to cache	10	10
Only local users can access CD ROM drives	Enabled	Disabled
Only local users can access floppy drives	Enabled	Disabled
Require secure channel traffic to be signed or encrypted	Disabled	Disabled
Restrict management of shared resources (e.g. Com1:)	Enabled	Disabled
Send downlevel LanMan compatible password	As Request	Always

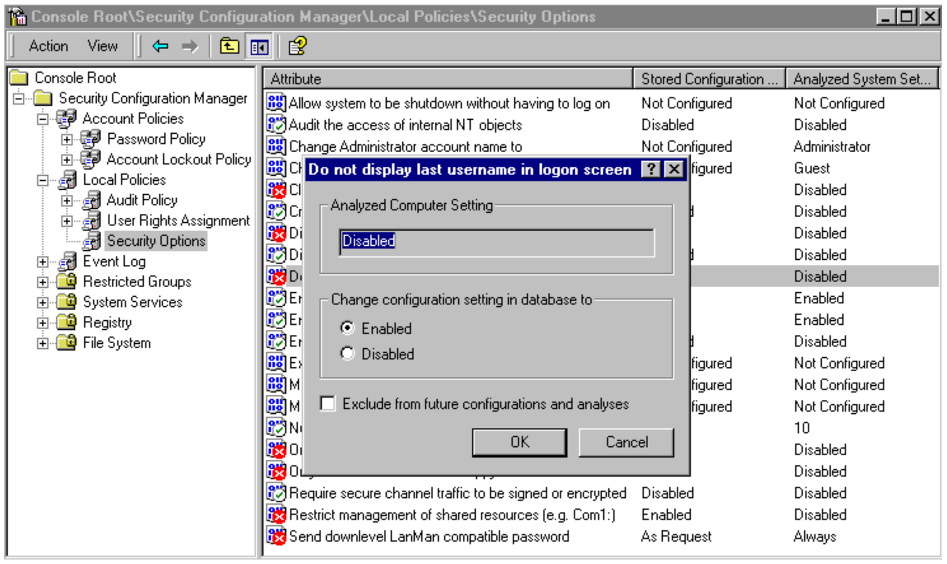
Figure 12. Security Analysis User Interface

The Security Configuration Manager supports two modes of security analysis for Windows NT-based systems: *configured system analysis* and *unconfigured system analysis*.

- ***Configured system analysis*** refers to situations where the system has already been configured using a security configuration file prior to performing the analysis. In this case, the baseline configuration has already been imported into a database and an analysis can be performed against that same database. This type of analysis can be used to answer the question: What security relevant system parameters have changed since the last time this machine was configured?
- ***Unconfigured system analysis*** refers to situations where the system has not been configured with the baseline configuration. This type of analysis can be used to answer questions such as, How do current system settings compare with this baseline configuration? What system settings would change if I were to apply this configuration? In this case, the baseline security configuration file is imported into a database prior to performing the analysis. If you later want to configure the system with the baseline configuration, the created database can be used.

In both cases, the end result is a database that contains both configuration information as well as analysis results.

In addition to analyzing the system's current configuration, the Security Configuration analysis interface allows interactive changes to the baseline configuration stored in the database. To modify the baseline configuration stored in the database, double click on the mismatched item in the result pane:


Console Root\Security Configuration Manager\Local Policies\Security Options

Action View

- Console Root
  - Security Configuration Manager
    - Account Policies
      - Password Policy
      - Account Lockout Policy
    - Local Policies
      - Audit Policy
      - User Rights Assignment
      - Security Options
    - Event Log
    - Restricted Groups
    - System Services
    - Registry
    - File System

Attribute	Stored Configuration ...	Analyzed System Set...
Allow system to be shutdown without having to log on	Not Configured	Not Configured
Audit the access of internal NT objects	Disabled	Disabled
Change Administrator account name to	Not Configured	Administrator
Ch	figured	Guest
Cl		Disabled
Cr	d	Disabled
Di		Disabled
Di	d	Disabled
D		Disabled
Er		Enabled
Er		Enabled
Er	d	Disabled
Ex	figured	Not Configured
M	figured	Not Configured
M	figured	Not Configured
N		10
O		Disabled
O		Disabled
Require secure channel traffic to be signed or encrypted	Disabled	Disabled
Restrict management of shared resources (e.g. Com1:)	Enabled	Disabled
Send downlevel LanMan compatible password	As Request	Always

Do not display last username in logon screen

Analyzed Computer Setting: Disabled

Change configuration setting in database to: [✓ Enabled] Disabled

☐ Exclude from future configurations and analyses

OK Cancel

Figure 13. Modifying a baseline configuration

This feature is provided to update the baseline configuration after you have investigated a mismatch and consider the actual system setting to be reasonable. In the example above, the stored configuration may be modified so that *the Do not display last username in logon screen* parameter is disabled (like the actual system setting) or so that it is Excluded (that is, not configured at all). If the system is reconfigured using this database, the actual system setting will be retained. If the system is reanalyzed using this database, this security parameter will no longer be flagged.

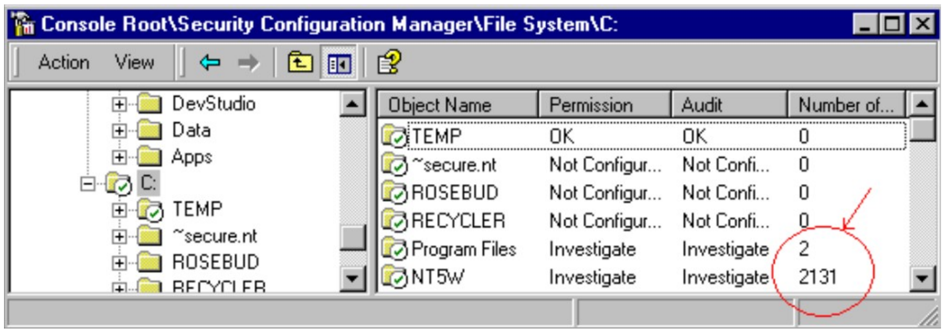
Since interactive updates of analysis information affect only the database, an export option is available which creates a security configuration file from the configuration information stored in a database.

Analyzing File System and Registry Security

Analyzing parameters in most security areas is straightforward because only the security on that specific object needs to be considered. Either the analyzed setting matches the baseline configuration or it doesn't. However tree based objects such as file system and registry objects for which an inheritance mechanism exists, the security of child objects must also be considered. In the case of Registry and File System objects, Security Configuration Manager adds an additional column which indicates whether or not child objects should be further investigated:

FOR MORE INFORMATION

APPENDIX A. IMPLEMENTING SERVICE SECURITY ATTACHMENTS



Object Name	Permission	Audit	Number of...
TEMP	OK	OK	0
~secure.nt	Not Configur...	Not Confi...	0
ROSEBUD	Not Configur...	Not Confi...	0
RECYCLER	Not Configur...	Not Confi...	0
Program Files	Investigate	Investigate	2
NT5W	Investigate	Investigate	2131

Figure 14. Reviewing security of child objects

In the above figure, we see that the security descriptors for the Program Files and Windows NT 5 Workstation folder objects are themselves OK, however there are child objects which may or may not be mismatched. The numbers indicate how many child objects exist. To further explore these child objects, expand the corresponding node in the scope pane on the left hand side.

