CHAPTER 24

WINDOWS AND WINDOWS VISTA SECURITY

by Michael Howard
Principal Security Program Manager, Microsoft Corporation

24.1 **Windows Security Architecture**

- The Security Reference Monitor
- The Local Security Authority
- The Security Account Manager
- Active Directory
- Windows Security Basics: An End-to-End Domain Example
- Privileges in Windows
- Access Control Lists
- Access Checks
- Impersonation
- Mandatory Access Control

24.2 **Windows Vulnerabilities**

24.3 **Windows Security Defenses**

- Windows System Hardening Overview
- Account Defenses
- Network Defenses
- Buffer Overrun Defenses

24.4 **Browser Defenses**

24.5 **Cryptographic Services**

- Encrypting File System
- Data Protection API
- BitLocker
- TPM

24.6 **Common Criteria**

24.7 **Recommended Reading and Web Site**

24.8 **Key Terms, Review Questions, Problems, and Projects**

Windows is the world's most popular operating system and as such has a number of interesting security-related advantages and challenges. The major advantage is any security advancement made to Windows can protect hundreds of millions of nontechnical users, and advances in security technologies can be used by thousands of corporations to secure their assets. The challenges for Microsoft are many, including the fact that security vulnerabilities in Windows can affect millions of users. Of course, there is nothing unique about Windows having security vulnerabilities; all software products have security bugs. However, Windows is used by so many nontechnical users that Microsoft has some interesting engineering challenges.

This chapter begins with a description the overall security architecture of Windows 2000 and later (Section 24.1). It is important to point out that versions of Windows based on the Windows 95 code base, including Windows 98, Windows 98 SE, and Windows Me, had no real security model, in contrast to Windows NT and later versions. The Windows 9x codebase is no longer supported.

The remainder of the chapter cover the security defenses built into Windows, most notably the new defenses in Windows Vista.

24.1 WINDOWS SECURITY ARCHITECTURE

Anyone who wants to understand Windows security must have knowledge of the basic fundamental security blocks in the operating system. There are many important components in Windows that make up the fundamental security infrastructure, among them the following:

- The Security Reference Monitor (SRM)
- The Local Security Authority (LSA)
- The Security Account Manager (SAM)
- Active Directory (AD)
- Authentication Packages
- WinLogon and NetLogon

Let's look at each in detail.

The Security Reference Monitor

This kernel-mode component performs access checks, generates audit log entries, and manipulates user rights, also called privileges. Ultimately, every permission check is performed by the SRM.

The Local Security Authority

The Local Security Authority resides in a user-mode process named lsass.exe and is responsible for enforcing local security policy in Windows. It also issues security tokens to accounts as they log on to the system. Security policy includes password policy (such as complexity rules and expiration times), auditing policy, and privilege settings.

The Security Account Manager

The SAM is a database that stores user accounts and relevant security information about local users and local groups. Note the term *local*. Windows has the notion of local and domain accounts. We will explain more about this subsequently, but for now note that Windows users can log on to a computer using either accounts that are known only on that particular computer or accounts that are managed centrally. When a user logs on to a computer using a local account, the SAM process (SamSrv) takes the logon information and performs a lookup against the SAM database, which resides in the Windows System32\config directory. If you're familiar with UNIX, think /etc/passwd (or similar). If the credentials match, then the user can log on to the system, assuming there are no other factors preventing logon, such as logon time restrictions or privilege issues, which we discuss later in this chapter. Note that the SAM does not perform the logon; that is the job of the LSA. The SAM file is binary rather than text, and passwords are stored using the MD4 hash algorithm. On Windows Vista, the SAM stores password information using a password-based key derivation function (PBKCS).

Note that WinLogon handles local logons at the keyboard and NetLogon handles logons across the network.

Active Directory

Active Directory is Microsoft's LDAP directory included with Windows Server 2000 and later. All client versions of Windows, including Windows XP and Windows Vista, can communicate with AD to perform security operations including account logon. A Windows client will authenticate using AD when the user logs on to the computer using a domain account rather than a local account. Like the SAM scenario, the user's credential information is sent securely across the network, verified by AD, and then, if the information is correct, the user can logon.

Local versus Domain Accounts We used the terms *local* and *domain*. A networked Windows computer can be in one of two configurations, either domain joined or in a workgroup. When a computer is domain joined, users can gain access to that computer using domain accounts, which are centrally managed in Active Directory. They can, if they wish, also log on using local accounts, but local accounts may not have access to domain resources such as networked printers, Web servers, e-mail servers, and so on. When a computer is in a workgroup, only local accounts can be used, held in the SAM. There are pros and cons to each scenario. A domain has the major advantage of being centrally managed and as such is much more secure. If an environment has 1000 Windows computers and an employee leaves, the user's account can be disabled centrally rather than on 1000 individual computers. The only advantage of using local accounts is that a computer does not need the infrastructure required to support a domain using AD.

Windows also has the notion of a workgroup, which is simply a collection of computers connected to one another using a network; but rather than using a central database of accounts in AD, the machines use only local accounts. The difference between a workgroup and a domain is simply where accounts are authenticated. A workgroup has no domain controllers; authentication is performed on each computer, and a domain authenticates accounts at domain controllers running AD.

Windows Security Basics: An End-to-End Domain Example

Now that you know the basic elements that make up the core Windows security infrastructure, we give an example of what happens when a user logs on to a Windows system.

Before a user can log on to a Windows network, a domain administrator must add the user's account information to the system; this will include the user's name, account name (which must be unique within the domain), and password. Optionally, the administrator can grant group membership and privileges.

After the administrator has entered the user's account information, Windows creates an account for the user in the domain controller running Active Directory. Each user account is uniquely represented by a Security ID (SID). SIDs are unique within a domain, and every account gets a different SID. This is an important point. If you create an account named Blake, delete the account, and "re-create" the account named Blake, they are in fact two totally different accounts because they will have different SIDs.

