Rootkits

Stéphanie Deléamont and Benoît Perroud
EPFL/LASEC: Security Protocols and Applications

Abstract: Rootkits are becoming a bigger threat every day even though people don't know them. The goal of this paper is to give an overview of rootkits from both a theoretical and practical point of view in order to sensitize people to this problem.

1. Introduction

In the first part of this paper, we will give the definition of a rootkit and list its main features. We will categorize rootkits depending on how deep they hide themselves into the system. In the second part, we will discuss the current threats caused by rootkits and focus on the Sony's example. We will then present methods to respectively avoid, detect and recover from rootkits. Finally we will give a proof of concept of a kernel rootkit to help the reader to have a more practical approach.

2. Definition

The word rootkit is composed of two parts:

- *root*, which is the highest privileged user in a Unix system (=administrator)
- *kit*, which refers to a set of tools

A rootkit is thus a **set of tools** (programs) that helps an attacker **to keep an unauthorized root access** after having compromised a system. It moreover **hides the presence** and traces of the attacker.

This term rootkit is very old and is dated back to the days when Unix ruled the world. It is believed that rootkits originated around 1994; of course the first operating system that rootkits focused on primarily was Unix. The way the rootkit became unique in that day and age was to simply replace the login binary (they didn't have SSH at that time) and so rootkits began. Rootkits for the Unix operating system were typically used to elevate the privileges of a user to the root level. This explains the name of this category of tools.

The basic idea of rootkits is strongly inspired from stealth viruses on MS-DOS: they were the first software to try to hide themselves by modifying the OS.

It is however important to understand the distinction between rootkits and viruses/worms. The key distinction between them relates to propagation. In general, a rootkit limits itself to maintaining control of one system as viruses and worms also attempt to spread to other systems. However viruses, worms and rootkits use similar techniques to infect systems (e.g. modify core software components of the system) and hide from the administrator. Of course there are hybrids. A worm can install a rootkit, and a rootkit might include copies of one or more worms, packet sniffers or port scanners.

However rootkits are typically not malicious by themselves. They don't cause deliberate damages but are used to hide malicious code: a malware (virus, worms, backdoors, spywares ...) could remain active and undetected in a system for a long time if it uses a rootkit, even if the computer is protected with state-of-the-art antivirus.

Throughout the rest of this paper, we generally make the assumption that the attacker has already broken into the system (using any vulnerabilities) and has gained sufficient privileges to install a rootkit (using any local exploits) when talking about rootkits.

The basic features of a rootkit are:

- to cloak itself to not be detected by the system administrator
  - by hooking (patching) system call table (explained below)
  - by modifying installed softwares
  - by cleaning logs
- to be persistent on the system, for example by launching itself automatically after each reboot
- to check if all the components of the rootkit are still installed and if they are not, reinstall them

Moreover, depending of its complexity, it can hide

- files and directories according to
  - a prefix
  - an owner
  - a group owner
- processes
  - the children of the running rootkit
  - processes which have a custom groupid
- network connections
  - at a certain port
  - from/to a given source

The main purpose of installing a rootkit is to gain a ***long term remote control*** of a computer using a simple TCP listener, a connection to an IRC channel or even a complex ICMP backdoor, and then to use it to:

- sniff and spy the network (for logins, ...)

- launch distributed deny of service (DDos) attack
- spam
- ...

A machine infected in such a way is called a zombie.

In the following part, we will describe the three main types of rootkits in increasing order of deepness of insertion in the system.

2.1 Binary rootkits

Binary rootkits are the *first rootkits' generation*. They appear under the form of patched binaries (like *ls* or *ps* in the Unix world, *cd* or *dir* in the Windows world), which replace the original ones. We speak from trojaned binaries because these binaries act as the normal ones, but they also do others things in background, such as hide some information, open backdoors, connect to IRC channels, send mails...

The binaries that are commonly modified by rootkits are the ones mostly used by system administrators, like changing directory, listing directories, listing processes and so on. This is what helps the rootkit hide in first place.

2.2 Kernel rootkits

Kernel rootkits are the *second generation of rootkits*. They are inserted very deep into the kernel or the libraries of the operating system. They are harder to detect because they hook system calls, like *sys_read(...)*, *sys_open(...)* or even *sys_kill(...)* to modify their behavior. System calls are the interfaces provided by the kernel to allow the high level software to access raw data.