For the latest information on Windows NT Server, check out our World Wide Web site at http://www.microsoft.com/security/ntprod.htm.

This appendix describes the procedures for building and implementing Service Security Attachments for the Security Configuration Tool Set.

Introduction

Security Configuration Tool Set handles general security settings for individual services directly. These general settings include the service invocation policy (disabled, automatic, or manual) as well as the security descriptors for each service. Therefore, *no Security Configuration Attachment should attempt to configure these settings*. The Service Security Attachment architecture within the Security Configuration tool set provides an infrastructure to configure and analyze service-specific security settings for individual services. For example, Spooler is a Windows NT service that defines *private objects* (in this case, printers) that need to be secured. In addition, it has configuration parameters that are security-sensitive. For Spooler, a Service Security Attachment must allow configuration and analysis of security settings on printer objects and security-sensitive parameters for the service.

Service Security Attachment architecture requires implementation of the following two pieces:

- An attachment engine DLL that implements three interfaces (described later in this appendix).
- A Microsoft Management Console (MMC) extension snap-in that provides the

configuration editor and manager functionality used to configure and analyze specific security settings for the service. This extension is exposed as a node under the **Services** security area in the Security Configuration Editor and Manager snap-ins.

Security Configuration tool set provides a set of support APIs that the attachment engine or extension snap-in can use to query or set service-specific information contained in the security configuration and analyses database.

Architecture

Figure 1A shows the three pieces of the snap-in architecture where the attachment infrastructure fits in (the extension snap-ins for attachments, the service attachment engines, and the inspection database).

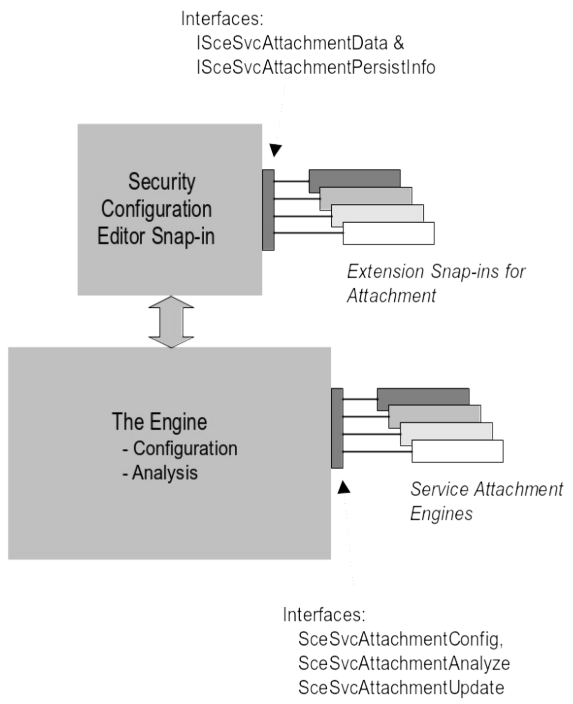


Figure 1A. Security Configuration tool set Snap-in, Engine, and Extension Architecture

Security Configuration tool set, which consists of the main engine and the MMC snap-in, provides the overall framework for configuring and analyzing system security for Windows NT installations.

In the attachment framework, attachment engine DLLs register with Security Configuration engine. Security Configuration engine service then loads the attachment during execution. It calls the attachment's configuration interface when the system is configured, the analysis interface when the system is analyzed, and the update interface when parameters in the database are modified by the extension snap-in.

Similarly, the extension snap-in for the service attachment must register as a Security Configuration Tool Set snap-in extension. The Security Configuration tool set snap-ins loads the extension snap-in as a node under the Services security area in both the editing and manager tools. If you are writing an extension snap-in, you must follow the MMC extension writer's documentation to implement the overall extension snap-in. In addition, you must implement the component object model (COM) interface described below, to communicate with the Security Configuration Editor or Manager snap-ins.

Building the Attachment Engine DLL

Building the service attachment involves implementing three interfaces and installing/registering the attachment with the Security Configuration tool set. Security Configuration tool set will load the attachment and call these interfaces based on user-invoked operations.

Before describing which interfaces need to be implemented and how to implement them, it is important to define various data structures and support APIs provided by Security Configuration tool set.

The Data Structures

Note that the data structures described here are defined in scesvc.h, included in the Microsoft Win32® SDK.

- SCE Status Codes—various Security Configuration tool set status codes returned by helper functions and expected from the attachment interfaces.

```
typedef DWORD                   SCESTATUS;

#define SCESTATUS_SUCCESS              0L
#define SCESTATUS_INVALID_PARAMETER    1L
#define SCESTATUS_RECORD_NOT_FOUND     2L
#define SCESTATUS_INVALID_DATA         3L
#define SCESTATUS_OBJECT_EXIST         4L
#define SCESTATUS_BUFFER_TOO_SMALL     5L
#define SCESTATUS_PROFILE_NOT_FOUND    6L
#define SCESTATUS_BAD_FORMAT           7L
#define SCESTATUS_NOT_ENOUGH_RESOURCE  8L
#define SCESTATUS_ACCESS_DENIED        9L
#define SCESTATUS_CANT_DELETE          10L
#define SCESTATUS_PREFIX_OVERFLOW      11L
#define SCESTATUS_OTHER_ERROR          12L
#define SCESTATUS_ALREADY_RUNNING      13L
#define SCESTATUS_SERVICE_NOT_SUPPORT  14L
```

- SCE Handles—the opaque handles provided by Security Configuration tool set to attachment interfaces for support of callback APIs.

```
typedef PVOID SCE_HANDLE;
typedef ULONG SCE_ENUMERATION_CONTEXT, *PSCE_ENUMERATION_CONTEXT;
```

- SCE Service Configuration Information—the information structure to be used by attachment APIs when querying or setting configuration information to the database or configuration via the support callback APIs.

```
typedef enum _SCESVC_INFO_TYPE {

    SceSvcConfigurationInfo,
    SceSvcAnalysisInfo,
    SceSvcInternalUse                // DO NOT USE.

} SCESVC_INFO_TYPE;

typedef struct _SCESVC_CONFIGURATION_LINE_ {

    LPTSTR  Key;
    LPTSTR  Value;
    DWORD   ValueLen; // number of bytes

} SCESVC_CONFIGURATION_LINE, *PSCESVC_CONFIGURATION_LINE;

typedef struct _SCESVC_CONFIGURATION_INFO_ {

    DWORD   Count;
    PSCESVC_CONFIGURATION_LINE Lines;

} SCESVC_CONFIGURATION_INFO, *PSCESVC_CONFIGURATION_INFO;
```