A user account's SID is of the following form:

- S-1–5–21-AAA-BBB-CCC-RRR.
- S simple means SID.
- 1 is the SID version number.
- 5 is the identifier authority; in this example, 5 is SECURITY_NT_AUTHORITY.
- 21 means "not unique," which just means there is no guarantee of uniqueness; however, a SID is unique within a domain, as you'll see in a moment.
- AAA-BBB-CCC is a unique number representing the domain.
- RRR is called a relative ID (RID); it's a number that increments by 1 as each new account is created. RIDs are never repeated; this is what makes each SID unique.

For example, a SID might look like this:

```
S-1-5-21-123625317-425641126-188346712-2895
```

In Windows, a username can be in one of two formats. The first, named the SAM format, is supported by all versions of Windows and is of the form DOMAIN\Username. The second is called User Principal Name (UPN) and looks more like an RFC822 e-mail address: username@domain.company.com. The SAM name should be considered a legacy format.

When a user logs on to Windows, he or she does so using either a username and password, or a username and a smart card. It is possible to use other authentication or identification mechanisms, such as an RSA SecureID token or biometric device, but these require third-party support.

Assuming the user logs on correctly, a token is generated by the operating system and assigned to the user. A token contains the user's SID, group membership information, and privileges. Groups are also represented using SIDs. We explain privileges subsequently. The user's token is assigned to every process run by the user. It is used to perform access checks, discussed subsequently.

Privileges in Windows

Privileges are essentially systemwide permissions assigned to user accounts. Examples of Windows privileges include the ability to back up the computer, or the ability to change the system time. Performing a backup is privileged because it bypasses all access checks so a complete backup can be preformed. Likewise, setting the system time is privileged because changing the time can make Kerberos authentication fail and lead to erroneous data being written to the logging system. There are over 45 privileges in Windows Vista. Some privileges are deemed "dangerous," which means a malicious account that is granted such a privilege can cause damage. Examples of such privileges include the following:

- **Act as part of operating system privilege.** This is often referred to as the Trusted Computing Base (TCB) privilege, because it allows code run by an account granted this privilege to act as part of the most trusted code in the operating system: the security code. This is the most dangerous privilege in Windows and is granted only the Local System account; even administrators are not granted this privilege.
- **Debug programs privilege.** This privilege allows an account to debug any process running in Windows. A user account does not need this privilege to debug an application running under the user's account. Because of the nature of debuggers, this privilege basically means a user can run any code he or she wants in any running process.
- **Backup files and directories privilege.** Any process running with this privilege will bypass all access control list (ACL) checks, because the process must be able to read all files to build a complete backup. Its sister privilege Restore files and directories is just as dangerous because it will ignore ACL checks when copying files to source media.

Some privileges are generally deemed benign. An example is the "bypass traverse checking" privilege that is used to traverse directory trees even though the user may not have permissions on the traversed directory. This privilege is assigned to all user accounts by default and is used as an NTFS file system optimization.

Access Control Lists

Windows has two forms of access control list (ACL). The first is called a Discretionary ACL (DACL) is usually what most people mean when they say *ACL*. A DACL grants or denies access to protected resources in Windows such as files, shared memory, named pipes, and so on. The other kind of ACL is the System ACL (ACL), which is used for auditing and in Windows Vista used to enforce mandatory integrity policy. Let's take a moment to look at the DACL.

Objects that require protection are assigned a DACL (and possible a SACL), which includes the SID of the object owner (usually the object creator) as well as a list of access control entries (ACEs). Each ACE includes a SID and an access mask. An access mask could include the ability to read, write, create, delete, modify, and so on. Note that access masks are object-type specific; for example, services (the Windows equivalent of UNIX daemons) are protected objects and support an

access mask to create a service (SC_MANAGER_CREATE_SERVICE) and a mask that allows service enumeration (SC_MANAGER_ENUMERATE_SERVICE). The data structure that includes the object owner, DACL, and SACL is referred to as a the object's security descriptor (SD).

A sample SD with no SACL is as follows:

```
Owner: CORP\Blake
ACE[0]: Allow CORP\Paige Full Control
ACE[1]: Allow Administrators Full Control
ACE[2]: Allow CORP\Cheryl Read, Write and Delete
```

The DACL in this SD allows the user named Paige (from the CORP domain) full access to the object; she can do anything to this object. Members of the Administrators can do likewise. Cheryl can read, write, and delete the object. Note that the object owner is Blake; as the owner, he can do anything to the object, too. This was always the case until the release of Windows Vista. Some customers do not want owners to have such unbridled access to objects, even though they created them. In Windows Vista you can include an Owner SID in the DACL, and the access mask associated with that account applies to the object owner.

There are two important things to keep in mind about access control in Windows. First, if the user accesses an object with the SD example above, and the user is not Blake, not Paige, not Cheryl, and not a member of the Administrator's group, then that user is denied access to the object. There is no implied access. Second, if Cheryl requests read access to the object, she is granted read access. If she requests read and write access, she is also granted access. If she requests create access, she is denied access unless Cheryl is also a member of the Administrators group, because the "Cheryl ACE" does not include the "create" access mask. The last point is critically important. When a Windows application accesses an object, it must request the type of access the application requires. Many developers would simply request "all access" when in fact the application may only want to read the object. If Cheryl uses an application that attempts to access the object described above and the application requests full access to the object, she is denied access to the object unless she is an administrator. This is the prime reason so many applications failed to execute correctly on Windows XP unless the user is a member of the Administrator's group.

We mentioned earlier that a DACL grants or denies access; technically this is not 100% accurate. Each ACE in the DACL determines access; and an ACE can be an allow ACE or a deny ACE. Look at this variant of the previous SD:

```
Owner: CORP\Blake
ACE[0]: Deny Guests Full Control
ACE[1]: Allow CORP\Paige Full Control
ACE[2]: Allow Administrators Full Control
ACE[3]: Allow CORP\Cheryl Read, Write and Delete
```

Note that the first ACE is set to deny members of the guests account full control to the object. Basically, guests are out of luck if they attempt to access the object

protected by this SD. Deny ACEs are not often used in Windows because they can be complicated to troubleshoot. Also note that the first ACE is the deny ACE; it is important that deny ACEs come before allow ACEs because Windows evaluates each ACE in the ACL until access is granted or explicitly denied. If the ACL grants access, then Windows will stop ACL evaluation, and if the deny ACE is at the end of the ACL, then it is not evaluated, so the user is granted access even if the account may be denied access. When setting an ACL from the user interface, Windows will always put deny ACEs before allow ACEs, but if you create an ACL programmatically (for example, by using the *SetSecurityDescriptorDacl* function), you must explicitly place the deny ACEs first.

