Asian Journal of Information Technology 6 (5): 618-621, 2007

Finding Hided Processes in Linux

Yuan Yuan and Dai Guanzhong
College of Automation, Northwestern Polytechnical University,
Xi'an Shaanxi 710072, People Republic China

Abstract: This research analyses the mechanism of using LKMs backdoors to hide processes. According to the flaw in backdoors' design and the characteristics of/proc filesystem, a new method for finding hided processes is presented. That is traversing all possible PID directories to find out each existent process in fact. Through comparing them with the ordinary output, the hided processes would be discovered. At last the code realized in Perl has been presented. The experiment shows that this method can find the processes hided by LKMs backdoors efficiently.

Key words: Linux, LKMs backdoors, system calls, hide processes, PID

INTRODUCTION

Linux OS (Operation System) has arrested more and more attention because of its stability, security and open source. Due to this mechanism called Loadable Kernel Modules (LKMs), it is able to dynamically add code to the kernel at runtime, which allows us to access very sensitive parts of OS. Once LKMs are loaded in kernel, they can change kernel variables, reload kernel functions, expand or cut down some function of OS (Pragmatic, 1999). Moreover, when modules are no longer needed, they can be removed in security. All of these fit the development of device drivers very much. So, new modules that support the latest hardware are being developed continuously and added to the default Linux kernel distribution. However, because of its high privilege it also offers a chance to hackers and leads many malicious code based on LKMs appearing. It brings huge threat to Linux OS.

This study is divided into three parts: At first, the features of Linux/proc filesystem are introduced. Following, after analyzing the principle of LKMs backdoors and the flaw in hiding progresses technique, one method for finding hided progresses by exhaustion is presented. At the end of this study we realize it in Perl and show the experiment results.

PROC FILE SYSTEM AND PROCESSES

In Linux/proc is a virtual filesystem and used heavily. It doesn't reside on any physical or remotely mounted disk. It only provides an interface for the communication between kernel and application programs. This makes application programs get the current runtime state, data information in kernel and modify some system configuration (Zhao, 2004). Many commands in Linux, such as ps, top and uptime, obtain their information from /proc. Some device drivers also export messages via /proc.

Within/proc there are many numerical directories which correspond to the PID (Process Identification) of processes running in the system currently. Each directory contains both subdirectories and regular files. They elaborate further on the runtime attributes of process. For example, within directory/proc/1, the result of command ls-l is shown in Fig. 1.

The three symbol link files are very important to next analysis. File cwd links the current directory of this process; file exe links the complete path of this executable file; file root links its root directory, it is/commonly.

PRINCIPLE OF LKMs BACKDOORS

This part only explains how LKMs backdoors are implemented. To learn how to write your own LKMs, see our reference list (Pragmatic, 1999; Yuan, 2005; Shi, 2003).

LKMs backdoors are called rootkits for short. When these rootkits are loaded or installed on the aim machine, they will run as a part of kernel. Then, hackers will own the root privilege by them so that this system will be controlled completely by intruder (Steath, 2003). Unfortunately, it is too difficult to find the hidden danger for system managers.

The famous rootkits: Knark and Adore rootkits are popular at the moment (Rob, 2003). Knark is based on kernel 2.2 and it isn't a stable version. Its important functions are hiding files, hiding progresses, redirecting ELF files, hiding information of network and so on.

Corresponding Author: Yuan Yuan, College of Automation, Northwestern Polytechnical University, Xi'an Shaanxi 710072, People Republic China

```
-r--r--r--    1 root    root    0 8□ 31    15: 43 cmdl i ne
l rwxrwxrwx   1 root    root    0 8□ 31    15: 43 cwd -> /
-r--------    1 root    root    0 8□ 31    15: 43 envi r on
l rwxrwxrwx   1 root    root    0 8□ 31    15: 43 exe -> / sbi n/ I ni t
dr- x------   2 root    root    0 8□ 31    15: 43 f d
-r--r--r--    1 root    root    0 8□ 31    15: 43 maps
-rw-------    1 root    root    0 8□ 31    15: 43 mem
-r--r--r--    1 root    root    0 8□ 31    15: 43 mount s
l rwxrwxrwx   1 root    root    0 8□ 31    15: 43 root -> /
-r--r--r--    1 root    root    0 8□ 31    15: 43 st at
-r--r--r--    1 root    root    0 8□ 31    15: 43 st at m
-r--r--r--    1 root    root    0 8□ 31    15: 43 st at us
```

Fig. 1: Structure of process init

Adore resembles Knark for functions, but it supplies uninstall (need be validated by password) function and is used on kernel 2.2 and 2.4. It is easily to configure and use Adore.

Moreover, hackers have exploited many rootkits according to their needs. Such as Vlogger is especial for recording the knock of keyboard and Rkit is used just for setting the defined UID to zero to get the manager privilege.

Technique of rootkits: Each OS has some functions build into its kernel, which are used for every operation on that OS. These functions are system calls. They represent a transition from user space to kernel space. Most library functions rely on system calls. System calls are implemented through a given maskable interrupt. In Linux, the interrupt is int 0×80. When the int 0×80 instruction is executed, control is given to the kernel.

