

# Windows NT Security

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

Preventing unauthorized access to sensitive data is essential in environments in which multiple users have access to the same physical or network resources. Operating systems (OSs) and individual users must be able to protect files, memory, and configuration settings from unauthorized viewing and modification. OS security includes obvious mechanisms such as accounts, and passwords. However, OS security also includes less-obvious mechanisms for protecting the OS from corruption, preventing less-privileged users from performing actions such as rebooting the computer, and preventing the programs of less-privileged users from adversely affecting the programs of other users or the OS.

The stringent requirements of providing robust security influenced the design of Windows NT, which has earned a C2 security rating. This security rating puts NT on par with most UNIX systems. Although you're probably familiar with user accounts, groups, and NT's file and Registry security editors, you might find the way NT implements its logon validation, object protection, and privilege checking a mystery. Yet if you have a basic knowledge of what goes on behind the scenes in NT security, you'll know which security policies you need to install, and you can better protect your systems.

To help you gain a basic knowledge of NT security, I'm beginning a two-part look at NT security this month. I'll review what a C2 security rating is and what facilities an OS must include to earn a C2 rating. I'll discuss NT's security identifiers (SIDs), which NT uses to identify users, groups, computers, and domains. Next, I'll present an overview of NT's logon procedure, and I'll discuss local and network (domain) logon. Finally, I'll discuss access tokens. Next month, I'll conclude with a detailed description of object security access validation, client/server impersonation, privileges, and policies.

## C2 Security

The US Department of Defense (DoD) National Security Agency (NSA) established the National Computer Security Center (NCSC, at <http://www.radium.ncsc.mil>) in 1981 to help the government, corporations, and home users protect proprietary and personal data stored in computer systems. The NCSC created a range of security ratings, which Table 1 shows, that measure the degree of protection commercial OSs, network components, and trusted applications offer. The NCSC assigned these security ratings in 1983 based on DoD's Trusted Computer System Evaluation Criteria (TCSEC).

The security ratings are commonly known as the "Orange Book."

TABLE 1: NCSC Security Ratings	
Rating Code	Rating Name
A1	Verified Design
B3	Security Domains
B2	Structured Protection
B1	Labeled Security Protection
C2	Controlled Access Protection (Windows NT 3.5 earned this rating)
C1	Discretionary Access Protection (this rating is obsolete)
D	Minimal Protection

The TCSEC standard consists of levels of trust ratings, in which higher levels of security build on lower levels, adding more rigorous protection requirements. No OS has earned the A1 rating. A few

# Windows NT Security

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

OSs have earned B1, B2, and B3 ratings, including variants of HP's HP-UX (a UNIX system), Digital's Ultrix and SEVMS, Unisys' OS 1100, and Silicon Graphics' IRIX.

OSs that have earned the C2 rating include versions of IBM's OS/400 and Digital's OpenVMS. NT 3.5 (Workstation and Server) with Service Pack 3 (SP3) earned the C2 rating in July 1995. Microsoft reportedly submitted NT 4.0 for NCSC evaluation, but the evaluation process usually takes several years and is not complete at press time (Microsoft first submitted NT 3.5 in 1991). Because the security-related components in NT 4.0's architecture are virtually identical to those in NT 3.5's architecture, NT 4.0 will probably meet the C2 requirements.

To earn a C2 security rating, an OS must implement the following features: a secure logon facility, discretionary access control, auditing, and object reuse protection. A secure logon facility requires users to enter a unique identifier and password to identify themselves before it will grant them access to the computer. NT uses accounts for user identification and password-based logon for its default authentication mechanism.

When an OS implements discretionary access control, it lets all shareable OS resources associate with a block of information that specifies which users can perform operations on the resource. If you've viewed or set NTFS file or directory permissions or you've modified the security settings on Registry keys, you've seen a representation of NT's discretionary access control, which NT organizes as a list. The list elements describe the actions a user can and cannot perform on an object.

Auditing capability lets authorized users place watchdogs on resources that monitor and record users' failed or successful attempts to access the resources. The NTFS permission editors and the Registry provide access to NT's implementation of file system and Registry object auditing. All shareable objects in NT can have auditing enabled. But auditing can introduce unwanted overhead, so NT disables it systemwide by default.

To have object reuse protection, an OS must prevent users from seeing data that another user has deleted or from accessing memory that another user previously used and released. For example, in some OSs you can create a new file of a certain length and then examine the file's contents to see data that previously occupied the location on the disk allocated to the new file. This data might be sensitive information that another user stored in a file and then deleted. NT prevents this type of security breach by preinitializing file data, memory, and other objects when it allocates them. If you create a file, NT zeros the contents before you can access the file, which prevents you from seeing any data that existed previously in the file's location on the disk.

When NT earned its C2 security rating, NCSC also recognized NT as meeting two requirements of B-level security: Trusted Path functionality and Trusted Facility Management functionality. Trusted Path functionality prevents Trojan horse programs from intercepting a user's name and password as the user logs on. NT's Trusted Path functionality exists in the form of its Ctrl+Alt+Del logon-attention sequence. This sequence of keystrokes, the Secure Attention Sequence (SAS), causes an NT logon dialog box to pop up, which initializes a process that helps NT recognize would-be Trojan horses. NT bypasses any Trojan horse that presents a fake logon dialog when a user enters the attention sequence.

NT meets the Trusted Facility Management requirement by supporting separate account roles for administrative functions. For instance, NT provides separate accounts for administration (Administrators), user accounts charged with backing up the computer (Backup Operators), and

# Windows NT Security

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

standard users (Users). Microsoft is reportedly working on a B-level version of NT, but the company has not made a public statement about when it might release this version.