Access Checks

It's now time to put this all together. When a user account attempts to access a protected object, the operating system performs an access check. It does this by comparing the user account and group information in the user's token and the ACEs in the object's ACL. If all the requested operations (read, write, delete, and so on) are granted, then access is granted; otherwise the user gets an access denied error status (error value 5).

Impersonation

There is one last thing you should understand about Windows. Windows is a multithreaded operating system, which means a single process can have more than one thread of execution at a time. This is very common for both server and client applications. For example, a word processor might have one thread accepting user input and another performing a background spellcheck. A server application, such as a database server, might start a large number of threads to handle concurrent user requests. Let's say the database server process runs as a predefined account named DB_ACCOUNT; when it takes a user request, the application can impersonate the calling user by calling an impersonation function. For example, one networking protocol supported by Windows is called Named Pipes, and the *ImpersonateNamedPipeClient* function will impersonate the caller. Impersonation means setting the user's token on the current thread. Normally, access checks are performed against the process token, but when a thread is impersonating a user, the user's token is assigned to the thread, and the access check for that thread is performed against the token on the thread, not the process token. When the connection is done, the thread "reverts," which means the token is dropped from the thread.

So why impersonate? Imagine if the database server accesses a file named db.txt, and the DB_ACCOUNT account has read, write, delete, and update permission on the file. Without impersonation, any user could potentially read, write, delete, and update the file. With impersonation, it is possible to restrict who can do what to the db.txt file.

Mandatory Access Control

Windows Vista includes a new authorization technology named Integrity Control, which goes one step beyond DACLs. DACLs allow fine-grained access control, but integrity controls limit operations that might change the state of an object. The general premise behind integrity controls is simple; objects (such as files and processes) and principals (users) are labeled with one of the following integrity levels:

- Low integrity (S-1-16-4096)
- Medium integrity (S-1-16-8192)
- High integrity (S-1-16-12288)
- System integrity (S-1-16-16384)

Note the SIDs after the integrity levels. Microsoft implemented integrity levels using SIDs. For example, a high-integrity process will include the S-1-16-12288 SID in the process token. If a subject or object does not include an integrity label, then the subject or object is deemed medium integrity.

The screen shot of Figure 24.1 shows a normal user token in Windows Vista. It includes medium-integrity SID, which means this user account is medium

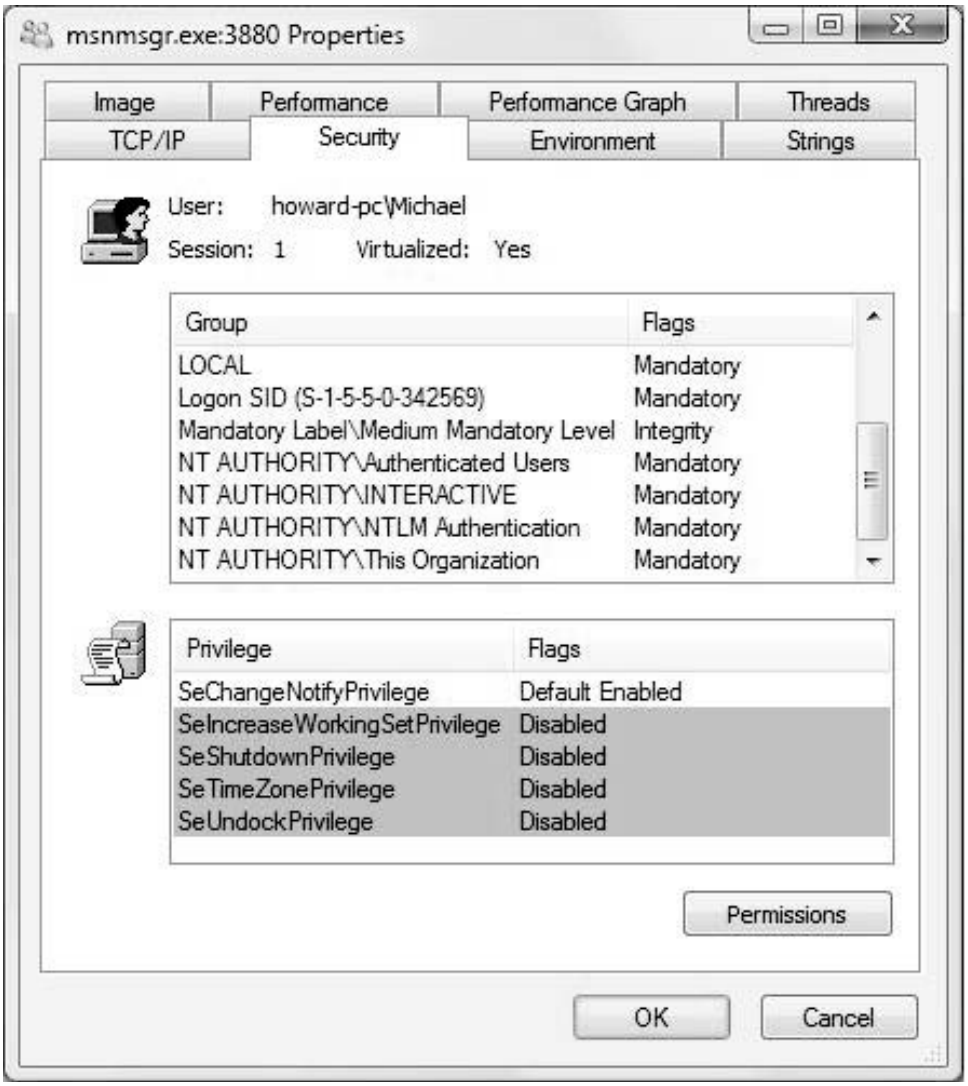

Figure 24.1 **Screen shot of User Account in Windows Vista**

integrity and any process run by this user can write only to objects of medium and lower integrity.

