Microsoft

secure software DEVELOPMENT SERIES

BEST PRACTICES

WRITING SECURE CODE

2

Second Edition

Practical strategies and techniques for secure application coding in a networked world

"Required reading at Microsoft."
– Bill Gates

Michael Howard and David LeBlanc

Microsoft

WRITING SECURE CODE

2

Second Edition

Michael Howard
and David LeBlanc

Practical strategies and proven techniques for building secure applications in a networked world

"Required reading at Microsoft."
- Bill Gates

Table 7-1 Some Potent Windows Privileges *(continued)*

Display Name	Internal Name (Decimal)	*#define* (Winnt.h)
Load And Unload Device Drivers	*SeLoadDriverPrivilege (9)*	*SE_LOAD_DRIVER_NAME*
Take Ownership Of Files Or Other Objects	*SeTakeOwnershipPrivilege (8)*	*SE_TAKE_OWNERSHIP_NAME*

Let's look at the security ramifications of these privileges.

SeBackupPrivilege Issues

An account having the Backup Files And Directories privilege can read files the account would normally not have access to. For example, if a user named Blake wants to back up a file and the ACL on the file would normally deny Blake access, the fact that he has this privilege will allow him to read the file. A backup program reads files by setting the *FILE_FLAG_BACKUP_SEMANTICS* flag when calling *CreateFile*. Try for yourself by performing these steps:

1. Log on as an account that has the backup privilege—for example, a local administrator or a backup operator.
2. Create a small text file, named Test.txt, that contains some junk text.
3. Using the ACL editor tool, add a deny ACE to the file to deny yourself access. For example, if your account name is Blake, add a Blake (Deny All) ACE.
4. Compile and run the code that follows this list. Refer to MSDN at *http://msdn.microsoft.com* or the Platform SDK for details about the security-related functions.

```
/*
  WOWAccess.cpp
*/
#include <stdio.h>
#include <windows.h>

int EnablePriv (char *szPriv) {
    HANDLE hToken = 0;

    if (!OpenProcessToken(GetCurrentProcess(),
                          TOKEN_ADJUST_PRIVILEGES,
                          &hToken)) {
```

```
        printf("OpenProcessToken() failed -> %d", GetLastError());
        return -1;
    }

    TOKEN_PRIVILEGES newPrivs;
    if (!LookupPrivilegeValue (NULL, szPriv,
                               &newPrivs.Privileges[0].Luid)) {
        printf("LookupPrivilegeValue() failed->%d",
           GetLastError());
        CloseHandle (hToken);
        return -1;
    }

    newPrivs.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
    newPrivs.PrivilegeCount = 1;

    if (!AdjustTokenPrivileges(hToken, FALSE, &newPrivs , 0,
            NULL, NULL)) {
        printf("AdjustTokenPrivileges() failed->%d",
           GetLastError());
        CloseHandle (hToken);
        return -1;
    }

    if (GetLastError() == ERROR_NOT_ALL_ASSIGNED)
        printf
("AdjustTokenPrivileges() succeeded, but not all privs set\n");

    CloseHandle (hToken);
    return 0;
}

void DoIt(char *szFileName, DWORD dwFlags) {

    printf("\n\nAttempting to read %s, with 0x%x flags\ n",
           szFileName, dwFlags);

    HANDLE hFile = CreateFile(szFileName,
                              GENERIC_READ, FILE_SHARE_READ,
                              NULL, OPEN_EXISTING,
                              dwFlags,
                              NULL);

    if (hFile == INVALID_HANDLE_VALUE) {
        printf("CreateFile() failed->%d",
```

(continued)

```
                GetLastError());
            return;
        }

        char buff[128];
        DWORD cbRead=0, cbBuff = sizeof buff;
        ZeroMemory(buff, sizeof buff);

        if (ReadFile(hFile, buff, cbBuff, &cbRead, NULL)) {
            printf("Success, read %d bytes\n\nText is: %s",
                   cbRead, buff);
        } else {
            printf("ReadFile() failed - > %d", GetLastError());
        }
        CloseHandle(hFile);
    }

    void main(int argc, char* argv[]) {
        if (argc < 2) {
           printf("Usage: %s <filename>", argv[0]);
           return;
        }

        //Need to enable backup priv first.
        if (EnablePriv(SE_BACKUP_NAME) == -1)
            return;

        //Try with no backup flag -  should get access denied.
        DoIt(argv[1], FILE_ATTRIBUTE_NORMAL);

        //Try with backup flag - should work!
        DoIt(argv[1], FILE_ATTRIBUTE_NORMAL |
                      FILE_FLAG_BACKUP_SEMANTICS);
    }
```

This sample code is also available with the book's sample files in the folder Secureco2\Chapter07. You should see output that looks like this:

```
Attempting to read Test.txt, with 0x80 flags
CreateFile() failed -> 5

Attempting to read Test.txt, with 0x2000080 flags
Success, read 15 bytes
Text is: Hello, Blake!
```

As you can see, the first call to *CreateFile* failed with an access denied error (error #5), and the second call succeeded because backup privilege was enabled and the backup flag was used.