If you rely on NT's C2 security rating in your security decisions, you must keep in mind two important considerations. First, a C2 security rating is different from a C2 security certification. OSs and programs earn ratings, but individual installations must be certified. This distinction means that most NT installations are not C2 certified, nor would they necessarily want to because security needs vary, and too much security can hamper productivity. You can use the Microsoft Windows NT Server 4.0 Resource Kit tool C2Config to help your NT systems meet the requirements for a C2 certification.

Second, NT earned its C2 rating as a standalone system, with no networking enabled. If you take your C2Config C2-certified system and attach it to your LAN, your system loses its C2 certification. Securing a network-based system is harder than securing a standalone machine, but if you keep up to date with service packs and security alerts, you can remain close to a C2 certification level.

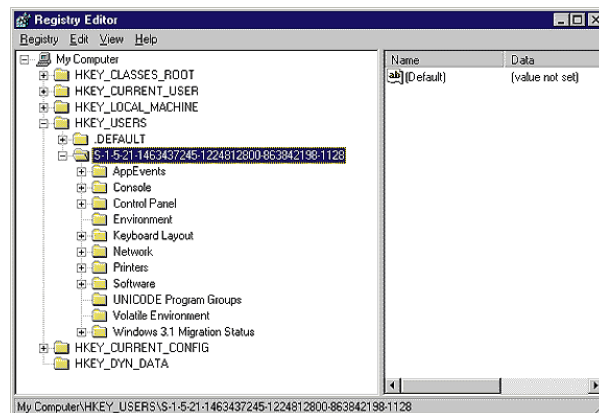
## SIDs

NT uses SIDs rather than names (which might not be unique) to identify entities that perform actions in a system. Users, local and domain groups, local computers, domains, and domain members have SIDs. A SID is a variable-length numeric value that consists of a SID revision number, a 48-bit identifier authority value, and a variable number of 32-bit subauthority or Relative Identifier (RID) values. The authority value identifies the agent that issued the SID, and this agent is typically an NT local system or a domain. Subauthority values identify trustees relative to the issuing authority, and RIDs are simply a way for NT to create unique SIDs from a base SID. Because SIDs are long and NT takes care to generate truly random values within each SID, it is virtually impossible for NT to issue duplicate SIDs on machines or domains anywhere in the world.

In text form, each SID carries an S prefix, and hyphens separate its various components. To see the SID representation for any account you are using, run regedit and open the HKEY\_USERS key, as Screen 1 shows. This key contains the current user's profile and the default account profile, which is in use when no one is logged on locally. In Screen 1, the SID of the current account is:

**S-1-5-21-1463437245-1224812800-863842198-1128**

This SID has a revision number of 1, one identifier authority of 5, (which represents the NT security authority), another identifier authority of 21 (which identifies the SID as not built in), three subauthority values, and one RID (1128). This SID is a domain SID, but a local computer on the domain would have a SID with the same revision number, identifier authority value, and number of subauthority values.



# Windows NT Security

Mark Russinovich  
(Reprinted from WindowsItPro Magazine)

When you install NT, the NT setup program issues the computer a SID. NT assigns SIDs to local accounts on the computer: Each local-account SID is based on the source computer's SID and has a RID at the end. RIDs for user accounts and groups start at 1000 and increase in increments of 1 for each new user or group. Similarly, NT issues a SID to each newly created NT domain. NT issues to new domain accounts SIDs that are based on the domain SID and have an appended RID (again starting at 1000 and increasing in increments of 1 for each new user or group). The RID in Screen 1 (1128) identifies that SID as the 129th SID the domain issued.

NT also defines several built-in local and domain SIDs to represent groups. For example, a SID that identifies all accounts is the Everyone or World SID: S-1-1-0. Another example of a group that a SID can represent is the network group, which is the group that represents users who can log on to a machine from the network. The network-group SID is S-1-5-2. NT issues SIDs that consist of a computer or domain SID with a predefined RID to many predefined accounts and groups. For example, the RID for the administrator account is 500, and the RID for the guest account is 501. A computer's local administrator account, for instance, has the computer SID as its base with the RID of 500 appended to it: S-1-5-21-13124455-12541255-61235125-500.

## Logon

NT implements a secure logon process that takes as input a username and password and returns as output to the OS SIDs that identify the user's account and the groups the account belongs to. In the first step of the secure logon process, NT recognizes the SAS and prompts the user for identification and a password. The winlogon.exe program is responsible for presenting NT's logon dialog boxes. Instead of containing a built-in user interface, Winlogon loads the interface dynamically. This strategy lets third-party vendors implement their logon interfaces. Logon interface packages are known as Graphical Identification and Authentication (GINA) libraries. The default logon interface, which presents the logon dialog box most of us are familiar with, is MSGINA (it's located in winnt\system32\msgina.dll). NT's ability to replace the logon interface lets third-party vendors replace MSGINA with a proprietary GINA. For example, a custom GINA might recognize a voice command instead of Ctrl+Alt+Del as the logon sequence, or it might use a retinal scanning device to identify users.

After users identify themselves to a GINA with a username and password, the GINA sends the gathered information to the Local Security Authority Sub System (LSASS, in winnt\system32\lsass.exe) process with a local procedure call (LPC) message. LSASS is the front end for logon authentication in NT. Authentication is the mechanism whereby NT validates the username and password and retrieves the SID information that identifies the user. Reflecting the configurability of Winlogon, LSASS uses a replaceable library as its authentication package. If you look at the Authentication Packages Registry value:

`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\CONTROL\Lsa`

you'll see MSV1\_0 listed as the LSASS authenticator. (MSV1\_0 is in winnt\system32\msv1\_0.dll.) If you have File and Print Services for NetWare (FPNW) or Client Services for NetWare (CSNW), you'll see an authentication package for that as well.