When a write operation occurs, Windows Visa will first checks to see if the subject's integrity level dominates the object's integrity level, which means the subject's integrity level is equal to or above the object's integrity level. If it is, and the normal DACL check succeeds, then the write operation is granted. The most important component in Windows Vista that uses integrity controls is Internet Explorer 7.0; but the majority of the operating system is marked medium or higher integrity.

That completes this whirlwind tour of Windows security principles. Now let's shift focus to security defenses within Windows, most notably Windows Vista.

24.2 WINDOWS VULNERABILITIES

Windows, like all operating systems, has security bugs, and a number of these bugs have been exploited by attackers to compromise customer operating systems. After 2001, Microsoft decided to change its software development process to better accommodate secure design, coding, testing, and maintenance requirements, with one goal in mind: reduce the number of vulnerabilities in all Microsoft products. This process improvement is called the Security Development Lifecycle [HOWA06]. The core SDL requirements are as follows:

- Mandatory security education
- Secure design requirements
- Threat modeling
- Attack surface analysis and reduction
- Secure coding requirements and tools
- Secure testing requirements and tools
- Security push
- Final security review
- Security response

A full explanation of SDL is beyond the scope of this chapter, but the net effect has been an approximately 50% reduction in security bugs. Windows Vista is the first version of Windows to have undergone SDL from start to finish. Other versions of Windows had a taste of SDL, such as Windows XP SP2, but Windows XP predates the introduction of SDL at Microsoft.

SDL does not equate to "bug free" and the process is certainly not perfect, but there have been some major SDL success stories. Microsoft's Web server, Internet Information Services (IIS), has a much-maligned reputation because of serious bugs found in the product that led to worms, such as CodeRed. IIS version 6, included with Windows Server 2003, has had a stellar security track record since its release; there have only been three reported vulnerabilities in the four years since its release, none of them critical. And this figure is an order of magnitude less bugs than IIS's main competitor, Apache [HOWA04].

Another example of SDL working is Microsoft's database server, SQL Server 2000 SP3 and SQL Server 2005. At the time of writing, there have no reported security vulnerabilities in SQL Server 2005, and only two in SQL Server 2000 SP3. When compared to SQL Server's major competitor "Unbreakable Oracle," this is a significant engineering feat.

The most visible part of any vendor's security process is patch management, and Microsoft has substantially fine-tuned the security update process over the last few years. At first, Microsoft issued security updates as soon as they were ready, but now Microsoft issues security updates the second Tuesday of each month. This day is now affectionately referred to as "Patch Tuesday." More recently, Microsoft introduced a novel idea; the Thursday before the second Tuesday, Microsoft announces how many security updates will be shipped, for which products, and what the highest severity rating will be. This streamlined security update process gives system administrators some much-needed predictability to their busy schedules.

24.3 WINDOWS SECURITY DEFENSES

This section and the next will focus on defenses within Windows. The defenses can be grouped into four broad categories:

- Account defenses
- Network defenses
- Buffer overrun defenses.
- Browser defenses.

We discuss each in detail, most notably as each relates to Windows Vista.

All versions of Windows offer security defenses, but the list of defenses has grown rapidly in the last five years to accommodate increased Internet-based threats. The attackers today are not just kids; they are criminals who see money in compromised computers. A zombie network comprised of a few thousand computers under the control of an attacker could be trained on an e-commerce site for a few hours, effectively knocking it off the Internet and losing sales and potentially customers. The attack stops when the extortion money is paid. Again, we want to stress that attacks and compromises are very real, and the attackers are highly motivated by money. Attackers are no longer just young, anarchic miscreants; attackers are real criminals.

Before we discuss security defenses, we discuss system hardening, which is critical to the defensive posture of a computer system and network.

Windows System Hardening Overview

The process of hardening is the process of shoring up defenses, reducing the amount of functionality exposed to untrusted users, and disabling less-used features. At Microsoft, this process is called Attack Surface Reduction. The concept is simple: Apply the 80/20 rule to features. If the feature is not used by 80% of the population, then the feature should be disabled by default. While this is the goal, it is not always achievable simply because disabling vast amounts of functionality

makes the product unusable for nontechnical users, which leads to increased support calls and customer frustration. One of the simplest and effective ways to reduce attack surface is to replace anonymous networking protocols with authenticated networking protocols. The biggest change of this nature in Windows XP SP2 was to change all anonymous remote procedure call (RPC) access to require authentication. This was a direct result of the Blaster worm. Worms spread anonymously, and making this simple change to RPC will help prevent worms that take advantage of vulnerabilities in RPC code and code that uses RPC. It turns out that, in practice, requiring authentication is a very good defense; the Zotob worm, which took advantage of a vulnerability in Plug 'n' Play and which was accessible through RPC, did not affect Windows XP SP2, even the coding bug was there, because an attacker must be authenticated first. But perhaps the beauty of using authentication to reduce attack surface is that most users don't even know it's there, yet the user is protected.

Another example of hardening Windows occurred in Windows Server 2003, which was released in April 2003. Because Windows Server 2003 is a server and not a client platform, the Web browser Internet Explorer was stripped of all mobile code support by default.

In general, hardening servers is easier than hardening clients, for the following reasons:

1. Servers tend to be used for very specific and controlled purposes, while client computers are more general purpose.
2. Whether it's true or not, the perception is that server users are administrators and have more computer configuration skill than a typical client computer user.

Account Defenses

As noted earlier in this chapter, user accounts can contain highly privileged SIDs (such as the Administrators or Account operators groups) and dangerous privileges (such as Act as part of operating system), and malicious software running with these SIDs or privileges can wreak havoc. The principle of least privilege dictates that users should operate with just enough privilege to get the tasks done, and no more. Historically, Windows XP users operated by default as members of the local Administrators group; this was done simply for application compatibility reasons. Many applications that used to run on Windows 95, 98, and Me would not run correctly on Windows XP unless the user was an administrator. In other words, in some cases a Windows XP user running as a "Standard User" could run into some errors. Of course, there is nothing stopping a user from running as a "Standard User."

