Security Reference Monitor

Executive Summary

The goal of this document is to explain the security reference monitor and its components from the standpoint from Windows Operating System.

Overview

This section explains the important components of the security infrastructure of the Windows Operating System and how it aligns for implementing a cybersecurity policy.

After completing this section, we will discuss, and you will be able to understand:

- The OS security components and their functions
- The high-level flow of events during user authentication
- How the OS performs access checks when a process tries to access a resource

Security System Components

These are the core components and databases that implement Windows security:

Security Reference Monitor (SRM)

Windows uses an access control list (ACL) to determine which objects have what security. The Windows kernel-mode Security Reference Monitor (SRM) provides routines for your driver to work with access control

A kernel component in the Windows executive **(%SystemRoot%\System32\Ntoskrnl.exe)** that is responsible for defining the access token data structure to represent a security context, performing security access checks on objects, manipulating privileges (user rights), and generating any resulting security audit messages.

Local Security Authority Subsystem (Lsass) controls

- which users are allowed to log on to the computer
- password policies
- privileges granted to users and groups
- system security auditing settings

- user authentication
- sending security audit messages to the Event Log

A user-mode process running the image **%SystemRoot%\System32\Lsass.exe** that is responsible for the local system security policy (such as which users are allowed to log on to the machine, password policies, privileges granted to users and groups, and the system security auditing settings), user authentication, and sending security audit messages to the Event Log.

The Local Security Authority service **(Lsasrv—%SystemRoot%\System32\Lsasrv.dll**), a library that Lsass loads, implements most of this functionality.

Lsass Policy Database in registry (HKLM/Security) stores:

- trusted domains
- who has permission to access the system and how (interactive, network, service logon)
- who has which priviledges
- stores cached domain logons
- windows service user account logons

A database that contains the local system security policy settings.

This database is stored in the registry under **HKLM\SECURITY**. It includes such information as what domains are entrusted to authenticate logon attempts, who has permission to access the system and how (interactive, network, and service logons), who is assigned which privileges, and what kind of security auditing is to be performed.

The Lsass policy database also stores secrets that include:

- Logon information used for cached domain logons
- Windows service user-account logons

Security Accounts Manager (SAM) Service

- Runs in LSASS process
- Set of subroutines providing access to the SAM database

A set of subroutines responsible for managing the database that contains the user names and groups defined on the local machine.

The SAM service, which is implemented as **%SystemRoot%\System32\Samsrv.dll**, runs in the Lsass process.

SAM Database in registry (HKLM/SAM)

- Contains local users and groups including password and other attributes
- On DC stores the system admin recovery account definition and password

A database that contains the defined local users and groups along with their passwords and other attributes on systems not functioning as domain controllers. On domain controllers, the SAM stores the system's administrator recovery account definition and password. This database is stored in the registry under **HKLM\SAM**.

Active Directory

- Directory service
- Contains database that stores information about objects in a domain including security data (passwords, privileges, and access control lists)
- An object can be user or group or hardware component (computer or printer).
- Replicated between domain controllers
- AD directory system agent runs in LSASS

A directory service that contains a database that stores information about objects in a domain. A domain is a collection of computers and their associated security groups that are managed as a single entity.

Active Directory stores information about the objects in the domain including users, groups, and computers. Password information and privileges for domain users and groups are stored in Active Directory, which is replicated across the computers that are designated as domain controllers of the domain.

The Active Directory server, implemented as **%SystemRoot%\System32\Ntdsa.dll**, runs in the Lsass process.

Authentication Packages

There is mostly two: msv1_0, Kerberos.dll

- HKLM\SYSTEM\CurrentControlSet\Control\Lsa\MSV1_0
- C:\Windows\System32\msv1_0.dll
- Run in context of LSASS and client process
- Responsible to check user-id/password match and return user security identity
- Used by LASS to generate access token

These include dynamic-link libraries (DLLs) that run both in the context of the Lsass process and client processes and that implement Windows authentication policy.

An authentication DLL is responsible for checking whether a given user name and password match, and if so, returning to the Lsass information detailing the user's security identity, which Lsass uses to generate a token.

Interactive logon Manager (Winlogon)

A user mode process running %System-Root%\System32\Winlogon.exe that is responsible for responding to the SAS and for managing interactive logon sessions. For example, Winlogon creates a user's first process when the user logs on.

Logon User Interface (LogonUI)

A user mode process that presents users with the user interface (UI) they can use to authenticate themselves on the system. Uses credential providers to query user credentials through various methods.