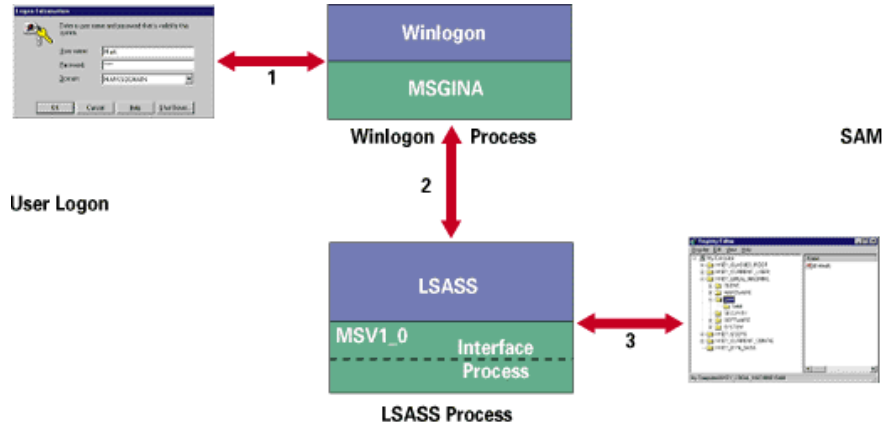
LSASS calls MSV1\_0 and gives it the username and password. Next, MSV1\_0 determines whether the logon attempt is local (onto a workgroup or the local computer) or domain based. MSV1\_0 consists of an interface component, a processing component, and a component called Netlogon, which I'll describe shortly. The interface component takes the password and encrypts it. If the logon is

# Windows NT Security

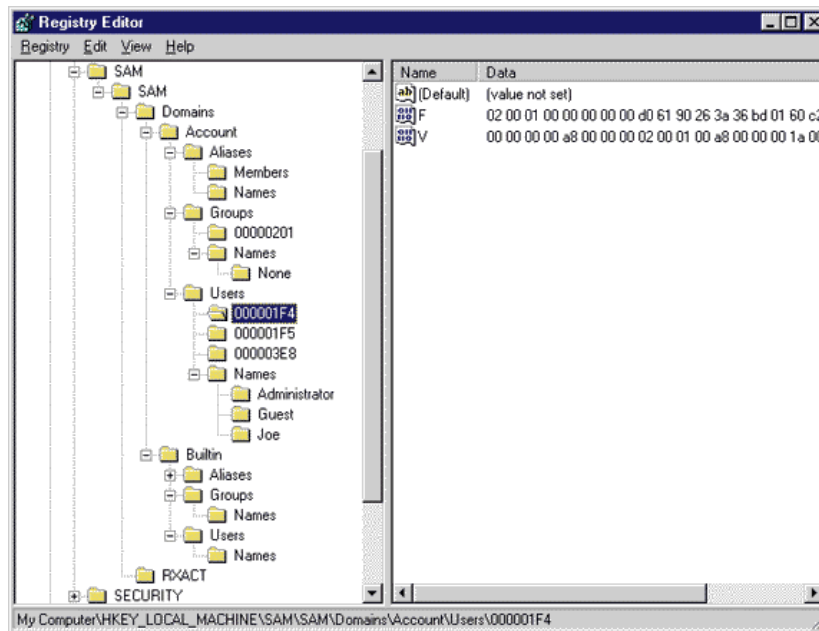
Mark Russinovich

(Reprinted from WindowsItPro Magazine)

local or directed at a domain and the machine the logon is taking place on is a domain controller, the interface component passes the username and password to the processing component. The processing component uses the functions exported by the samsrv.dll library to look in HKEY\_LOCAL\_MACHINE\SAM, which serves as the local account database on NT, to validate the name and password. Figure 1 illustrates this local validation flow of control.



HKEY\_LOCAL\_MACHINE\SAM, which is also known as the Security Accounts Manager (SAM) database, is by default not viewable even from an Administrator account. Consequently, the SAM's content and layout are a mystery to most people. Looking inside the SAM probably won't provide you with any useful information, but if you're like me, you're curious about why NT keeps the SAM so secret. If you change the SAM's security settings (do so only in a nonproduction environment) and open it, you'll see something like Screen 2 on page 66. In Screen 2, you can see the Domains key, which contains all local account information beneath it. If the machine is a domain controller, the Domains key contains domain-account and computer-membership information as well.



Under the Account subkey, you'll find information pertaining to non-built-in aliases, accounts, groups, and computers. In Screen 2, you can see three add-on accounts (Administrator, Guest, and Joe)