Windows XP and Windows Server 2003 add a new feature named "Secondary Logon," which allows a user account to right click an application, select "Run as . . . ," and then enter another user account and password to run the application. Windows XP and Windows Server 2003 also include support for another way to reduce privilege on a per-thread level, called a restricted token. A restricted token is simply a thread token with privileges removed and/or SIDs marked as deny-only SIDs. You can learn more about restricted tokens and how to use them programmatically or through Windows Policy [HOWA04].

Windows Vista changes the default; all user accounts are users and not administrators. This is referred to as User Account Control (UAC.)

When a user wants to perform a privileged operation, the user is prompted to enter an administrator's account name and password. If the user is an administrator, the user is prompted to consent to the operation. The reason for doing this is if malware attempts to perform a privileged task, the user is notified. Note that in the case of Windows "Longhorn" Server, the successor to Windows Server 2003, if a user enters a command in the Run dialog box from the Start menu, the command will always run elevated if the user is normally an administrator and will not prompt the user. The great amount of user interaction required to perform these privileged operations mitigates the threat of malware performing tasks off the Run dialog box.

Low Privilege Service Accounts Windows services are long-lived processes that usually start when the computer boots. Examples include the File and Print service and the DNS service. Many such services run with elevated privileges because they perform privileged operations. It is true, however, that many services do not need such elevated requirements, and in Windows XP, Microsoft added two new service accounts, the Local Service account and the Network service account, which allow a service local or network access, respectively, but processes running with these accounts operate at a much lower privilege level. Note that unlike the system account, neither the local service nor the network service accounts are members of the local administrator's group.

In Windows XP SP2, Microsoft made an important change to the RPC service (RPCSS) as an outcome of the Blaster worm. In versions of Windows prior to Windows XP SP2, RPCSS ran as the System account, the most privileged account in Windows. For Windows XP SP2, a major architectural change was made; RPCSS was split in two. The reason RPCSS ran with System identity was simply to allow it to execute Distributed COM (DCOM, which layered on top of RPC) objects on a remote computer correctly, but raw RPC traffic does not require such elevated privileges. So RPCSS was re-architected into components, RPCSS shed its DCOM activation code, and a new service was created called the DCOM Server Process Launcher. RPCSS runs as the lower-privilege Network service account; DCOM runs as SYSTEM. This is a good example of the principle of least privilege in action. Apache, OpenSSH, and Internet Information Services (IIS) 6 and later also use this model. A small amount of code runs with elevated identity, and related components run with lower identity. In the case of Apache on Linux, the initial httpd daemon runs as root because it must open port 80; once the port is open httpd spawns 'worker' httpd dameons as lower-privilege accounts such as nobody or apache. It is these worker processes that receive potentially malicious input. IIS6 follows a similar model, a process named inetinfo starts under the System identity because it must perform administrative tasks, and it starts worker processes named w3wp.exe to handle user requests (these processes run under the lower-privilege network service identity).

Stripping Privileges Another useful defense, albeit not often used in Windows, is to strip privileges from an account when the application starts. This should be performed very early in the application startup code (for example, early in the application's *main* function). The best way to describe this is by way

of example. In Windows, the Index server process runs as the system account because it needs administrative access to all disk volumes to determine if any file has changed so it can reindex the file. Only members of the local Administrators group can get a volume handle. This is the sole reason Index server must run as the system account; yet as you will remember, the system account is bristling with dangerous privileges, such as the TCB privilege and backup privilege. So when the main index server process starts (cidaemon.exe), it sheds any unneeded privileges as soon as possible. The function that performs this is *AdjustTokenPrivileges*.

Windows Vista also adds a function to define the set of privileges required by a service to run correctly. This function that performs this is *ChangeServiceConfig2*.

That ends the overview of core user-account-related security defenses and technologies. Now let's switch focus to network defenses.

Network Defenses

There is one big problem with defenses that focus on the user and user accounts: They do nothing to protect computers from low-level network attacks. The author finds it interesting that so many users and industry pundits focus on "users-as-non-admins" and sometimes lose sight of attacks that do not require human interaction. No user confirmation, no user-based least-privilege defense will protect a computer from an attack that takes advantage of a vulnerability in a network facing process that has no user interaction, such as DNS server, e-mail server, or Web server. As Sun Tzu said in *The Art of War*, "So in war, the way is to avoid what is strong and to strike at what is weak." If a software product shores up its defenses in one area, it must shore them up everywhere else in the product.

Windows offers many network defenses, most notably native IPSec and IPv6 support, and a bi-directional firewall.

IPSec and IPv6 The reason why distributed denial-of-service (DDoS) attacks occur is because IPv4 is an unauthenticated protocol. UDP is one of the worst offenders because it's a connectionless protocol, and it is trivial to spoof UDP packets. But even with TCP, the initial SYN packet is unauthenticated, and a set of attack servers could easily incapacitate a vulnerable server on the Internet by sending millions of bogus TCP SYN packets. There are many other kinds of TCP/IP-related issues, and the IETF is currently discussing the issues in depth. Two draft IETF documents of interest are "Defending TCP Against Spoofing Attacks" and "TCP SYN Flooding Attacks and Common Mitigations." Remember, these are IETF drafts and may never lead to anything substantial, but they are a worthwhile read.

The problem with any potential solution that uses IPv4 is that IPv4 is fundamentally flawed. Enter IPSec and IPv6. IPSec and IPv6 both support authenticated network packets. A full explanation of these protocols is beyond the scope of this chapter, but in Windows Vista, IPv6 is enabled by default. IPv4 is enabled by default, too, but over time Microsoft anticipates that more of the world's networks will migrate to the much more secure protocol. A good example of this is the XBOX Live online network. The core XBOX operating system is a stripped-down version of Windows, but its core networking protocol is essentially IPSec. The

XBOX Live team did not want to use IPv4 because the team knew their servers would be under constant DDoS attack. Requiring IPSec substantially raises the bar on the attackers.

Firewall All versions of Windows since Windows XP have included a built-in software firewall. The version included with Windows XP was limited in that (1) it was not enabled by default, and (2) its configuration was limited to blocking only inbound connections on specific ports. The firewall in Windows XP SP2 was substantially improved to address one core issue: Users with multiple computers in the home wanted to share files and print documents, but the old firewall would only allow this to happen if the file and print ports (TCP 139 and 445) were open to the Internet. So in Windows XP SP2, there is an option to open a port, but only on the local subnet. The other change in Windows XP SP2, and by far the most important, is that the firewall is enabled by default.