- SCE Service Analysis Information—the information structure to be used by attachment APIs when querying or setting analysis information to the database via the support callback APIs.

```
typedef enum _SCESVC_INFO_TYPE {

    SceSvcConfigurationInfo,
    SceSvcAnalysisInfo,
    SceSvcInternalUse         // DO NOT USE

} SCESVC_INFO_TYPE;


typedef struct _SCESVC_ANALYSIS_LINE_ {

    LPTSTR  Key;
    PBYTE   Value;
    DWORD   ValueLen; // number of bytes

} SCESVC_ANALYSIS_LINE, *PSCESVC_ANALYSIS_LINE;

typedef struct _SCESVC_ANALYSIS_INFO_ {

    DWORD   Count;
    PSCESVC_ANALYSIS_LINE Lines;

} SCESVC_ANALYSIS_INFO, *PSCESVC_ANALYSIS_INFO;
```

Security Configuration Tool Set Helper APIs

Security Configuration tool set provides a set of support application programming interfaces (APIs) that the attachment should use to read or write information to the configuration file and the database.

These APIs are:

- **SceSvcQueryInfo**—this API lets the attachment query configuration or analysis information from the database for a given service.
- **SceSvcSetInfo**—this API lets the attachment set configuration or analysis information in the database for a given service.

- **SceSvcFree**—this API should be used to free buffers allocated by the Security Configuration tool set for the attachment.
- **SceSvcConvertSDToText**—this API takes a binary self-relative Windows NT security descriptor and returns a text representation for it. This is useful for storing security descriptors in configuration files.
- **SceSvcConvertTextToSD**—this API takes a text form security descriptor that was originally generated via **SceSvcConvertSDToText**, and returns a self-relative binary Windows NT security descriptor that represents it. This is useful in reading a textual security descriptor from configuration file and applying it to an object on the system.

These APIs are defined in scesvc.h in the Win32 SDK. The static library to link to is scesvc.lib, available for x86 and Alpha platforms. These APIs are described in greater detail next.

SceSvcQueryInfo

```
SCESTATUS
WINAPI
SceSvcQueryInfo(
    IN SCE_HANDLE           sceHandle,
    IN SCESVC_INFO_TYPE     sceType,
    IN LPTSTR               lpPrefix OPTIONAL,
    IN BOOL                 bExact,
    OUT PVOID               *ppvInfo,
    OUT PSCE_ENUMERATION_CONTEXT psceEnumHandle
    );
```

The **SceSvcQueryInfo** support API allows the attachment to query configuration or analysis information from the database.

The parameters are:

- **sceHandle**—the opaque SCE handle passed to the attachment by Security Configuration tool set. This is used to determine where the information will be stored.
- **sceType**—this must be `SCE_SERVICE_CONFIGURATION_INFO` if configuration information is being queried, or be `SCE_SERVICE_ANALYSIS_INFO` if analysis information is being queried.
- **lpPrefix**—this parameter may be NULL. If it is NULL, all keys are returned. If a string is supplied, then information returned contains all keys (and corresponding values) with the same prefix as the specified string.
- **bExact**—this parameter is not used if **lpPrefix** is NULL. If this parameter is TRUE, the key that matches exactly with the specifed string in **lpPrefix** is returned. If this parameter is FALSE, all keys (and their values) that have the same prefix as the specified string in **lpPrefix** are returned.
- **ppvInfo**—this must be a pointer to a pointer of type `SCESVC_CONFIGURATION_INFO` if the **sceType** is `SceSvcConfigurationInfo`. Otherwise, it must be `SCESVC_ANALYSIS_INFO` if the **sceType** is `SceSvcAnalysisInfo`.

Security Configuration tool set—and not the attachment—will allocate the buffer; therefore the pointer must point to NULL.

- **psceEnumHandle**—this contains the handle that must be used in successive calls to this API. The API may not return all the keys in a single call as there could be large number of keys. (The maximum number of keys returned in a single call is 256.)

This API returns the following codes:

- SCESTATUS_SUCCESS
- SCESTATUS_INVALID_PARAMETER
- SCESTATUS_RECORD_NOT_FOUND
- SCESTATUS_BAD_FORMAT
- SCESTATUS_OTHER_ERROR
- SCESTATUS_NOT_ENOUGH_RESOURCE

SceSvcSetInfo

```
SCESTATUS
WINAPI
SceSvcSetInfo(
    IN SCE_HANDLE            sceHandle,
    IN SCESVC_INFO_TYPE sceType,
    IN LPTSTR               lpPrefix OPTIONAL,
    IN BOOL                 bExact,
    IN PVOID                pvInfo
    );
```

The **SceSvcSetInfo** support API allows the attachment to set/overwrite configuration/analysis information stored in the database about a particular service.

The parameters are:

- **SceHandle**—the opaque handle that Security Configuration tool set passes to the attachment. This is used to determine where the information will be stored.
- **SceType**—this must be `SCE_SERVICE_CONFIGURATION_INFO` if configuration information is being set, or `SCE_SERVICE_ANALYSIS_INFO` if analysis information is being set.
- **lpPrefix**—this parameter may be NULL. If it is NULL, all service information is overwritten with the supplied information. If a string is supplied, then information overwritten contains all keys (and corresponding values) with the same prefix as the specified string.
- **bExact**—this parameter is not used if **lpPrefix** is NULL. If this parameter is TRUE, the key that matches exactly with the specified string in **lpPrefix** is overwritten. If this parameter is FALSE, all keys (and their values) that have the same prefix as the specified string in **lpPrefix** are overwritten.
- **pvInfo**—this must be a pointer of type `SCESVC_CONFIGURATION_INFO` if the **sceType** is `SceSvcConfigurationInfo`. Otherwise, it must be `SCESVC_ANALYSIS_INFO` if the **sceType** is `SceSvcAnalysisInfo`.

This API returns the following codes:

- SCESTATUS_SUCCESS
- SCESTATUS_INVALID_PARAMETER
- SCESTATUS_RECORD_NOT_FOUND
- SCESTATUS_BAD_FORMAT
- SCESTATUS_NOT_ENOUGH_RESOURCE
- SCESTATUS_ACCESS_DENIED
- SCESTATUS_DATA_OVERFLOW
- SCESTATUS_OTHER_ERROR

SceSvcFree

```
SCESTATUS
WINAPI
SceSvcFree(
    IN PVOID pvServiceInfo
    );
```

The attachment must call this API to free buffers allocated by Security Configuration tool set in calls to **SceSvcQueryInfo**.