Credential Providers (CPs)

In-process COM objects that run in the LogonUI process (started on demand by Winlogon when the secure attention sequence (SAS) is performed) and used to obtain a user's name and password, smartcard PIN, or biometric data (such as a fingerprint). The standard CPs are %SystemRoot%\System32\authui.dll and %SystemRoot%\System32\SmartcardCredentialProvider.dll

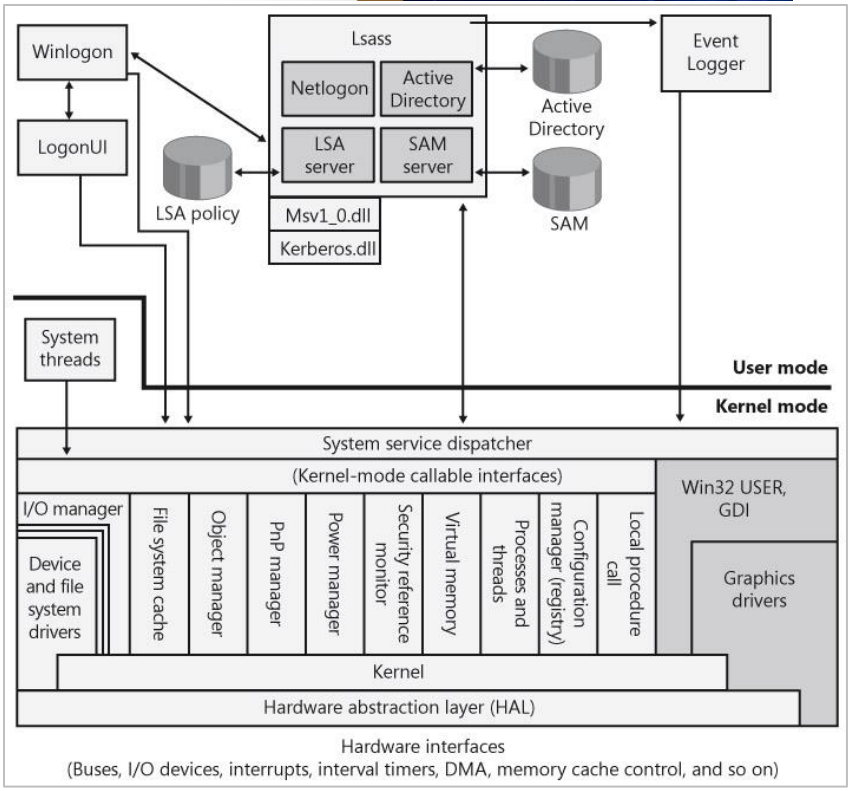
Network Logon Service (Netlogon)

A Windows service (%SystemRoot%\System32\Netlogon.dll) that sets up the secure channel to a domain controller, over which security requests such as an interactive logon (if the domain controller is running Windows NT 4) or LAN Manager and NT LAN Manager (v1 and v2) authentication validation—are sent.

Kernel Security Device Driver (KSecDD)

A kernel mode library of functions that implement the local procedure call (LPC) interfaces that other kernel mode security components including the Encrypting File System (EFS), use to communicate with Lsass in user mode. KSecDD is located in %SystemRoot%\System32\Drivers\Ksecdd.sys.

The figure below shows the relationships between some of these Windows Security components and the databases they manage.



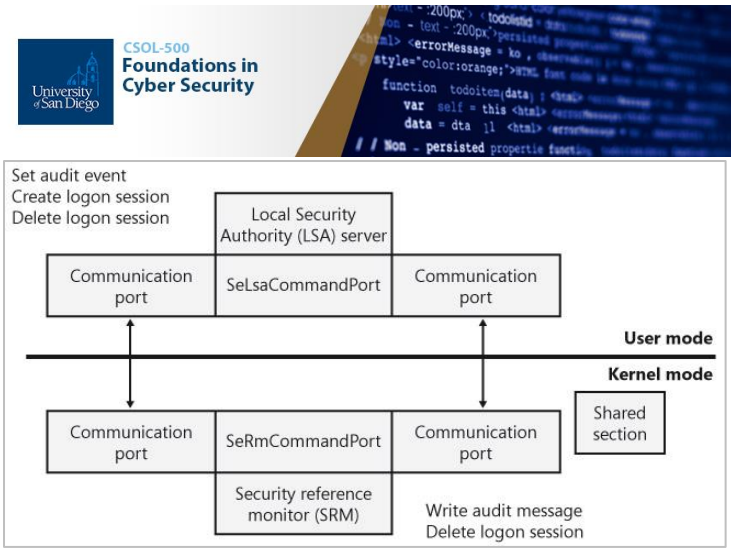
The **Security Reference Monitor**, which runs in kernel mode, and Lsass, which runs in user mode, communicate using the Advanced Local Procedure Call (ALPC) facility.

During system initialization, the Security Reference Monitor (SRM) creates a port named SeRmCommandPort, to which Lsass connects. When the Lsass process starts, it creates an ALPC port named SeLsaCommandPort.

The Security Reference Monitor (SRM) connects to this port, resulting in the creation of private communication ports. The Security Reference Monitor (SRM) creates a shared memory section for messages longer than 256 bytes, passing a handle in the connect call.

Once the Security Reference Monitor (SRM) and Lsass connect to each other during system initialization, they no longer listen on their respective connect ports.

Therefore, a later user process has no way to connect successfully to either of these ports for malicious purposes—the connect request will never complete.



The figure above shows the communication between the SRM and Lsass.