Windows Vista adds two other functions. The first is that the firewall is a fully integrated component of the rewritten TCP/IP networking stack. Second, the firewall supports optionally blocking outbound connections. In the author's opinion, blocking outbound connections is "security theater," it's not real security. Here's why. Let's say a user has a browser installed (it doesn't matter which one) and the user allows the browser to make outbound connections without prompting the user for confirmation. Malware writers will simply leverage the browser to run their malicious code from within the browser, so to the firewall it looks like the browser is making the request—which is true. The firewall in Windows Vista is intended for management and policy enforcement, not for protection against malicious code.

All firewalls that support outbound connection blocking can easily be circumvented—unless the user wishes to be prompted for every single outbound connection, in which case the user will totally frustrated after 10 minutes of typical use on the Internet.

Let's now discuss another set of defensive technologies in Windows, buffer overrun defenses.

Buffer Overrun Defenses

Most operating systems today, indeed most software in use today, is written in the C and C++ programming languages. C was designed as a high-level assembly language, and because of that requirement, C gives the developer direct access to memory through pointers. Pointers simply point to a memory location. For example, in the following code snippet, the pointer *p* points to an array of 32 characters (a character is an 8-bit value) named password.

```
char password[32];
char *p = password;
```

With this incredible functionality comes risk. The worst kind of security-related bug is the buffer overrun. Most computers are compromised through attacks that take advantage of buffer overrun bugs. Most people's first reaction is, "why not just re-write everything in [insert language dejour]?". There are two reasons. The first is the same reason that the world's cars do not run on hydrogen. It's a great idea, and

it's good for the planet, but gasoline has a massive momentum behind it because people know how to get oil from the ground, refine it, ship it, store it, pump it, build engines that use it, repair engines that use it, and so on. There are also problems with hydrogen that still make it impractical today. The same reasoning applies for replacing C and C++ with, say, Java or C#. These languages and run-time environments are not quite up to the task for building operating systems. That may change in the future, but it will be a monumental task to convert C and C++ code to Java or C#.

The other reason is that simply replacing C and C++ with another language does not solve the real problem, which is that software developers have too much trust in the data they receive. A buffer overrun occurs because the developer expects a buffer of 32 bytes, and the attack provides a buffer that is larger. In the author's opinion, the real way to solve the buffer overrun problem is to teach new developers (and old, jaded developers, for that matter) the simple rule of never trusting input and to identify data as data enter the system and to sanitize or reject the data.

Taking the example code above, the following is a classic buffer overrun example:

```
void ParseData(char *pwd) {
    char password[32];
    strcpy(password, pwd);
    // etc.
}
```

The problem with this code is that the strcpy function continues copying pwd into password and stops only when it hits a NULL character ('\0') in the source string, pwd. If the attacker controls pwd, then he or she can determine where the trailing NULL resides, and if the attacker decided to place it after the 32nd character in pwd, strcpy overflows the password buffer. This example is a classic "stack smash," because the buffer overflow corrupts the password buffer, which resides on the function's stack. So let's look at some of the stack defenses enabled by default in Windows Vista.

Stack-Based Buffer Overrun Detection (/GS) Normally in Windows, a function's stack looks like Figure 24.2a. You will notice two interesting items on the stack, EBP (extended base pointer) and EIP (extended instruction pointer). When the function returns, it must continue execution at the next instruction after the instruction that called this function. The CPU does this by taking the values off the stack (called popping) and populating the EBP and EIP registers. Here is where the fun starts. If the attacker can overflow the buffer on the stack, he or she can overrun the data used to populate the EBP and EIP registers with values under his or her control and hence change the application's execution flow. The source code for Windows XP SP2 is compiled with a special compiler option in Microsoft Visual C++ to add defenses to the function's stack. The compiler switch is /GS, and it is usable by anyone with access to a Visual C++ compiler. Once the code is compiled with this option, the stack is laid out as shown in Figure 24.2b.

As you can see, a cookie has been inserted between stack data and the function return address. This random value is checked when the function exits, and if the cookie is corrupted, the application is halted. You will also notice that buffers on the

(a) Without/GS option

(b) With/GS option

Figure 24.2 **Stack Layout in Windows Vista**

stack are placed in higher memory than nonbuffers, such as function pointers, C++ objects, and scalar values. The reason for this is to make it harder for some attacks to succeed. Function pointers and C++ objects with virtual destructors (which are simply function pointers) are also subject to attack because they determine execution flow. If these constructs are placed in memory higher than buffers, then overflowing a buffer could corrupt a function pointer, for example. By switching the order around, the attacker must take advantage of a buffer *underrun*, which is rarer, to successfully corrupt the function pointer. There are variants of the buffer overrun that will still corrupt a function pointer, such as corrupting a stack frame in higher memory, but that's beyond the scope of this chapter.

This compiler option does not affect every function; it affects only functions that have at least 4-bytes of contiguous stack data and only when the function takes a pointer or buffer as an argument. Note that the stack data must be of a "classic stack smash" type, such as a char array.

No eXecute Named NX by Advanced Micro Devices (AMD), Data Execution Prevention (DEP) by Microsoft, and eXecution Disable (XD) by Intel, this technology requires CPU support that helps prevent code from executing in data segments. Most modern Intel CPUs support this capability today, and all current AMD CPUs support NX. DEP support was first introduced in Windows XP SP2 and is a critically important defense in Windows Vista, especially when used with address space layout randomization (ASLR), which we will explain later.

The goal of NX is to prevent data executing. Most buffer overrun exploits enter a computer system as data, and then those data are executed. By default, most system components in Windows Vista and applications can use NX by linking with the /NXCOMPAT linker option.

Stack Randomization This defense is available only in Windows Vista and later. When a thread starts in Windows Vista, the operating system will randomize the stack base address by 0–31 pages. Normally, a page is 4k bytes in size. Once the page is chosen, a random offset is chosen within the page, and the stack starts from that spot. The purpose of randomization is to remove some of the predictability from the attacker. Attackers love predictability because it makes it more likely that an attack will be successful.