The pointer **ServiceInfo** points to the allocated buffer.

This API returns the following codes:

- SCESTATUS_SUCCESS
- SCESTATUS_INVALID_PARAMETER

SceSvcConvertSDToText

```
SCESTATUS
WINAPI
SceSvcConvertSDToText (
    IN PSECURITY_DESCRIPTOR   pSD,
    IN SECURITY_INFORMATION   siSecurityInfo,
    OUT PWSTR                  *ppwszTextSD,
    OUT PULONG                 pulTextSize
    );
```

SceSceConvertSDToText is a helper API that allows the attachment to convert a self-relative security descriptor into a textual form that can be stored in the configuration file and the database. This API is useful when attachment is configuring security on a service that supports private objects that have security descriptors on them.

The parameters are:

- **pSD**—pointer to the security descriptor. (Refer to the Win32 SDK for APIs that manipulate security descriptors.)
- **siSecurityInfo**—the security information part of the security descriptor that must be converted to textual form. (Refer to the Win32 SDK for value values of

SECURITY_INFORMATION.)

- **ppwszTextSD**—the string form of the security descriptor returned by the API. The buffer is allocated by the API and must be freed using **SceSvcFree** when it is no longer needed.
- **pulTextSize**—pointer to a ULONG which is filled with the length of the string.

The string to return the textual form is allocated by this helper API. It must be freed using **LocalFree**.

This API returns the following codes:

- SCESTATUS_SUCCESS
- SCESTATUS_INVALID_PARAMETER
- SCESTATUS_NOT_ENOUGH_RESOURCE

SceSvcConvertTextToSD

```
SCESTATUS
WINAPI
SceSvcConvertTextToSD (
    IN  PWSTR                     pwszTextSD,
    OUT PSECURITY_DESCRIPTOR   *ppSD,
    OUT PULONG                    pulSDSize,
    OUT PSECURITY_INFORMATION   psiSeInfo
    );
```

SceSvcConvertTextToSD is a helper API that allows the attachment to convert a textual form of a security descriptor (created earlier using **SceSvcConvertSDToText**) back to its self-relative binary form.

The parameters are:

- **pwszTextSD**—the text form of the security descriptor.
- **ppSD**—pointer to a security descriptor pointer. This API will allocate necessary memory to create the self-relative security descriptor. It must be freed using **SceSvcFree** when it is no longer needed.
- **pulSDSize**—the size of allocated security descriptor.
- **psiSeInfo**—the pieces of valid security information in the descriptor. (Refer to the Win32 SDK for values of SECURITY_INFORMATION.)

The buffer to return security descriptor is allocated by this helper API. It must be freed using **LocalFree**.

This API returns the following codes:

- SCESTATUS_SUCCESS
- SCESTATUS_INVALID_PARAMETER
- SCESTATUS_RECORD_NOT_FOUND
- SCESTATUS_NOT_ENOUGH_RESOURCE

Required Attachment Interfaces

The three interfaces that the attachment must implement are:

- **SceSvcAttachmentConfig**—Security Manager calls this interface when the system is configured.
- **SceSvcAttachmentAnalyze**—Security Manager calls this interface when the system is analyzed.
- **SceSvcAttachmentUpdate**—Security Manager calls this interface when it receives an configuration update request from the MMC snap-in.

SceSvcAttachmentConfig

```
SCESTATUS
WINAPI
SceSvcAttachmentConfig(
IN SCE_HANDLE sceHandle,
        OUT PWSTR *ppszErrMessage OPTIONAL,
        OUT PDWORD pdErrLength
        );
```

The parameters are:

- **sceHandle**—the opaque handle that Security Configuration tool set passes to the attachment for use during callbacks on various support interfaces defined above. Security Configuration tool set uses this handle to read or write information passed by the attachment to the database.
- **ppszErrMessage**—the attachment can allocate a string buffer using **LocalAlloc** to return an error message in this optional parameter.
- **pdErrLength**—the length of the allocated error message buffer (in bytes). Security Configuration tools will always free this buffer using **LocalFree**; therefore, the buffer must be allocated with **LocalAlloc**.

This interface must do the following:

- Use the Security Configuration tool set support interface **SceSvcQueryInfo** to query configuration information from the configuration.
- Configure the service using the configuration information.

This interface returns the following codes:

- SCESTATUS_SUCCESS if the call completes successfully.
- Any other status code defined above.

Sample Code

```
SCESTATUS
WINAPI
SceSvcAttachmentConfig(
    IN SCE_HANDLE sceHandle,
    OUT PWSTR *ppszErrMessage,
    OUT PDWORD *pdErrLength
    )
{
        //
        //variable definitions
        //
        PSCESVC_CONFIGURATION_INFO    pConfigInfo = NULL;
        SCESTATUS      retCode;
        SCE_ENUMERATION_CONTEXT       EnumContext = 0;

if ( sceHandle == NULL ) {
return(SCESTATUS_INVALID_PARAMETER);
}

//
// now read the information and configure system using it.
//
// NOTE: you may decide to read all the information first
// and then do the configure, it is implementor's choice.
//

do {
        retCode = SceSvcQueryInfo(
                                      sceHandle,
                                      SceSvcConfigurationInfo,
                                      NULL,
                                      FALSE,
                                      (PVOID *)&pConfigInfo,
                                      &EnumContext
                                      );

        if(retCode == SCESTATUS_SUCCESS &&
                pConfigInfo != NULL)
        {
                ULONG   i;
                //
                // We have some information, let's configure.
                //
                for(i = 0;i < pConfigInfo->Count; i++)
                {
                        if(pConfigInfo->Line[i].Key == NULL)
                                continue;

                        //
                        // We have a key that we should process.
                        // This will the core of doing configuration.
                        //
                        ProcessConfigurationLine(pConfigInfo->Line[i]);
                }

                //
                // free the data we got back.
                //
                SceSvcFree((PVOID)pConfigInfo);
                PConfigInfo = NULL;
        }
        //
        // handle other return codes, as needed.
        //
} while ( retCode == SCESTATUS_SUCCESS && CountReturned > 0);

//
// if return code is not success, we should set up
// error message appropriately.
//

//
// return the retCode.
//
return retCode;

}
```

SceSvcAttachmentAnalyze

```
SCESTATUS
WINAPI
SceSvcAttachmentAnalyze(
IN SCE_HANDLE sceHandle,
        OUT PWSTR *ppszErrMessage,
        OUT PDWORD pdErrLength
);
```

The parameters are:

- **sceHandle**—the opaque handle that Security Configuration tool set passes to the attachment to be used during callbacks on various support interfaces. Security Configuration tool set uses this handle to read and information from the database and to write the analysis information to the database.
- **ppszErrMessage**—the attachment can allocate a string buffer using **LocalAlloc** to return an error message in this optional parameter.
- **pdErrLength**—the length of the allocated error message buffer (in bytes). Security Configuration tool set will always free this buffer using **LocalFree**; therefore, the buffer must be allocated with **LocalAlloc**.