# Windows NT Security

Mark Russinovich

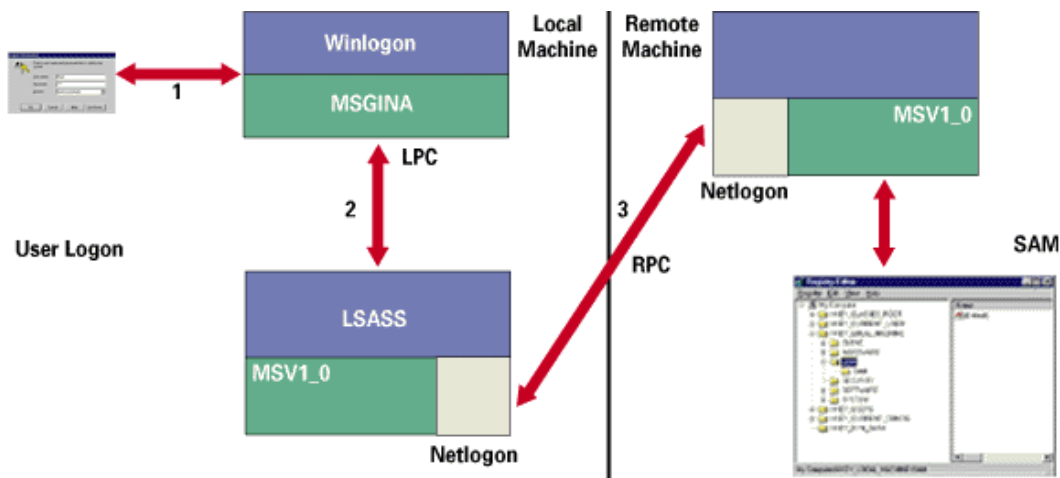
(Reprinted from WindowsItPro Magazine)

under Account\Users\Names. Three subkeys under Account\Users have numeric names. The keys with numeric names correspond directly to the account names under the Names subkey, and the numeric names of these keys are the account RIDs (e.g., the Administrator account has a RID of 000001F4, which is the hexadecimal representation of 500). For the RID-name keys, you'll find two values, F and V, that contain account description, password policy, password (encrypted), and SID. Of course, all this information is in an undocumented format and will change in NT 5.0 (see the sidebar, "Logon in NT 5.0," page 68).

The Builtin subkey is a mirror of the Account subkey, but Builtin also contains information for aliases, groups, and accounts that are built in to NT. Unless you have a nonstandard security setup, you'll find in this subkey only built-in alias information that describes which accounts are members of aliases. For instance, under the Builtin subkey you'll find Administrators, Power Users, Backup Operators, and other aliases.

To validate a user logon, MSV1\_0 looks up the specified logon account in the SAM and compares the password the user keyed in with what it finds in the SAM. If the information matches, MSV1\_0 validates the user, obtains the SIDs of the account and the groups the account belongs to, and returns these SIDs to the interface component, which in turn passes the SIDS back to LSASS. LSASS finally obtains a list of privileges associated with the account and groups. An example privilege is User can reboot the computer. NT stores the privileges lists in HKEY\_LOCAL\_MACHINE\SECURITY. Like the SAM, the security database is thoroughly locked down, but I'll show you what's inside it when I discuss privileges next month.

The flow of logon control is different if the user logon is directed at a domain and the user machine is not a domain controller or if the logon is directed at a trusted domain. Figure 2 shows the flow of remote logon control. The MSV1\_0 interface component presents the username and encrypted password to the processing component, which gives the username and encrypted password to the Netlogon component. Netlogon (winnt\system32\netlogon.dll) sends the information to MSV1\_0 on a server (chosen through a process called discovery) for the domain the logon is targeting; this server also has an instance of MSV1\_0 running on it. Netlogon on the target server is listening for network authentication requests. When it receives a request, it hands the information to the processing component of MSV1\_0. The processing component performs the same authentication and SID-retrieval steps as for local logons, but it gives the information to Netlogon on the target server, which returns the information to the originating computer's Netlogon. Finally, the originating Netlogon returns its results to the interface component of MSV1\_0, which returns the results to LSASS.



# Windows NT Security

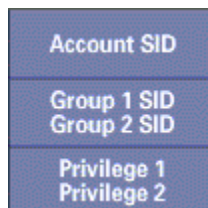
Mark Russinovich

(Reprinted from WindowsItPro Magazine)

I've just described interactive logons, in which a user sits at a computer and types in a username and password, but the flow of control is the same for noninteractive logons. For example, when a Win32 service (a daemon process) runs in the context of a specific user account, the service logs on to the computer by using account name and password information, which the service stores with its other configuration parameters. The difference between interactive and noninteractive logons is that an interactive logon results in the creation of an interactive window station, with a desktop GUI that programs can interact with. You can think of noninteractive logons as "headless" logons.

## Access Tokens

If the interface portion of MSV1\_0 returns a result to LSASS that confirms a successful logon, LSASS must create an access token. An access token stores all the information MSV1\_0 gathered, including the account SID, the group SIDs, the collective privileges, and the account name. An access token is a complete description of a user from the viewpoint of NT security, and NT attaches a copy of the token to all processes that the user executes. For example, when someone performs an interactive logon, Winlogon executes a shell program (in most cases Explorer) and gets the ball rolling by attaching to the shell process the access token that LSASS returned through the GINA. When LSASS creates the first token, the user is officially logged on. If the user starts another program, such as Word, inside Explorer, the process Word is associated with will get a copy of the access token. When LSASS closes the last token belonging to a user, the user is officially logged off. Figure 3 depicts an access token's logical contents. When a process tries to access an object, such as a file, Registry key, named pipe, or even an unnamed synchronization object, the Security Reference Monitor subsystem (discussed in "Windows NT Architecture, Part 1," March 1998) uses the process' access token to determine whether to allow the access.



Next month I'll describe the decision-making process the Security Reference Monitor uses when it processes access objects. I'll show you how server processes such as those a file server initiates can temporarily alter their identities to look like a client user through a mechanism known as impersonation. I'll wrap up this two-part look at NT security by discussing privileges and policies in depth.

Last month, I began a two-part series on Windows NT security by discussing NT's C2 security rating. I discussed the security subsystem's use of security identifiers (SIDs) to identify users, groups, and computers, and I stepped through local and remote logon sequences. I concluded by introducing access tokens. This month, I'll explain tokens in more detail. I'll discuss how impersonation, or the adoption of someone else's token, is a useful and common NT technique. I'll look at security descriptors and access validation, and I'll explain privileges. Then, I'll briefly consider policies and conclude with a preview of NT 5.0's security enhancements.