There is more to life than stack-based buffer overruns. Data can also reside in another kind of system memory, the heap.

Head-Based Buffer Overrun Detection The seminal buffer overrun paper is "Smashing the Stack for Fun and Profit" by AlephOne [LEVY96]. It's a fantastic read. For quite some time, "smashing the stack" was the attack dejour and little attention was paid to heap-based buffer overruns. Eventually, people realized that even though the heap is laid out differently than the stack, heap-based buffer overruns are exploitable, and can lead to code execution. The nature of such attacks is something you should research [LITC03].

The first heap defense, added to Windows XP SP2, is to add a random value to each heap block and detect that this cookie has not been tampered with. If the cookie has changed, then the heap has been corrupted and the application could be forced to crash. Note that the application crash is not due to instability in the application caused by data corruption; rather the heap manager detects the corruption and fails the application. The process of shutting down an application in this manner is often called "failstop."

The second defense is heap integrity checking; when heap blocks are freed, metadata in the heap data structures are checked for validity, and if the data are compromised, either the heap block is leaked or the application crashes.

Heap Randomization Like stack randomization, heap randomization is designed to take some of the predictability away from the attacker, but it applies to the heap. When a heap is created, the start of the heap is offset by 0–4 MB. Again, this makes things a little harder for the attacker. This feature is new to Windows Vista.

Image Randomization As far as making thinks a little less predictable for the attacker, Windows Vista also adds image randomization. When the operating system boots, it starts up in one of 256 configurations. In other words, the entire operating system is shifted up or down in memory when it is booted. The best way to think of this is to imagine that a random number is selected at boot, and every operating system component is loaded as an offset from that location, but the offset between each component is fixed. Again, this makes the operating system less predictable for attackers and makes it less likely that an exploit will succeed.

Service Restart Policy In Windows a service can be configured to restart if the service fails. This is great for reliability but lousy for security, because if an attacker attacks the service and the attack fails but the service crashes, the server might restart and the attacker will have another chance to attack the system. In Windows Vista, Microsoft set some of the critical services to restart only twice, after which the service will not restart unless the administrator manually restarts

it. This gives the attacker only two attempts to get the attack to work, and in the face of stack, heap, and image randomization, the server is much more difficult.

Note that a full description of all the defenses described in this section, and how to use them in your own code, can be found in [HOWA07].

24.4 BROWSER DEFENSES

There is no point of attack quite like a Web browser. A Web browser interprets a reasonable complex language, HTML, and renders the results. But a Web page can also contain code in the form of scripting languages such as JavaScript, or richer, more capable code such as ActiveX controls, Flash, Java applets, or .NET applications; and mixing code and data is bad for security. Also, all this code and data makes for a rich and productive end-user environment, but it's hard to secure. Web browsers can also render various multimedia objects such as sound, JPEG, BMP, GIF, animated GIFs, and PNG files. Many file formats are rendered by helper objects, called MIME handlers. Examples include video formats such as Quicktime, Windows Media Player, or Real Player. A malicious Web page could take advantage of many possible attack vectors; some vectors are under the direct control of the browser and some are not.

With this setting in mind, Microsoft decided to add many defenses to Internet Explorer 7, especially the browser version in Windows Vista. Perhaps the most important single defense is ActiveX opt-in. An ActiveX control is a binary object that can potentially be invoked by the Web browser using the <OBJECT> HTML tag or by calling the object directly from script. Many common Web browser extensions are implemented as ActiveX controls; probably the most well known is Adobe Flash. It is possible for ActiveX controls to be malicious, and chances are very good that a user already has one or more ActiveX controls installed on his or her computer. But does the user know which controls are installed? We would wager that for most users, the answer is a resounding "no!" Internet Explorer 7 adds a new feature called "ActiveX opt-in," which essentially unloads ActiveX controls by default, and when a control is used for the first time, the user is prompted to allow the control to run. At this point, the user knows that the control is on the computer.

Another important defense on Windows Vista is protected mode. When this default configuration is used, Internet Explorer runs at low integrity level, making it more difficult for malware to manipulate the operating system. See Section 24.1 for a discussion of integrity levels in Windows Vista.

24.5 CRYPTOGRAPHIC SERVICES

Windows includes a complete set of cryptographic functionality, from low-level cryptographic primitives for encryption, hashing, and signing to full-fledged cryptographic defenses, such as the Encrypting File System (EFS), Data Protection API, and BitLocker. Let's look at each of these features in more detail.

Encrypting File System

EFS allows files and directories to be encrypted and decrypted transparently for authorized users. All versions of Windows since Windows 2000 support EFS. On the surface, EFS is very simple; a user or administrator marks a directory to use EFS, and from that point on, any file created in that directory is encrypted. It is possible to encrypt single files, but this is problematic because it is common for applications to create temporary files while manipulating the file in question. But if the target file is marked for encryption, the temporary files are not encrypted, and if the temporary files contain sensitive data, the data are not protected. The way to fix this is to encrypt the entire directory.

At a very high level, EFS works by generating a random encryption key and storing that key, encrypted using the user's encryption key. This key is protected using the Data Protection API (DPAPI) in Windows, and the key used by DPAPI is derived from the user's password. The process of allowing a new user to access an EFS-encrypted file is fairly simple. The file encryption key is encrypted with the user's key, and it is stored alongside the other user keys in the file metadata.

EFS also supports the concept of a file recovery agent, a special capability to decrypt files if, for some reason the user's lose their EFS keys.

The cornerstone of EFS is DPAPI, and that's the next topic.

Data Protection API

The data protection API (DPAPI) allows users to encrypt and decrypt data transparently; in other words, the tasks of maintaining and protecting encryption keys is removed from the user and administered by the operating system. When DPAPI is used to encrypt user data, the encryption keys are derived in part from the user's password. A full explanation of how DPAPI works is available at [NAI01]. Again, the beauty of DPAPI lies in removing the key-management problem from the user and developers. Developers need only call one of two functions, *CryptProtectData* to encrypt and *CryptUnprotectData* to decrypt. These functions also add a message authentication code to the encrypted data to help detect tampering.