The system call table (sys_call_table) is a vector table in memory which stores the address pointers to all the system functions of the kernel.
In Linux operating system, the file *asm/unistd.h* defines all the system calls and gives the index of the table which points to the system function [14].

asm/unistd.h:

```
#define __NR_restart_syscall     0
#define __NR_exit                1
#define __NR_fork                2
#define __NR_read                3
#define __NR_write               4
#define __NR_open                5
...
#define __NR_inotify_rm_watch  293
```

In memory, the table looks like:

```
sys_call_table[__NR_restart_syscall] = 0xc0128f20
sys_call_table[__NR_exit]            = 0xc011ee60
sys_call_table[__NR_fork]            = 0xc0101c00
sys_call_table[__NR_read]            = 0xf9173000
sys_call_table[__NR_write]           = 0xc0161880
...
```

So an attacker can easily modify any entry of this table to make it point to a trojaned function of a kernel module. This operation is called **hooking**. Notice that, in our example, *sys_read* has a strange address compared to the others, so it is very probable that it has already been hooked (but maybe by a normal driver).

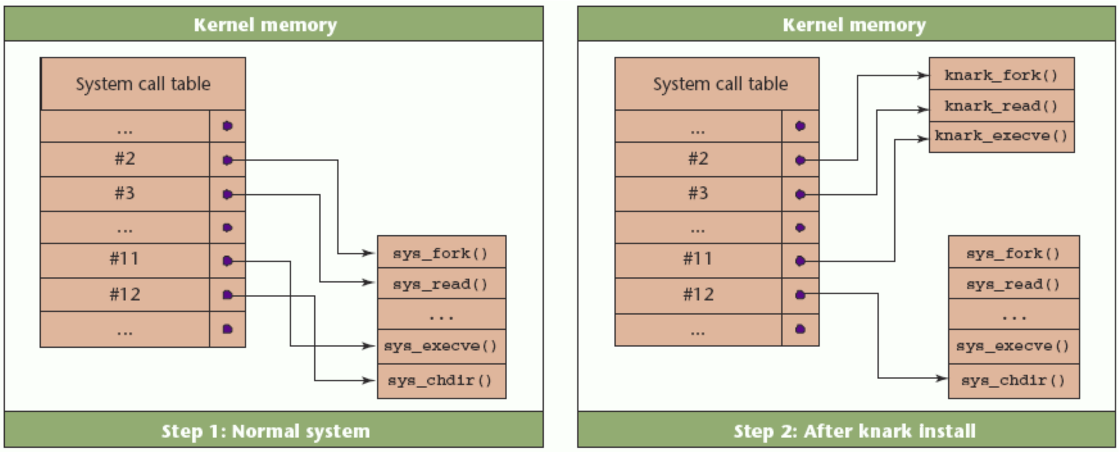

Figure 1: Example of system call table hooking by the knark rootkit
Source: Security and Privacy Magazine, Volume4 Issue 1 (January-February 2006)

We also find the same concept of system call table in Windows, so the idea of kernel rootkits is the same in both worlds. The difference is essentially in naming; in Windows system calls reside in libraries (DLL) and are called the APIs.

Example of kernel rootkit for the linux kernel 2.6:

- *eNYeLKM* [28]: it inserts salts inside *system_call* and *sysenter_entry* handlers. It does not modify directly *sys_call_table*, but the effect remains exactly the same, but harder to detect. It then hides files, directories, and processes, as well as chunks inside of files, remote reverse_shell access, etc.

Example of kernel rootkit for Windows:

- *HackDefender* [29]: registers itself as service to be launched at each startup, hooks various API in allocating memory inside a process and injecting its own code, and then place a jump into the hooked API to its injected code. It then hides files and process specified in a file called *Hxdefxxx.ini*, where xxx is a number between 026 and 071.