This interface must do the following:

- Query configuration information from the service directly.
- Use the Security Configuration tool set's support interface **SceSvcQueryInfo** to query configuration information from the configuration.
- Compute the differences of the parameters based on type and syntax.
- Use the Security Configuration tool set's support interface **SceSvcSetInfo** to write the differential information to the database.

This interface returns the following codes:

- SCESTATUS_SUCCESS if the call completes successfully.
- Any defined SCESTATUS error codes are accepted.

Sample Code

```
SCESTATUS
WINAPI
SceSvcAttachmentAnalyze(
IN SCE_HANDLE sceHandle,
       OUT PWSTR *ppszErrMessage,
       OUT PDWORD pdErrLength
);
{
       //
       // define various local variables.
       //

       if(sceHandle == NULL)
              return (SCESTATUS_INVALID_PARAMETER);

//
// now read the base config information, query system
// setting corresponding to it, compare them
// and write to the database.
//
//

do {
       retCode = SceSvcQueryInfo(
                                   sceHandle,
                                   SceSvcConfigurationInfo,
                                   NULL,
                                   FALSE,
                                   &pConfigInfo,
                                   &EnumContext
                                   );

       if(retCode == SCESTATUS_SUCCESS &&
                pConfigInfo != NULL)
        {
                ULONG   i;
                //
                // We have some information, let's configure.
                //
                for(i = 0;i < pConfigInfo->Count; i++)
                {
                        if(pConfigInfo->Line[i].Key == NULL)
                                continue;

                        //
                        // We have a key that we should query.
                        // This function is expected to query
                        // the system configuration corresponding
                        // to the key value.
                        //
                        QueryConfigurationLine(pConfigInfo->Line[i].Key,
&SystemValue);

                                //
                                // now compare the values.
                                //
                                CompareValue(pConfigInfo->Line[i].Key,
                                                    SystemValue,
                                                    pConfigInfo-
>Line[i].Value,
                                                    &Result
                                               );

                              //
                              // Check if there is something that should
                              // be written to analysis part of the
                              // database.
if(Result != NULL)
                              {
                                   //
                                   // we will overwrite exactly one
                                   // value.
                                   // more efficient way to do this
                                   // would be to accumulate a
                                   // set of values and commit.
                                   //
                                   retCode = SceSvcSetInfo(
                                                    sceHandle,
                                                    SceSvcAnalysisInfo,
                                                    PconfigInfo-
>Line[I].Key,
                                                    TRUE,
                                                    Result
                                                    );
                                   if(retCode != SCESTATUS_SUCCESS)
                                   {
                                        // if it doesn't get set, we
                                        // need to do some cleanup
                                        // here.
                                   }
                              }
               }

               //
               // free the data we got back.
               //
               SceSvcFree((PVOID)pConfigInfo);
               PConfigInfo = NULL;

//
// should also free possible buffers SystemValue and
// Result, up to each attachment
//

       }
       //
       // handle other return codes, as needed.
       //
} while ( retCode == SCESTATUS_SUCCESS && pConfigInfo != NULL);

//
// if return code is not success, we should set up
// error message appropriately, if error buffer is not NULL
//

//
// return the retCode.
//
return retCode;

}
```

SceSvcAttachmentUpdate

```
SCESTATUS
WINAPI
SceSvcAttachmentUpdate(
        IN SCE_HANDLE sceHandle,
        IN SCESVC_CONFIGURATION_INFO *ServiceInfo
        );
```

Security Configuration Editor (or Manager) calls this interface when the Security Configuration Editor (or Manager) snap-in passes service specific changes to the configuration settings stored in the database.

The parameters are:

- **sceHandle**—the opaque handle that Security Configuration tool set passes to the attachment to be used during callbacks on various support interfaces. Security Configuration tool set uses this handle to read and information from the database and to write the analysis information to the database.
- **ServiceInfo**—the updated configuration information based on user edits and supplied by the attachment's extension snap-in. (See the explanation of the SCESVC_CONFIGURATION_INFO data structure in the data structures section.)

This attachment interface must do the following:

- Use the Security Configuration tool set's support interface **SceSvcQueryInfo** to query the base information (configuration information) stored in the database.
- Use the Security Configuration tool set's support interface **SceSceQueryInfo** to query the last set of differences (analysis information) stored in the database
- Use the **ServiceInfo** supplied to compute the new base configuration information.
- Use the **ServiceInfo** supplied and the last stored differences to compute the new differential information.
- Use the Security Configuration tool set's support interface **SceSvcSetInfo** to write the new base configuration information to the database.
- Use the Security Configuration tool set's support interface **SceSvcSetInfo** to write the new differential information to the database.

This interface returns the following codes:

- SCESTATUS_SUCCESS if the call completes successfully.
- Any valid SCESTATUS error codes are accepted.