In exploiting *SeBackupPrivilege*, I showed some custom code. However, if a user has both *SeBackupPrivilege* and *SeRestorePrivilege*, no custom code is needed. A user with these privileges can read any file on the system by launching NTBackup.exe, back up any file regardless of the file ACL, and then restore the file to an arbitrary location.

Assigning this user right can be a security risk. Since there is no way to be sure whether a user is backing up data legitimately or stealing data, assign this user right to trusted users only.

SeRestorePrivilege Issues

Obviously, this privilege is the inverse of the backup privilege. With this privilege, an attacker could overwrite files, including DLLs and EXEs, he would normally not have access to! The attacker could also change object ownership with this privilege, and the owner has full control of the object.

SeDebugPrivilege Issues

An account having the Debug Programs privilege can attach to any process and view and adjust its memory. Hence, if an application has some secret to protect, any user having this privilege and enough know-how can access the secret data by attaching a debugger to the process. You can find a good example of the risk this privilege poses in Chapter 9, "Protecting Secret Data." A tool from nCipher (*http://www.ncipher.com*) can read the private key used for SSL/TLS communications by groveling through a process's memory, but only if the attacker has this privilege.

The Debug Programs privilege also allows the caller to terminate any process on the computer through use of the *TerminateProcess* function call. In essence, a nonadministrator with this privilege can shut down a computer by terminating a critical system process, such as the Local Security Authority (LSA), Lsass.exe.

But wait, there's more!

The most insidious possibility: an attacker with debug privileges can execute code in any running process by using the *CreateRemoteThread* function. This is how the LSADUMP2 tool, available at *http://razor.bindview.com/tools*, works. LSADUMP2 allows the user having this privilege to view secret data stored in the LSA by injecting a new thread into Lsass.exe to run code that reads private data after it has been decrypted by the LSA. Refer to Chapter 9 for more information about LSA secrets.

The best source of information about thread injection is *Programming Applications for Microsoft Windows*, by Jeffrey Richter (Microsoft Press).

Note Contrary to popular belief, an account needs the Debug Programs privilege to attach to processes and debug them if the process is owned by another account. You do not require the privilege to debug processes owned by you. For example, Blake does not require the debug privilege to debug any application he owns, but he does need it to debug processes that belong to Cheryl.

SeTcbPrivilege Issues

An account having the Act As Part Of The Operating System privilege essentially behaves as a highly trusted system component. The privilege is also referred to as the Trusted Computing Base (TCB) privilege. TCB is the most trusted and hence most dangerous privilege in Windows. Because of this, the only account that has this privilege by default is SYSTEM.

Important You should not grant an account the TCB privilege unless you have a really good reason. Hopefully, after you've read this chapter, you'll realize that you do not need the privilege often.

Note The most common reason developers claim they require the TCB privilege is so that they can call functions that require this privilege, such as *LogonUser*. Starting with Windows XP, *LogonUser* no longer requires this privilege if your application is calling to log on a Windows user account. This privilege is required, however, if you plan to use *LogonUser* to log on Passport account or if the *GroupsSid* parameter is not *NULL*.

SeAssignPrimaryTokenPrivilege and *SeIncreaseQuotaPrivilege* Issues

An account having the Replace A Process Level Token and Increase Quotas privileges can access a process token and then create a new process on behalf of the user of the other process. This can potentially lead to spoofing or privilege elevation attacks.

SeLoadDriverPrivilege Issues

Executable code that runs in the kernel is highly trusted and can perform just about any task possible. To load code into the kernel requires the *SeLoadDriverPrivilege* privilege because the code can perform so many potentially dangerous tasks. Therefore, assigning this privilege to untrusted users is not a great idea, and that's why only administrators have this privilege by default.

Note that this privilege is not required to load Plug and Play drivers because the code is loaded by the Plug and Play service that runs as SYSTEM.

SeRemoteShutdownPrivilege Issues

I think it's obvious what this privilege allows—the ability to shut down a remote computer. Note that, like all privileges, the user account in question must have this privilege enabled on the target computer. Imagine the fun an attacker could have if you gave the Everyone group this privilege on all computers in your network! Talk about distributed denial of service!

SeTakeOwnershipPrivilege Issues

The concept of object *owners* exists in Windows NT and later, and the owner always has full control of any object the account owns. An account that has this privilege can potentially take object ownership away from the original owner. The upshot of this is that an account with this privilege can potentially have total control of any object in the system.

More Info Note that in versions of Windows earlier than Windows XP, an object created by a local administrator is owned by the local administrators group. In Windows XP and later versions, including Windows .NET Server 2003, this is configurable; the owner can be either the local Administrators group or the user account that created the object.