Access Checks

The Windows security model requires that a thread specify up front, at the time that it opens an object, what types of actions it wants to perform on the object. The object manager calls the SRM to perform access checks based on a thread's desired access and if the access is granted, a handle is assigned to the thread's process with which the thread (or other threads in the process) can perform further operations on the object. The object manager records the access permissions granted for a handle in the process's handle table.

One event that causes the object manager to perform security access validation is when a process opens an existing object using a name. When an object is opened by name, the object manager performs a lookup of the specified object in the object manager namespace. If the object is not located in a secondary namespace, such as the configuration manager's registry namespace or a file system driver's file system namespace, the object manager calls the internal function ObpCreateHandle once it locates the object. As its name implies, ObpCreateHandle creates an entry in the process's handle table that becomes associated with the object. ObpCreateHandle first calls ObpIncrementHandleCount to see if the thread has permission to access the object. If the thread does, ObpCreateHandle calls the executive function ExCreateHandle to create the entry in the process handle table. ObpIncrementHandleCount calls ObCheckObjectAccess to carry out the security access check.

ObpIncrementHandleCount passes ObCheckObjectAccess:

- The security credentials of the thread opening the object
- The types of access to the object that the thread is requesting (read, write, delete, and so forth)
- A pointer to the object

ObCheckObjectAccess first locks the object's security descriptor and the security context of the thread. The object security lock prevents another thread in the system from changing the object's security while the access check is in progress.

The lock on the thread's security context prevents another thread of that process or a different process from altering the security identity of the thread while security validation is in progress. ObCheckObjectAccess then calls the object's security method to obtain the security settings of the object. The call to the security method might invoke a function in a different executive component. However, many executive objects rely on the system's default security management support.

> When an executive component defining an object does not want to override the SRM's default security policy, it marks the object type as having default security. Whenever the SRM calls an object's security method, it first checks to see whether the object has default security. An object with default security stores its security information in its header, and its security method is SeDefaultObjectMethod. An object that does not rely on default security must manage its own security information and supply a specific security method. Objects that rely on default security include mutexes, events, and semaphores. A file object is an example of an object that overrides default security. The I/O manager, which defines the file object type, has the file system driver on which a file resides manage (or choose not to implement) the security for its files. Thus, when the system queries the security on a file object that represents a file on an NTFS volume, the I/O manager file object security method retrieves the file's security using the NTFS file system driver.

After obtaining an object's security information, ObCheckObjectAccess invokes the SRM function SeAccessCheck. SeAccessCheck is one of the functions at the heart of the Windows security model. Among the input parameters SeAccessCheck accepts are:

- The object's security information
- The security identity of the thread as captured by ObCheckObjectAccess
- The access that the thread is requesting

SeAccessCheck returns True or False depending on whether the thread is granted the access it requested to the object.

Another event that causes the object manager to execute access validation is when a process references an object using an existing handle. Such references often occur indirectly, as when a process calls on a Windows API to manipulate an object and passes an object handle.

For example, a thread opening a file can request read permission to a file. If the thread has permission to access the object in this way, as dictated by its security context and the security settings of the file, the object manager creates a handle—representing the file—in the handle table of the thread's process. The accesses the process is granted through the handle are stored with the handle by the object manager.

Subsequently, the thread could attempt to write to the file using the WriteFile Windows function, passing the file's handle as a parameter. The system service NtWriteFile, which WriteFile calls via Ntdll.dll, uses the object manager function ObReferenceObjectByHandle to obtain a pointer to the file object from the handle. ObReferenceObjectByHandle accepts the access that the caller wants from the object as a parameter. After finding the handle entry in the process's handle table, ObReferenceObjectByHandle compares the access being requested with the access granted at the time the file was opened. In this example, ObReferenceObjectByHandle will indicate that the write operation should fail because the caller did not obtain write access when the file was opened.

The Windows security functions also enable Windows applications to define their own private objects and to call on the services of the security reference monitor to enforce the Windows security model on those objects. Many kernel-mode functions that the object manager and other executive components use to protect their own objects are exported as Windows user mode APIs.

The user-mode equivalent of SeAccessCheck is AccessCheck, for example. Windows applications can therefore leverage the flexibility of the security model and transparently integrate with the authentication and administrative interfaces that are present in Windows.

The essence of the Security Reference Monitor’s security model is an equation that takes three inputs:

- The security identity of a thread
- The access that the thread wants to an object
- The security settings of the object

The output is either yes or no and indicates whether or not the security model grants the thread the access it desires.

Conclusion

This document provides a deep understanding of the Windows Components and the relationship with Security Reference Monitor (SRM). It shows how information security policies can be enforced with Windows Systems.

Appendix

- Source: **Windows Internals Books**
  https://docs.microsoft.com/en-us/sysinternals/learn/windows-internals

- Source: **NT Debugging Blog**
  https://blogs.msdn.microsoft.com/ntdebugging/tag/windows-internals/

- Source: **Windows Privilege Escalation**
  https://recon.cx/2008/a/thomas_garnier/LPC-ALPC-slides.pdf