Sample Code

```
SCESTATUS
WINAPI
SceSvcAttachmentUpdate(
        IN SCE_HANDLE sceHandle,
        IN SCESVC_CONFIGURATION_INFO *ServiceInfo
        );
{

        if(sceHandle == NULL || ServiceInfo == NULL)
                return(SCESTATUS_INVALID_PARAMETER);

        //
        // process each line of the passed information.
        //
        for(i=0; i < ServiceInfo->Count; i++)
        {
                EnumContext = 0;
                retCode = SceSvcQueryInfo(
                                        sceHandle,
                                        SceSvcConfigurationInfo,
                                        ServiceInfo->Line[i].Key,
                                        TRUE,
                                        (PVOID *)&pConfigInfo,
                                        &EnumContext
                                        );

                if(retCode != SCESTATUS_SUCCESS &&
retCode != SCESTATUS_RECORD_NOT_FOUND)
{
                        //
                        // handle the error here.
                        //
                        break;
                }

                //
                // if the value specified is NULL, deletion
                // of the key is requested.
                //
if(ServiceInfo->Line[i].Value == NULL)
                {
                        if(retCode == SCESTATUS_SUCCESS)
                        {
                                //
                                // Lets ensure that analysis is ok.
                                //
                                EnumContext = 0;
                                retCode = SceSvcQueryInfo(
                                                        sceHandle,
                                                        SceSvcAnalysisInfo,
                                                        ServiceInfo-
>Line[i].Key,
                                                        TRUE,
                                                        (PVOID *)&pAnalInfo,
                                                        &EnumContext
                                                        );
                                if(retCode == SCESTATUS_RECORD_NOT_FOUND)
                                {
                                        //
                                        // Analysis Info was not found,
                                        // this means it was matched during
                                        // actual analysis.  Now, we are
                                        // deleting the configuration info,
                                        // hence current configuration is
                                        // what analysis should save.
                                        //
                                        UpdateInfo->Count = 1
                                        UpdateInfo->Line = &UpdateLine;
                                        UpdateLine.Key = pConfigInfo-
>Line[0].Key;
                                        UpdateLine.Value =
(PBYTE)pConfigInfo->Line[0].Value;

                                        RetCode = SceSvcSetInfo(
                                                    SceHandle,

    SceSvcAnalysisInfo,
                                                    NULL,
                                                    TRUE,
                                                    &UpdateInfo
                                                    );
                                        if(retCode != SCESTATUS_SUCCESS)
                                        {
                                            //
                                            // cleanup, something
                                            // failed.
                                            //
                                        }
                                    }
                                    elseif (retCode == SCESTATUS_SUCCESS)
                                    {
                                        //
                                        // simply delete the configuration.
                                        // we already have analysis info in
                                        // place.
                                    }
                                    else
                                    {
                                        //
                                        // handle other error codes.
                                        //
                                    }

                                    //
                                    // delete the key
                                    //
                                    RetCode = SceSvcSetInfo(
                                            SceHandle,
                                            SceSvcConfigurationInfo,
                                            ServiceInfo->Line[i].Key,
                                            TRUE,
                                            NULL
                                            );
                                    if(retCode != SCESTATUS_SUCCESS)
                                    {
                                        //
                                        // error cleanup.
                                        //
                                    }

                                }
                                //
                                // SCESTATUS_RECORD_NOT_FOUND means nothing more.
                                // as the key does not even exist.
                                //
                            }
                            else
{
                                //
                                // Value to set is non-NULL,
                                // hence we must compare with current analysis
                                // if it is same, then delete the current analysis
                                // if it is different, do nothing to the analysis.
                                // Simply update the configuration info.
                                //
                                // left as exercise to the implementor.
                                //
                            }
```

```
                SceSvcFree(pConfigInfo);
                pConfigInfo = NULL;

SceSvcFree(pAnalInfo);
PAnalInfo = NULL;

        }
        //
        // error cleanup
        // set detail error message appropriately if the buffer
        // is not NULL
        //
        return retCode;

}
```

Installation and Registration

The service DLL must be installed on the Windows NT-based system where it is expected to be used. In addition, Security Configuration tool set's needs to be made aware of the presence of the attachment. This process of installation and registration should include following steps:

1. Copy the service attachment DLL to a particular directory. The preferred directory is %windir%\secedit\attachments. You can create this directory if it does not already exist.
2. Create a registry key under

```
HKEY_LOCAL_MACHINE\
   Software\
      Microsoft\
         Windows NT\
            CurrentVersion\
               SecEdit\Services\
                  [Service Name]
```

   The *Service Name* used here will be the registered name for the attachment. It should be unique so as to not collide with other attachments. The service name **must** be the same name used in Service Control Manager. The name used in Service Control Manager is the name to link each service with Security Configuration tool set.
3. Create the following values in this key:
   - Value Name = ServiceAttachmentPath
   - Value Type = REG_SZ
   - Value = the full path to the service attachment DLL (for example, %windir%\SecEdit\attachments\foobar.dll).

Building the Extension Snap-in

The Security Configuration tool set snap-ins are designed to be extensible to support the service attachment extension snap-ins. The communication between the Security Configuration tool set snap-ins and the extension snap-ins is handled by the standard MMC mechanisms and two well-defined Component Object Model (COM) interfaces, as described in this section. While the service attachment engine is responsible for configuring and analyzing service security and updating the service configuration in the database, the service attachment extension allows user to view, create, and modify configurations and analysis information. To function correctly, it is imperative that the service snap-in follow the MMC extension snap-in guidelines and the attachment guidelines provided in this document.

Each service attachment snap-in must be an extension snap-in, and these extension snap-ins provide functionality only when invoked by the Security Configuration Editor (or Manager) snap-in. Each service attachment snap-in can extend a Services node type only. It declares itself as being a subordinate to nodes of Services, and then for each occurrence of the Services node type, the MMC console automatically adds the related snap-in extensions. Each service attachment owns one scope pane node and the related result pane in MMC. Service attachment extensions must allow the user to create or modify service security settings in a configuration managed by the Security Configuration Editor (and Manager) snap-in. It must also be able to display configuration and/or analysis security settings with analysis status. It must support editing of service configuration settings for a system, and the analysis results must be updated based on the updated configuration settings.

It is up to the service attachment extension to determine the format and implementation logic of its own result pane. COM interfaces provide a way to extend Security Configuration Editor (and Manager) functionality for services without dictating how each service extension performs its particular tasks. See the COM interface layout shown in Figure A2.

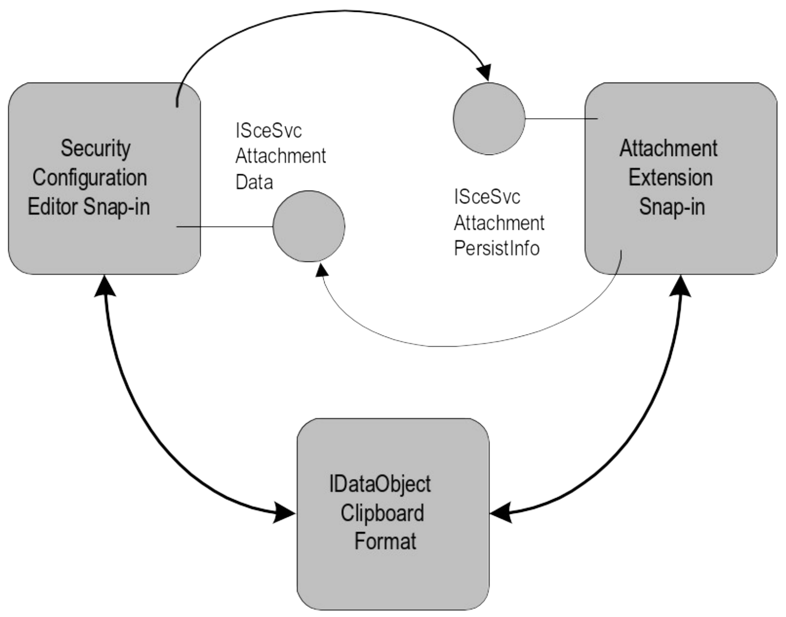

Security Configuration Editor Snap-in
ISceSvc Attachment Data
ISceSvc Attachment PersistInfo
Attachment Extension Snap-in
IDataObject Clipboard Format

Figure A2. COM Interface Layout for Service Attachment

In this illustration, the Security Configuration Editor (or Manager) Snap-in implements a COM interface called **ISceSvcAttachmentData**. The interface provides the attachment snap-in to query configuration and analysis information from configurations or the database respectively. The attachment snap-in implements the COM interface **ISceSvcAttachmentPersistInfo**, which is used by the Security Configuration Editor (or Manager) snap-in to get any modified information that may need to be written to the configuration or the database. he snap-in will then save this information appropriately.