## Tokens

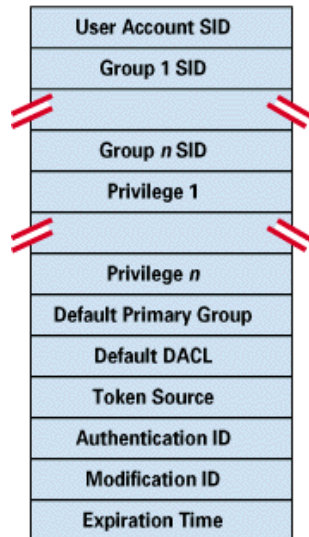
The Security Reference Monitor uses tokens (or access tokens) to identify the security context of a process or thread. A security context consists of information that describes the privileges, accounts, and groups associated with the process or thread. Last month, I described the logon process show NT

# Windows NT Security

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

creates an initial token to represent the user and attaches the token to the user's logon shell. All programs the user executes inherit a copy of the initial token. Tokens vary in size because different user accounts have different sets of privileges and associated group accounts. However, all tokens contain the same basic set of information, as Figure 1, page 60, shows.



NT's security mechanisms use two token components to determine what a token's thread or process can do. One component comprises the token's user account SID and group account SID fields. The Security Reference Monitor uses SIDs to determine whether a process or thread can obtain requested access to a securable object, such as an NTFS file.

The group account SIDs in a token signify which groups a user's account is a member of. A server application can disable specific groups to restrict a token's credentials when the server application is performing actions a client requests. Disabling a group produces the same effect as if the group were not present in the SID.

The second component in a token that determines what the token's thread or process can do is the privilege array. A token's privilege array is a list of rights associated with the token. An example privilege is the right for the process or thread associated with the token to shut down the computer. There are about two dozen token privileges.

A token's Default Primary Group field and Default Discretionary Access Control List (DACL) field are security attributes NT applies to objects that a process or thread creates when it uses the token. By including security information in tokens, NT makes it convenient for a process or thread to create objects with standard security attributes, because the process or thread doesn't need to request discrete security information for every object it creates.

Each token's type distinguishes a primary token (a token that identifies the security context of a process) from an impersonation token (a token that threads use to temporarily adopt a different security context, usually of another user). Impersonation tokens carry an impersonation level that signifies what type of impersonation is active in the token. I'll describe impersonation in more detail shortly.

# Windows NT Security

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

The remainder of the fields in a token serve informational purposes. The Token Source field contains a short textual description of the entity that created the token. Programs that want to know where a token originated use the Token Source to distinguish among sources such as the NT Session Manager, a network file server, or the remote procedure call (RPC) server. The token identifier is a locally unique identifier (LUID) that the Security Reference Monitor assigns to the token when it creates the token. NT's Executive maintains the Executive LUID, a counter it uses to assign a unique numeric identifier to each token.

The token Authentication ID is another kind of LUID. A token's creator assigns the token's Authentication ID. The Local Security Authority Sub System (LSASS) is typically the only token-creator on a system, and LSASS obtains the LUID to assign from the Executive LUID. LSASS then copies the Authentication ID for all tokens descended from an initial logon token. A program can obtain a token's Authentication ID to see whether the token belongs to the same logon session as other tokens the program has examined.

The Executive LUID refreshes the Modification ID every time a token's characteristics are modified. An application can test the Modification ID to discover changes in a security context since the context's last use.

Tokens contain an Expiration Time field that has been present but unused in NT since NT 3.1. Microsoft presumably has plans to allow for tokens that are valid for a period of time before expiring. Consider a user that the systems administrator sets an account expiration time for. Currently, if the user logs on and remains logged on past the account expiration, the system will let the user continue to access resources. The only way to prevent the user from accessing resources is to forcibly log the user off the machine. If NT supported token expiration, the system could prevent the user from opening resources past the token Expiration Time. NT 5.0 doesn't appear to implement token expiration functionality.

## Impersonation

Impersonation is a powerful feature NT uses frequently in its security model. NT also implements impersonation in its client/server programming model. In NT, a server exports resources such as files, printers, or databases. A client wanting to access a resource sends a request to the server. When the server receives the request, it must ensure that the client has permission to perform the desired operations on the resource. For example, if a user on a remote machine tries to delete a file on an NTFS share, the server exporting the share must determine whether the user is allowed to delete the file. The obvious process to determine whether a user has permission is for the server to query the user's account and group SIDs and scan the security attributes on the file. This process is tedious to program, prone to errors, and does not change as NT's model might change. Thus, NT uses impersonation to simplify the server's job.

Impersonation lets a server notify the Security Reference Monitor that the server is temporarily adopting the security profile of a client making a resource request. The server can then access resources on behalf of the client, and the Security Reference Monitor carries out the access validations. Usually, a server has access to more resources than a client does and loses some of its security credentials during impersonation. However, the reverse can be true: The server can gain security credentials during impersonation.

A server impersonates a client only within the thread that makes the impersonation request (for more information about threads and processes, see "Inside the Windows NT Scheduler, Part 1," July 1997). Thread-control data structures contain an entry for a possible impersonation token. However, a

# Windows NT Security

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

thread's primary token, which represents the thread's real security credentials, is always accessible in the process' control structure.