2.3 Virtual-machine based rootkits

This very new kind of rootkits, shortly called VMBR, are hopefully only at proof-of-concept state today. They are deduced by the fact that lower layers in a system have fundamental advantages on uppers, because upper layers depend on abstraction implemented by the lowers. Thus deeper the rootkit is hidden, the more difficult it is to find. So what happens when the rootkit is underneath the operating system? This is currently intensively studied by a group sponsored by Microsoft at the University of Michigan [11].

To install such a rootkit, one must modify the boot record of the system's primary hard disk (called MBR for Master Boot Record, the 512 first bytes of the hard disk specifying the OS to be started). This ensures that the VMBR loads before the target operating system. This step is easy to detect with anti-malware systems. A simple technique to avoid this protection is to modify the MBR at the last step of shutdown of the target system, after that most of the running services have already exited (especially the anti-malware systems).
Once the VMBR is started, it starts the target operating system, which depends on the virtualization provided by the malware.

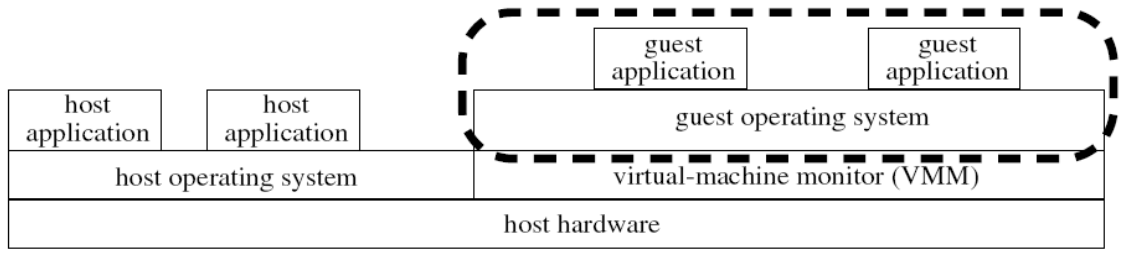

Figure 2: Architecture of a today's standard virtual machine
Source: http://www.eecs.umich.edu/virtual/papers/king06.pdf

There already exists some mechanisms to avoid VMBR:

- Secure hardware (Intel's LaGrande [12] or AMD's platform for trustworthy computing [13])
- Static MBR (read-only)
- Lower virtual machine which acts as a secure bootable media and monitors which OS is booting on the computer by gaining control of the boot sequence before any other device.

Several advanced techniques have also been developed to detect attacks of VMBR type. They include response time analysis, CPU/memory/disk/network bandwidth perturbation analysis and non virtualization of custom I/O.

3. Actuality

Nowadays, even though rootkits are unknown to most end users, they are already quite common in spyware programs but not as common in mass-mailing viruses. There is clear evidence that rootkits is a technique that works in practice. But the actual threat is still small compared to the potential of this technique.

Microsoft has publicly warned against the emergence of rootkits during the RSA conference in February 2005 "Be afraid, be very afraid"[6]. At the same occasion they have announced to be working on a large project called Ghostbuster [22] to protect computers against rootkits.

3.1 The Sony's story

In 2005, a large scandal has come up about Sony [18] shipping CDs with special DRM protection behaving as rootkits. It appeared that millions of individual discs had been sold containing either one of the following DRM technologies: XCP, created by the British company First4Internet and shipped on around 20 titles, and MediaMax created by the US company SunnComm and shipped on around 50 titles. The XCP version has become more famous because it is the most virulent version. It has been discovered and revealed by Mark Russinovich [19], a programmer at Sysinternals by using the program RootkitRevealer [20] (see below).

The basic idea behind the DRM protection is the following: The CD comes with a music player limiting the number of copies to 3. The music files of the CD cannot be read with another player than the one provided on the CD. Used for DRM purposes, the rootkit installed at the same time as the player scans the machine to check what the user is doing with the CD and sends the information to a web site. The player and thus the CD cannot be run if the rootkit is removed from the computer.

Both XCP and MediaMax target only Windows operating system by taking advantage of the autorun feature. The installation of the rootkit is automatically launched when the CD is inserted in the computer. The XCP rootkit patches several kernel functions via the system call table and hides all files, directories, registery key or process whose name begins with sys. It thus creates a big security breach since any other malware can hide by using such a name. Multiple malwares (e.g. Troyans, viruses,…) have already used this security breach to hide.