There are three operations that the attachment snap-in must support:

- ***Display configuration and/or inspection information***—to display information, the attachment snap-in node extends the Security Configuration Editor (or Manager) snap-in via the Services node. The Security Configuration Editor (or Manager) node types that can be extended are:
  - Configuration Services NodeType ={24a7f717-1f0c-11d1-affb-00c04fb984f9}
  - Inspection Services NodeType = {678050c7-1ff8-11d1-affb-00c04fb984f9}

  When creating or editing a configuration, if the services node is expanded, all registered extension snap-ins will be notified by the MMC directly. Each attachment should then insert itself under the Services node, and then complete the following steps:
  - Use the **QueryInterface** method to query the Security Configuration tool set interface **ISceSvcAttachmentData**.
  - Call the **Initialize** method to inform Security Configuration tool set that it is loaded and to establish a context to communicate for appropriate information.

    - Either use the **GetData** method to pull information right away or wait until its node is selected by the user.
- ***Modify configuration information in the configurations***—the attachment snap-in must allow the user to modify configuration information about the service. The modified information must be held by the attachment snap-in until the Security Configuration Editor (or Manager) uses the **ISceSvcAttachmentPersistInfo** interface to pull the information. To avoid memory leaks, memory allocated is freed by the owner. For this reason, both interfaces have a **FreeBuffer** method.
- ***Modify configuration information in the database***—the attachment snap-in must support modifications to configuration information via the inspection node also. This is to allow the user to make changes and re-apply the configuration over time. The logic to do this should be identical to modifying information in the configuration files. The changes made will take effect on the saved configuration in the inspection database.

The Clipboard Format

```
#define CCF_SCESVC_ATTACHMENT ( L"CCF_SCESVC_ATTACHMENT" )
```

This clipboard is used for each attachment snap-in to extract the configuration file name from Security Configuration Editor (or Manager). The configuration file name is a PWSTR. This configuration name is used in further communications between the service attachment and Security Configuration Editor (or Manager) in the **Initialize** method.

The Extension Snap-in Interfaces

The extension snap-in queries the following Security Configuration Editor (or Manager) snap-in interfaces.

ISceSvcAttachmentData

```
class ISceSvcAttachmentData  : public IUnknown
{
public:
    virtual /* [helpstring] */ HRESULT STDMETHODCALLTYPE GetData(
        /* [in] */ SCE_HANDLE sceHandle,
        /* [in] */ SCESVC_INFO_TYPE sceType,
        /* [out] */ PVOID *ppvData,
        /* [in out] */ PSCE_ENUMERATION_CONTEXT psceEnumHandle ) = 0;

    virtual /* [helpstring] */ HRESULT STDMETHODCALLTYPE Initialize(
        /* [in] */ LPCTSTR ServiceName,
        /* [in] */ LPCTSTR TemplateName,
        /* [in] */ LPUNKNOWN lpUnknown,
        /* [out] */ SCE_HANDLE *sceHandle) = 0;

    virtual /* [helpstring] */ HRESULT STDMETHODCALLTYPE FreeBuffer(
        /* [in] */ PVOID pvData) = 0;

    virtual /* [helpstring] */ HRESULT STDMETHODCALLTYPE CloseHandle(
        /* [in] */ SCE_HANDLE sceHandle) = 0;

};
```

ISceSvcAttachmentData is the COM interface implemented by the Security Configuration Editor (or Manager) snap-in to support extension snap-ins. The attachment extension snap-in should use it to retrieve service-specific information for display user modification.

ISceSvcAttachmentPersistInfo

```
class ISceSvcAttachmentPersistInfo : public IUnknown
{
public:
    virtual /* [helpstring] */ HRESULT STDMETHODCALLTYPE Save(
        /* [out] */ SCE_HANDLE *sceHandle,
        /* [out] */ PVOID *ppvData,
        /* [out] */ PBOOL pbOverwriteAll ) = 0;

    virtual /* [helpstring] */ HRESULT STDMETHODCALLTYPE IsDirty() = 0;

    virtual /* [helpstring] */ HRESULT STDMETHODCALLTYPE FreeBuffer(
        /* [in] */ PVOID pvData) = 0;

};
```

The **IsceSvcAttachmentPersistInfo** interface is an abstract class that must be implemented by each attachment extension snap-in. The Security Configuration Editor (or Manager) snap-in calls this interface to check if there is modified information that must be written back to the configuration file or the database (using **IsDirty**). If that is the case, it calls the **Save** method to make the extension snap-in communicate the information that must be saved.

Installation and Registration

The Security Configuration Editor (or Manager) snap-in provides extensions only through the Security Configuration Editor (or Manager) namespace. Context menus, toolbars, toolbar buttons, and property pages are not extensible at this point. The attachment snap-in must extend the Security Configuration Editor (or Manager) name space by populating its own node at well-defined places in the namespace.

Attachment snap-ins should be registered under the registry key:

```
HKEY_LOCAL_MACHINE\
   Software\
     Microsoft\
       MMC\
         Snapins
```

The StandAlone key should NOT be created under the snap-in because each attachment snap-in must be an extension only.

Attachment snap-ins must also register themselves under the Security Configuration Editor Services NodeType subkeys as follows:

- To extend the Security Configuration Editor name space, use the registry key:

```
HKLM\
   Software\
      Microsoft\
        MMC\
          NodeTypes\
            24a7f717-1f0c-11d1-affb-00c04fb984f9\
              Extensions\
                NameSpace
```

- To extend the Security Configuration Manager inspection (analysis) name space, use the registry key:

```
  HKLM\
    Software\
      Microsoft\
        MMC\
          NodeTypes\
            678050c7-1ff8-11d1-affb-00c04fb984f9\
              Extensions\
                NameSpace
```

For more information, refer to the public header scesvc.h in the Platforms SDK.

To register the attachment snap-ins as extensions to Security Configuration Editor/ Manager snap-in, create these keys in your DllRegisterServer and DllUnregisterServer function implementations.