NT makes impersonation available in several forms. If a server communicates with a client through a named pipe, the server can tell the Security Reference Monitor that it wants to impersonate the user on the other end of the pipe. If the server is communicating with the client through Dynamic Data Exchange (DDE) or RPC, it can make a similar impersonation request. After the server thread finishes its task, it reverts to its primary security profile. These forms of impersonation are convenient for carrying out specific actions at the request of a client. The disadvantage to these forms of impersonation is that they cannot execute an entire program in the context of a client. In addition, an impersonation token cannot access files or printers on network shares unless the file or printer share supports null sessions.

If an entire application must execute in a client's security context or must access network resources, the client must be logged on to the system. An NT function, LogonUser, enables this action. LogonUser takes an account name, password, and domain or computer name as input and returns a primary token. A server thread can adopt the token as an impersonation token, or the server can start a program that has the client's credentials as its primary token. From a security standpoint, the process that LogonUser creates to run the program looks exactly like a program a user starts by logging on to the machine interactively.

NT impersonates a client's security context in a second way, which is similar to LogonUser: by taking a client's access token, duplicating it, and using it as the primary token that is passed to the CreateProcessAsUser command. The disadvantage to using the LogonUser and CreateProcessAsUser approaches is that a server must obtain the user's account name and password. If the server transmits this information across the network, the server must encrypt it securely so that a malicious user snooping network traffic can't capture it.

To prevent misuse of impersonation, NT does not let servers perform impersonation without a client's consent. The client of a named pipe, DDE, or RPC can specify the level of impersonation it allows a server. Existing impersonation levels are Anonymous, Identification, Impersonation, and Delegation. Each level lets a server perform different types of operations in the client's name. Anonymous impersonation is the most restrictive level--the server cannot impersonate or identify the client. Identification impersonation lets the server obtain the identity (i.e., the SIDs) of the client and the client's privileges, but the server cannot impersonate the client. The Impersonation level lets the server identify and impersonate the client. Finally, Delegation impersonation, the most permissive level of impersonation, lets the server impersonate the client on local and remote systems. NT doesn't fully support Delegation impersonation in versions before NT 5.0. If the client does not set an impersonation level, NT chooses the Impersonation level by default.

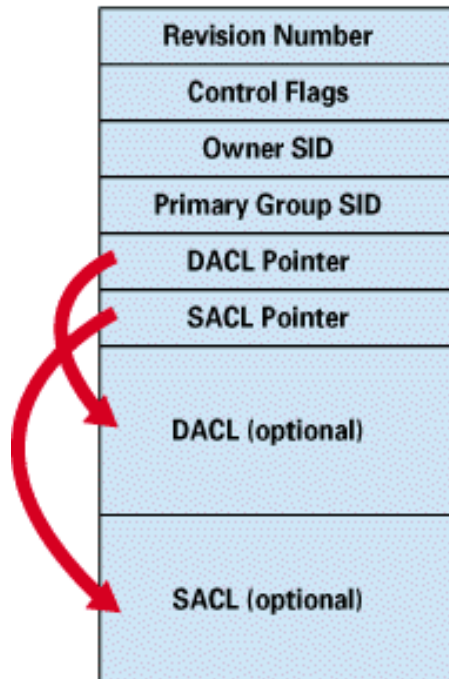
## Security Descriptors

Tokens, which identify a user's credentials, are only half of the object security equation. The other half of the equation is the information associated with an object, which specifies who can perform what actions on the object. The data structure for this information is called a security descriptor, which Figure 2 shows. The descriptor has a header that contains the descriptor's revision number, and control flags that specify attributes of the descriptor, such as what memory layout the descriptor uses. The security descriptor contains the SID of the object's owner, which can be for a local or domain user or group account. Following the object owner's SID is the SID of the group that is the main, or primary, group of the object. A security descriptor can have two attached optional lists: a DACL and a System Access Control List (SACL).

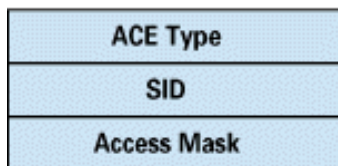
# Windows NT Security

Mark Russinovich

(Reprinted from WindowsItPro Magazine)



The security descriptor's optional DACL is a list of zero or more access control entries (ACEs). A DACL uses its ACEs to describe the actions a particular user or group can perform on the object. An ACE, as Figure 3 shows, has a header that specifies whether the ACE allows or denies accesses, a SID that identifies a user or a group, and a mask that lists the ACE's accesses.



When a user opens an object, the user specifies what actions to perform on the object (e.g., delete, read, or write). If the object is a secured object (i.e., a security descriptor is associated with it), it calls the Security Reference Monitor to determine whether the open is allowed.

Two special cases relate to access. In the first case, if the security descriptor doesn't have a DACL, the Security Reference Monitor allows any open. In the second case, if the descriptor has a DACL but the DACL has no ACEs, the Security Reference Monitor doesn't allow any access to the object.