Both XCP and MediaMax are installed without the user's informed consent, are difficult or impossible to uninstall, and transmit information about the user's activities without notice or consent. The information sent back to the vendor consists of the user's IP address, the disc inserted and the times and dates it was inserted.

Both XCP and MediaMax were designed to resist detection and removal. No uninstallers were initially provided. Sony has later on proposed an uninstaller after the scandal. The uninstaller reveals to not delete entirely the rootkit and to be even more dangerous than the rootkit itself. Sony has modified its uninstaller but it still seems not to delete the rootkit entirely.

4. Avoidance

Avoiding rootkits remains the same task as preventing an attacker from gaining administrator privileges on a computer. Thus all security principles (least privileges, defense in depth, choke point, weakest link, default deny, user participation and simplicity) must be applied to keep the attacker outside of the system. For example the system administrator must ensure no trivial password, usage of encrypted protocols, keeping softwares up to date and so on.

Moreover the Linux kernel permits to deactivate the loadable module support. It prevents attackers from loading evil modules in the kernel.

The other main technique consists of having a read-only system written in a ROM (like the Xbox). It thus becomes impossible to modify the binaries of the system and the other modifications of the system are not persistent. The drawback of this method is that the system cannot be updated easily. In addition, it is easy to damage definitely the ROM when writing something in it (which may require a physical manipulation on the hardware).

5. Detection

The main problem to detect a rootkit on a rooted system is that we cannot trust the answers of the system anymore, because either its binaries or its kernel have been modified.

The detection of hidden activities from the compromised system could even completely fail. The only solution is to refer to **external checks**.

- Use a port scanner (such as Nmap [27]) to find open port on the system which cannot be listed by *netstat*. This would reveal the presence of a TCP backdoor. If the rootkit installs a more complicated UDP backdoor or ICMP tunnels, the scanner is inefficient.
- Boot the computer on a sane support, like a LiveCD (e.g. Knoppix), and scan the rooted disk from the sane system. Current anti-virus have the definitions of the most common rootkits in their signatures databases. As the rootkits aren't activated at startup with the external system, they will not hide their footprints.
- Use cross-view detection technique which typically compares a low level view of a system with a high level view and notices any differences between the two
- Memory forensics mainly consist of detecting hooked system calls

- Check integrity files by filling in a (protected) database with the hash of the main binaries and configuration files.

The most common detection tools to find rootkits are listed below:

- Unix
    - Rkhunter [23] (scans the system for a list of known rootkits, uses cross-view for hidden process detection)
    - Chkrootkit [24] (checks the system by finding signatures (actually string) of rootkits into some predefined binaries of the system)
    - Tripwire [25] and its replacement AIDE [26] (checks files integrity)

- Windows
    - Rootkit revealer [20]: (uses cross-view detection to find hidden files or hidden Registry entries)
    - F-secure Blacklight [2]: (uses cross-view detection to find hidden processes).
    - MS Strider Ghostbuster [22] (under development, should use most of the actual techniques listed above)

6. Recovery

Once a system has been compromised, the time spent to fully reinstall the system will be small compared to the time to delete all the entries of a rootkit. Moreover, if one single rooted binary is forgotten, the day it is used the system will be rooted again. Remember that the main activity of a rootkit is to keep itself on the system, so each time a rooted binary is running, it may test if the system is always compromised, and if it is not the case, recompromise it.

7. Demo

In this part we will show to the reader a basic example of Linux kernel module which gives root access to a particular user. This example is intended for educational purposes only, and should not be used for malicious purposes.