BitLocker

Windows Vista adds a much-needed defense to the operating system, BitLocker Drive Encryption. The core threat this technology helps mitigate is data disclosure on stolen laptops. BitLocker encrypts the entire volume with using AES, and the encryption key is stored either on a USB drive or within a Trusted Platform Module (TPM) chip on the computer motherboard. When booting a system that requires the USB device, the device must be present so the keys can be read by the computer, after which BitLocker decrypts the hard drive on the fly, with no perceptible performance degradation. The downside to using a USB device is that if the device is lost, the user loses the encryption keys and cannot decrypt. Thankfully, BitLocker can integrate with Active Directory to store the encryption keys, and BitLocker also supports key recovery.

Perhaps the most important aspect of BitLocker is that, like most security settings in Windows, BitLocker policy can be set as a policy for a single computer and that policy "pushed" to computers that use Active Directory.

BitLocker is the first technology in Windows to use a TPM chip, and that's the next topic.

TPM

The Trusted Platform Module (TPM) is the product of a specification from the Trusted Computing Group, designed to enhance system security by moving many sensitive cryptographic operations into hardware. Many software-based attacks do not affect a hardware solution, such as TPM. TPMs are discussed in Chapter 10.

Windows Vista supports TPM version 1.2.

The best-known feature that uses the TPM, if one is available, is BitLocker Drive Encryption. When a TPM is present and the system is configured appropriately, Windows Vista will use the TPM to validate that the operating system has not been tampered with. This is known as trusted boot, or secure startup, and as the OS boots, critical portions are hashed and the hashes verified.

Microsoft expects more software vendors to make use of the TPM over time, especially as most laptops shipping today include a TPM on the motherboard, and more desktop and server computers ship with embedded TPMs.

24.6 COMMON CRITERIA

Versions of Windows since Windows 2000 have earned Common Criteria EAL4 + Flaw Remediation (ALC_FLR.3) or are in the process of being accredited. What's critically important about the work Microsoft has undertaken in getting its operating systems accredited is that the software stack (the security target) that is evaluated is useable. It's not a whittled-down configuration that is just an FTP server, for example. You can look at the Windows Server 2003 and Windows XP product validation reports at [NIAP05].

24.7 RECOMMENDED READING AND WEB SITE

[HOWA07] covers many of the defenses we built into Windows Vista and explains how you can take advantage of them in your own software. [SYMA07] provides an overview of Vista's security features from the point of view of a software security vendor.

HOWA07 Howard, M., and LeBlanc, D. *Writing Secure Code for Windows Vista.* Redmond, WA: Microsoft Press, 2006.

SYMA07 Symantec. "Security Implications of Microsoft Windows Vista." *Symantec Research Paper,* 2007. symantec.com

Recommended Web site:

- **Microsoft Security Central:** Good collection of information about Windows and Windows Vista security

24.8 KEY TERMS, REVIEW QUESTIONS, AND PROJECTS

Key Terms

Active Directory	domain account	Security Account Manager
BitLocker Drive Encryption	local account	Security Reference Monitor
authentication packages	Local Security Authority	WinLogon
	NetLogon	

Review Questions

24.1 What are the two kinds of ACLs in Windows, and what does each do?

24.2 On Windows, which privilege overrides all ACL checks, and why?

24.3 Which portions of the Windows Vista file system are marked as low integrity? Why are they marked this way?

24.4 Enumerate all aspects of a user's token in Windows Vista.

24.5 Describe the Windows SID format.

24.6 Why are SIDs never removed from a token but instead are marked "Deny"?

24.7 Why are some services in Windows Vista set to only restart twice if they fail?

24.8 Why does XBox Live use IPSec and not IPv4?

24.9 Itemize all the memory-related defenses in Windows Vista. Why are they enabled by default?

Problems

24.1 Paige's (simplified) token looks like this:

```
User:
      FOOCorp\PaigeH
Groups:
      Everyone
      Authenticated Users
      Developers
```

Her word processor attempts to open a file for RWX access, and the file has the following ACL:

```
      Administrators: Full Control
      Authenticated Users: RW
      Developers: RWD
```

Will Paige be granted access to the object? Why or why not?

24.2 Build a threat model for a TPM-based system.

24.3 Find the source code that generates the random number used in –GS. (*Hint:* It's included with Visual Studio and the Windows Development Kit.) Why do you think the random number generator used to generate the –GS cookie is so simple, and why is it not guaranteed to be cryptographically random?

24.4 Look at the assembly language of a non-GS and GS-protected function, how does –GS work?

24.5 Build some vulnerable C/C++ code that reads from a socket. Make the application crash. Compile the application with –GS and retest. What do you observe?

24.6 How are most operating system heap managers implemented?

24.7 How do heap-based buffer overruns work because of this implementation?

24.8 When running in a configuration that supports Common Criteria, Windows must crash when the security event log is full. Why?

24.9 It is recommended that when using BitLocker on a laptop, the laptop should not use standby mode, rather it should use hibernate mode. Why?

24.10 How does Internet Explorer 7.0 in Windows Vista let the user save a file to his or her desktop if the user's desktop is medium integrity but the IE process is low integrity?

24.11 Why do you think UDP is considered "higher attack surface" than TCP?

24.12 Why is it important that an application verify that impersonation functions succeed? What should an application do if impersonation fails?

24.13 Review the Security Development Lifecycle process. What would you change and add, and why?

24.14 Describe the security issues with IPv4. How does IPv6 remedy these issues?

24.15 Do you agree that blocking outbound connections at a PC's firewall is "security theater"? Why or why not?

24.16 Discuss Common Criteria. Does it work? Does it meet its goals? What are its goals? How could it be improved?

24.17 Compare and contrast the Windows ACL model with the Linux permissions model.

Projects

A series of projects are contained in a document, filename WindowsProjects.pdf, available at this book's Web site. These projects were developed by Ricky Magalhaes of Fastennet Security. These are designed to help you learn about Windows security. These are not review questions but rather exercises that expose parts of Windows to you in security context and will help you to learn parts of windows security.