In the case of a DACL with ACEs, the Security Reference Monitor scans the ACEs looking for one with a SID that matches any of the enabled SIDs in the user's access token (regardless of whether it's a primary or an impersonation token). If the Security Reference Monitor finds such a SID, and the SID's ACE is a deny ACE, the open will fail if the user is requesting any of the accesses listed in the ACE. If the ACE is an allow ACE, the Security Reference Monitor matches the accesses in the ACE to the accesses the user is requesting. If an explicit allow ACE does not match at least one of the user's requested accesses, the Security Reference Monitor continues scanning the ACEs. If allow ACEs match all the user's accesses, the open succeeds. If the Security Reference Monitor processes all the

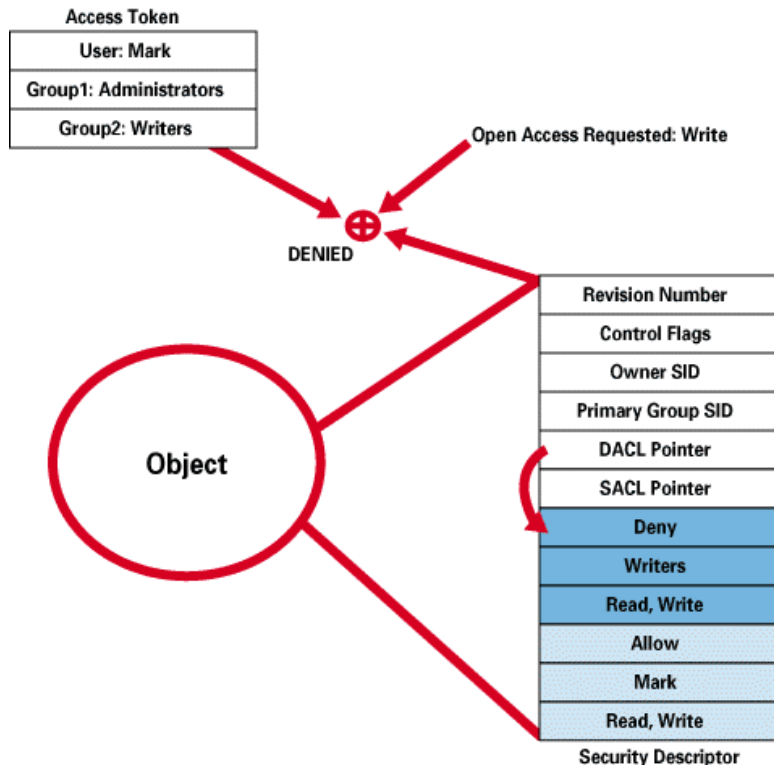
# Windows NT Security

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

ACEs without granting all of the access types the user requested, it denies the user access to the object.

The process I just explained is illustrated in Figure 4, page 62. User Mark attempts to open an object and requests permission to write to the object. When the Security Reference Monitor looks through the ACEs in the object's DACL, it comes across an ACE that denies members of the Writers group (of which Mark is a member) write access. Thus, even though a subsequent ACE allows Mark write access to the object, the Security Reference Monitor denies Mark access, and the open fails.



A special case is that a user who owns an object (i.e., the user SID matches the owner SID in the security descriptor) or belongs to the group that owns an object can always open the object to modify its security descriptor. Thus, an object's owner can always access an object to modify the security descriptor, even if the descriptor denies the owner all access.

A SACL is constructed similarly to a DACL, except that instead of identifying allowed and denied accesses, the SACL specifies which actions particular users or groups perform that cause NT to write audit events to the NT Security Log. The header of a SACL's ACE shows whether the ACE is of the success or failure type. The header's mask lists accesses that generate events when the user or group identified with the SID performs the access. NT uses SACLS only if an administrator has enabled auditing. Administrators usually disable auditing because SACL processing and security event record logging can introduce significant system overhead.

With auditing enabled, the Security Reference Monitor scans a SACL after a user opens an object associated with the SACL. The Security Reference Monitor looks for a match between the SIDs of the user's enabled groups and a SID in one of the SACL's ACEs combined with the user's SID. If the SIDs match, the Security Reference Monitor checks the ACE type to see whether it matches the result of the open (e.g., the ACE is a succeed ACE, and the open succeeded). If there is a match, the Security

# Windows NT Security

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

Reference Monitor makes a final check to see whether the access requested in the open matches any accesses in the ACE's mask (e.g., the user requested read access, and the read bit is set in the mask). A match at this point results in NT logging the event to the Security Log. Thus, SACLs monitor and record actions users perform on critical resources.

NT uses security descriptors to secure objects for several standard NT components, such as NTFS, the Registry, and the named pipe file system. However, applications written for NT can secure their private objects with security descriptors as well. An application is responsible for maintaining object-to-security-descriptor relationships and calling the Security Reference Monitor to perform access validation.

## Privileges

A privilege is the right a user has to perform a system-related action, such as load a device driver, shut down the computer, or open files for backup purposes. Built-in accounts and groups have default sets of privileges. For example, accounts that belong to the Backup Operators group have the Backup privilege for backing up files. Administrators can use User Manager to add or remove specific privileges from user or group accounts. Table 1, page 64, lists the privileges NT 5.0 beta 1 defines.

**TABLE 1: Privileges in NT 5.0 Beta 1**

Privilege	Description
ASSIGNPRIMARYTOKEN	Required to assign the primary token of a process.
AUDIT	Required to generate audit-log entries. Give this privilege to secure servers.
BACKUP	Required to perform backup operations.
CHANGE_NOTIFY	Required to receive notifications of changes to files or directories. This privilege also causes the system to skip all traversal access checks. NT enables it by default for all users.
CREATE_PAGEFILE	Required to create a paging file.
CREATE_PERMANENT	Required to create a permanent object.
CREATE_TOKEN	Required to create a primary token.
DEBUG	Required to debug a process.
INC_BAPRIORITY	Required to increase the base priority of a process.
INCREAQUOTA	Required to increase the quota assigned to a process.
LOAD_DRIVER	Required to load or unload a device driver.
LOCK_MEMORY	Required to lock physical pages in memory.
MACHINE_ACCOUNT	Required to create a machine account.
PROF_SINGLE_PROCESS	Required to gather profiling information for a single process.
REMOTE_SHUTDOWN	Required to shut down a system using a network request.
RESTORE	Required to perform restore operations. This privilege enables you to set any valid user or group SID as the owner of an object.
SECURITY	Required to perform several security-related functions, such as controlling and viewing audit messages. This privilege identifies its