The code given below is not exhaustive but shows well how to hook the system call table.

```
/* The user ID which will have root access */
#define MAGIC_UID  1000

/* System call table */
extern void **sys_call_table;

/* Original system functions */
int (*org_getuid) (void);
int (*org_getuid32) (void);
int (*org_geteuid) (void);
int (*org_geteuid32) (void);

/* Prototype of the modified system functions */
int my_getuid (void);
int my_geteuid (void);

/* This function return the user id of the current user, and 0 (root) for
the specified one */
int my_getuid () {
      int ret = org_getuid();
      if (ret == MAGIC_UID) {
            current->uid = 0;
            return 0;
      }
      return ret;
}

/* Same as previous one but with effective uid */
int my_geteuid () {
      int ret = org_geteuid();
      if (ret == MAGIC_UID) {
            current->euid = 0;
            return 0;
      }
      return ret;
}

/* Function called when loading the module into the kernel. This function
hook the system call table */
int init_module(void) {
      org_getuid = sys_call_table[__NR_getuid];
      org_getuid32 = sys_call_table[__NR_getuid32];
      (void *)sys_call_table[__NR_getuid] = (void *) my_getuid;
      (void *)sys_call_table[__NR_getuid32] = (void *) my_getuid;

      org_geteuid = sys_call_table[__NR_geteuid];
      org_geteuid32 = sys_call_table[__NR_geteuid32];
      (void *)sys_call_table[__NR_geteuid] = (void *) my_geteuid;
      (void *)sys_call_table[__NR_geteuid32] = (void *) my_geteuid;

      return 0;
}

/* Called when unloading the kernel module, restore to the initial state */
void cleanup_module(void) {
      sys_call_table[__NR_getuid] = org_getuid;
      sys_call_table[__NR_getuid32] = org_getuid32;
      sys_call_table[__NR_geteuid] = org_geteuid;
      sys_call_table[__NR_geteuid32] = org_geteuid32;
}
```

This example shows how to elevate privileges, but to build a “complete” rootkit, code should be added to hide the presence of this module to the administrator.

8. Conclusion

We tried to give an overview of how rootkits work both from a theoretical and a practical point of view. We hope that this paper will contribute to make people aware of rootkits because they are probably the next big threat against computer systems.

To conclude we would like to stress the fact that neither a perfect rootkit nor a perfect protection against rootkits will ever exist. Primarily because there will always remain exploits on operating systems and secondly because rootkits will always be softwares running on a computer which implies at least memory footprints. Protection against rootkits is thus a competition of competences between hackers and system administrators.

9. References

General references

[1] Rootkit Magazine : http://www.rootkit.com
[2] Packet storm security : http://packetstormsecurity.org/
[3] Wikipedia: http://en.wikipedia.org/wiki/Rootkit
[4] http://www.l0t3k.org/security/docs/rootkit/
[5] IEEE, Security and Privacy: volume 4 issue 1
[6] "Be afraid, very afraid": http://www.computerworld.com/securitytopics/security/holes/story/0,10801,99843,00.html
[7] Cryptography and Security Lecture: http://lasecwww.epfl.ch/cs2006/

Technical references

[8] FUto analysis : http://www.uninformed.org/?v=3&a=7&t=pdf
[9] http://www.porcupine.org/forensics/tct.html
[10] Security focus, Detecting Rootkits And Kernel-level Compromises In Linux : http://www.securityfocus.com/infocus/1811
[11] "SubVirt: Implementing malware with virtual machines", University of Michigan and Microsoft Research, http://www.eecs.umich.edu/virtual/papers/king06.pdf
[12] Intel LaGrande: http://www.intel.com/technology/security/
[13] AMD trustworthy computing: http://conference.digitalidworld.com/2004/attendees/slides/1027_1700_E1.pdf
[14] Linux kernel: http://www.kernel.org

Actualities - Sony

[15] Generation-nt: http://www.generation-nt.com/actualites/
[16] Futura-sciences: http://www.futura-sciences.com/
[17] Clubic : http://www.clubic.com
[18] Sony: http://cp.sonybmg.com/
[19] http://www.sysinternals.com/blog/2005/10/sony-rootkits-and-digital-rights.html

Tools

[20] RootkitRevealer: http://www.sysinternals.com/Utilities/RootkitRevealer.html
[21] F-Secure Blacklight: http://www.f-secure.com/blacklight/
[22] Ghostbuster: http://research.microsoft.com/rootkit/

[23] Rkhunter: http://www.rootkit.nl/
[24] Chkrootkit: http://www.chkrootkit.org/
[25] Tripwire: http://www.tripwire.org/
[26] AIDE: http://aide.sourceforge.net/
[27] Nmap : http://www.insecure.org/nmap/
[28] eNYeLKM : http://www.enye-sec.org/programas.html
[29] HackDefender : http://www.hxdef.org/news.php