Initialization – Adding the Attachment Node

When a Services node under either Security Configuration Editor or under Security Configuration Manager is expanded, MMC uses **IComponentData::Notify** and the **MMCN_EXPAND** event to notify Security Configuration Editor/Manager and all of its extensions. Security Configuration Editor/Manager then extracts its internal format from the **lpDataObject** and stop further processing when it sees the Services node type. The attachment snap-ins (registered as extensions) extract the node type from the **lpDataObject** also. If the node type is one of the Services node types defined earlier, the attachment snap-ins insert their root nodes under the specified parent node.

```
//
// detect which extension node to extend
//

GUID* nodeType = ExtractNodeType(lpDataObject);

if ( nodeType == NULL ) {
   return S_OK;
}

if ( ::IsEqualGUID(*nodeType, cNodetypeSceTemplateServices) == TRUE )
   folderType =ATTACHEMNT_STATIC;     // defined by attachment writer.
else if (::IsEqualGUID(*nodeType, cNodetypeSceAnalysisServices)
== TRUE)
   folderType =ATTACHMENT_STATIC_ANALYSIS;
// defined by attachment writer

// Free resources
::GlobalFree(reinterpret_cast<HANDLE>(nodeType));

//
// As an extension snapin, the service attachment
// root node should be added
// Insert that node, and remember it
// as the root of the SMB Extension namespace.
//
CheckAndInsertRootNodeToMMCScopePane
```

The next major step in the initialization is to establish communication with the Security Configuration Editor/Manager snap-in. This is necessary because the attachment gets its data, as well any changes made by the user, from the Security Configuration Editor (or Manager). To do this, follow these steps:

1. Obtain the configuration name. If the Services node type that the attachment is inserted under was that of a configuration, then the attachment needs to know which configuration it is. It communicates this information to the Security Configuration Editor (or Manager) during interface initialization. The configuration name can be obtained via the clipboard format, as follows:

   ```
   PWSTR * TemplateName =
   ExtractTemplateNameFromDataObject(lpDataObject);
   ```

2. Set up the context with the Security Configuration Editor/Manager. Once the configuration name is known (or if the Service node is of type Inspection), the attachment snap-in must query the **ISceSvcAttachmentData** interface and call **Initialize** to set up the context.

```
//
// QueryInterface for the main snap-in's IUnkown.
//
LPUNKNOWN pUnk;

hr = lpDataObject->QueryInterface(IID_IUnknown,
 reinterpret_cast<void**>(&pUnk));

//
// QueryInterface ISceSvcAttachmentData
//
if ( SUCCEEDED(hr) ) {
    hr = pUnk->QueryInterface(IID_ISceSvcAttachmentData,
 reinterpret_cast<void**>(&pSceData));
}

…

//
// QueryInterface the attachment's IUnknown as
// that is needed by the main snap-in.
//
((LPUNKNOWN)m_pSnapin)->QueryInterface(IID_IUnknown,
reinterpret_cast<void**>(&pUnk));

//
// Call Initialize to setup context with main snap-in.
//
m_pSceData->Initialize(ServiceName, TemplateName, pUnk, &sceHandle);

…
```

NOTE: You must call **CloseHandle** to close the **sceHandle** once you are done.

3. Get the appropriate data. (The previous step really completes the initialization.) The attachment snap-in can use the established context to query appropriate data from Security Configuration Editor as needed by using the **GetData** interface. The attachment may decide to do this proactively as soon as it initializes with Security Configuration Editor, or it may wait until the user actually attempts to expand the attachment node by clicking it. The attachment can display the information received using any UI controls available.

```
//
// GetData – we get the configuration information here.
//
m_pSceData->GetData (sceHandle, SceSvcConfigurationInfo, &pData,
 &enumHandle );
```

NOTE: You must use the FreeBuffer method to free the buffer allocated here by Security Configuration Editor/Manager.

Implementing ISceSvcAttachmentPersistInfo

After initialization, it is important that the attachment implement the **ISceSvcAttachmentPersistInfo** interface. The Security Configuration Editor/Manager queries this interface at various times, as when saving the configuration and when closing the snap-in, to allow the attachment to save any modifications that the user may have made to the inspection database or to the associated configuration.

```
class CSceSvcAttachmentPersistInfo:
    public ISceSvcAttachmentPersistInfo,
    public CComObjectRoot
{
BEGIN_COM_MAP(CSceSvcAttachmentPersistInfo)
    COM_INTERFACE_ENTRY(ISceSvcAttachmentPersistInfo)
END_COM_MAP()

    friend class CDataObject;
    friend class CComponentDataImpl;

    CSceSvcAttachmentPersistInfo();
    ~CSceSvcAttachmentPersistInfo();

public:

    // ISceSvcAttachmentPersistInfo interface members
    STDMETHOD(IsDirty)();
    STDMETHOD(Save)(SCE_HANDLE *sceHandle, PVOID *ppvData,
PBOOL pbOverwriteAll );
    STDMETHOD(FreeBuffer)(PVOID pvData);

    ...

private:
    CString m_TemplateName;
    LPSCESVCATTACHMENTDATA m_pSceData;
    SCE_HANDLE m_sceHandle;

    ...

};

//
// Implementing IsDirty()
//
STDMETHODIMP CSceSvcAttachmentPersistInfo::IsDirty()
{
    if ( m_pSnapin == NULL ) {
return S_FALSE;
    }
    //
    // just calling the snapin's main IsDirty.
    //
    return m_pSnapin->IsDirty();
}

//
// Implementing Save()
//
STDMETHODIMP CSceSvcAttachmentPersistInfo::Save(
              SCE_HANDLE *psceHandle,
  PVOID *ppvData,
  PBOOL pbOverwriteAll )
{
    if ( psceHandle == NULL || ppvData == NULL ||
pbOverwriteAll == NULL ) {
        return E_INVALIDARG;
    }

    if ( m_pSnapin != NULL ) {

        m_pSnapin->SaveDataInBuffer(ppvData, pbOverwriteAll);

        *psceHandle = m_sceHandle;

    }

    return S_OK;
}

//
// Implementing FreeBuffer
//
STDMETHODIMP CSceSvcAttachmentPersistInfo::FreeBuffer(PVOID pvData)
{
    if ( pvData == NULL ) {
        return S_OK;
    }

    PSCESVC_ANALYSIS_INFO pTempInfo=(PSCESVC_ANALYSIS_INFO)pvData;

    if ( pTempInfo->Lines != NULL ) {

        for ( DWORD i=0; i < pTempInfo->Count; i++ ) {

            if ( pTempInfo->Lines[i].Key != NULL )
                LocalFree(pTempInfo->Lines[i].Key);

            if ( pTempInfo->Lines[i].Value != NULL )
                LocalFree(pTempInfo->Lines[i].Value);
        }

        LocalFree( pTempInfo->Lines);
    }

    LocalFree(pTempInfo);

    return S_OK;
}
```