# Windows NT Security

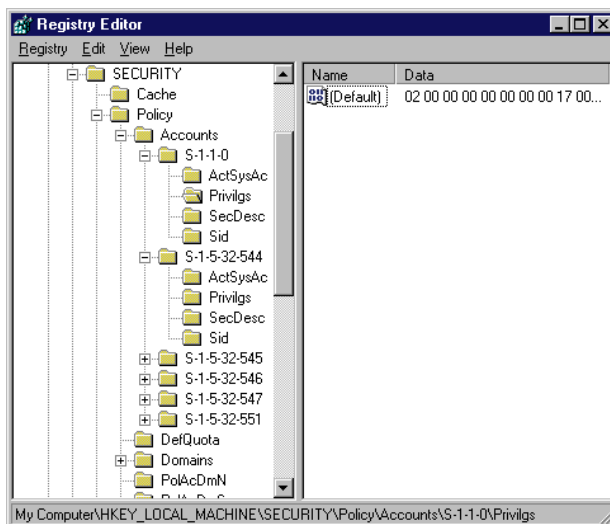
Mark Russinovich  
(Reprinted from WindowsItPro Magazine)

	holder as a security operator.
SHUTDOWN	Required to shut down a local system.
SYSTEM_ENVIRONMENT	Required to modify the nonvolatile RAM of systems that use this type of memory to store configuration information.
SYSTEM_PROFILE	Required to gather profiling information for the entire system.
SYSTEMTIME	Required to modify the system time.
TAKE_OWNERSHIP	Required to take ownership of an object without being granted discretionary access. This privilege lets the owner value be set only to those values that the holder may assign as the owner of an object.
TCB	Required to call the LogonUser function. This privilege identifies its holder as part of the trusted computer base. NT grants this privilege to some trusted protected subsystems.

Each privilege contains an LUID that identifies the privilege. Microsoft documentation states that the LUID for a given privilege can be different across different computers or from boot to boot. However, privilege LUIDs are hard-coded into NT so that each LUID is guaranteed to be identical on all systems running the same version of NT. In future releases of NT, applications will be able to define additional privileges.

By default, NT disables all the privileges (except the CHANGE\_NOTIFY privilege, which NT enables on all accounts) on privilege arrays associated with tokens. Disabling these privileges prevents an application from performing an operation accidentally. For instance, a program must enable the SHUTDOWN privilege before it can shut down the system.

I mentioned last month that I would go inside the SECURITY Registry key to show you what's hiding there. Screen 1 shows that under the Policy subkey you'll find subkeys for each local user and group account. The account keys, two of which are open in the figure, include the account's SID, the default security descriptor, and the list of the account's privileges. Other data in the SECURITY key includes various policy information and domain account definitions if the machine is a domain controller. The SECURITY key's layout is undocumented and subject to change at any time.



# Windows NT Security

Mark Russinovich

(Reprinted from WindowsItPro Magazine)

## Policies

Policies, a feature new to NT 4.0, aren't really a part of NT's security model. Individual applications and components define policies. Policies are similar to privileges in that they are rights granted to user accounts. For example, NT Explorer has a set of policies that administrators can control to allow or deny users the ability to move icons around their desktops, set their desktop backgrounds, or have a Run entry on their Start menus.

The System Policy Editor (SPE) is a tool administrators use to create policy profiles and apply them to specific machines or across a domain. NT stores policy values as Registry keys. For example, many of Explorer's policies are in the:

`HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies`

key. When NT Explorer starts, it reads the values of its policies and allows only specified rights.

## NT 5.0 Security

There are three enhancements to NT's security model in NT 5.0. The first enhancement is the ability to control access to securable objects that operating systems (OSs) other than NT create. Objects from OSs other than NT have definitions for access rights. For instance, read access for a directory in a UNIX file system doesn't map neatly to the same access on NT. NT 5.0 introduces the idea of provider-independent access rights. When an application identifies that it's dealing with foreign objects, it can implement provider-independent access rights and use new NT APIs to map the foreign rights to corresponding NT rights. Provider-independent access rights remove the burden of translating from applications.

The second NT 5.0 security enhancement extends ACLs to allow per-property granularity in an ACL. This enhancement lets a single security descriptor have a DACL or SACL that refers to different properties of an object. For instance, one ACE of a word processor document file's DACL could allow changes to the text, and another ACE could deny the ability to change the document's author. NT 5.0 will also let you specify which ACEs of a parent object's security descriptor will be inherited by the object's children. NT 5.0 considers files to be children of the directory in which they are created, for instance.

The third NT 5.0 security enhancement is an API that creates a new token from an existing token but with reduced privileges or groups. For example, suppose a server wants to impersonate a client by starting a program on the client's behalf, but the server wants to make sure the program cannot shut down the machine, even though the client's security profile possesses the shutdown privilege. The server creates a token identical to the client's token except without the shutdown privilege. If a group is disabled in a restricted token, NT considers the group to be absent for the purpose of granting access but present when NT considers denying access. Thus, removing a group from a token doesn't allow access to objects that aren't otherwise accessible.

## More Information

I've covered a lot of ground very quickly. Many NT security subtleties exist that I don't have the space to explain, but you have a solid foundation of how the NT security model works. If you want more detailed information, refer to the Microsoft Windows NT Workstation 4.0 Resource Kit. If you're interested in the intricacies of the security model, read the Microsoft Developer's Network documentation on NT security APIs.