Command ls is used to list directory contents. It needs sys_open(), sys_getdents64(), sys_write() system calls etc... Especially, the result of command "ls" is shown on screen by sys_write(). So, if we capture sys_write() in kernel and modify its display, the output of ls would be not the real directory information. The primary characteristic of LKMs backdoors is capturing and modifying many system calls at the same time. As a result, the system response will be changed entirely. Let's see an example of hacking sys_getdents64() system call to hide appointed progress.

Command ps-A is used to report process status and it gives a snapshot of the current processes. Its output before sys_getdents64() being captured is shown as Fig. 2.

There are thousands of processes in system, but we only show part of them simply. Our purpose is hiding a progress named server whose PID is 3257. But none of system calls could be used to get a list of current processes directly in Linux. Command ps-A is realized only by querying numerically directories within /proc. It just does ls on /proc. Every number it finds stands for a runtime process's PID. Now we can hide processes by the technique of hiding file, because /proc is realized as an interface of filesystem by previous introduction.

```
Pl D TTY                Time aVD
  1 ?                   00: 00: 04 I ni t
  2 ?                   00: 00: 00 kevent d
  3 ?                   00: 00: 00 kammd
  4 ?                   00: 00: 00 ksof t I rqd_OPU0
... ...
3206 ?                  00: 00: 00 smbd
3215 ?                  00: 00: 00 smbd
3229 pt s/ 5            00: 00: 00 vul ner abl e
3257 pt s/ 5            00: 00: 00 server
3258 ?                  00: 00: 00 i n. Tel net d
3259 ?                  00: 00: 00 login
3260 pt s/ 3            00: 00: 00 bash
3289 pt s/ 3            00: 00: 00 su
3290 pt s/ 3            00: 00: 00 bash
```

Fig. 2: Output before hacking sys_getdents64 ()

```
Pl D TTY                Time CMD
  1 ?                   00: 00: 00 I ni t
  2 ?                   00: 00: 00 kevent d
  3 ?                   00: 00: 00 kapmd
  4 ?                   00: 00: 00 ksof t I r qd_OPU0
... ...
3206 ?                  00: 00: 00 smbd
3215 ?                  00: 00: 00 smbd
3229 pt s/ 5            00: 00: 00 vul ner abl e
3258 ?                  00: 00: 00 i n. Tel net d
3259 ?                  00: 00: 00 login
3260 pt s/ 3            00: 00: 00 bash
3289 pt s/ 3            00: 00: 00 su
3290 pt s/ 3            00: 00: 00 bash
```

Fig. 3: Output after hacking sys_getdents64 ()

The system call for querying file information is sys_getdents 64(). When querying the related information of files, Linux kernel calls sys_getdents64() to execute checking and sends the result to the program running in user space. At this moment, we could capture the system call and cut off that information relative to the appointed file, this relative information wouldn't be send to user space. In this example, what we need to do is not to send the information containing server.

After sys_getdents64() is hacked, the output of command ps-A is shown in Fig. 3. Contrasting to Fig. 2, we could find that the server progress doesn't be displayed. Even if using ls on /proc, we can't see the numerically directory 3257. Hiding process succeeds.

It is fatal for system manager if other system calls are captured and modified further. This OS can't be trusted any more. Such as, capturing sys_write() to hide appointed screen output, capturing sys_kill() to make manager not kill Trojan progress, capturing sys_getuid() to configure a user get the root shell.

Further more, we can hack sys_query_module() to hide the LKMs self. So, even system manager find some questions, he can't use command lsmod find the backdoor easily.

FIND THE HIDED PROGRESSES

Principle: Although LKMs backdoors are strong and hard to find, there is a flaw for these rootkits. That is they only alter the display of output, those hided files still exist and are available. In the previous example, the process whose PID is equal to 3257 has been hided, but we can use cd /proc/3257 to enter this directory and use ls-l to read all subdirectories and regular files just like Fig. 4.

It should be explained that why using ps we couldn't find process server, but using ls-l we could see /cn_server, /cn_server/server. The link file's content is the path of linked file and sys_read() is used to read it. If only this system call hasn't been modified, we could get the right information of process.

In this way, if we hide a LKMs named module.o by hacking sys_query_module(), we can still use rmmod module delete this module successfully. Although in fact, except this one who loads the LKMs, nobody knows what this name of module would be. So, don't expect that system manager could delete hided LKMs by method of exhaustion. As we all known, this name of module maybe consists of 1 to 255 random number, letters and some especial characters. This course is unpredictable.

However, we can use method of exhaustion to obtain the list of all processes because each process's PID is just a number. We could get all actually existing progresses information by orderly checking each digital directory from 1 to a very big number such as 100,000 within /proc. For current processor it spends less than one second, so this course is predictable and controllable completely. Some digital directory has been come into, but its PID isn't displayed by command ps-A. We could draw a conclusion that there must be hided process running. This is the principle of looking for hided progresses. Actually, because the number of processes may be very large, it will take system manager many time and energy to finish this work by hand. We should use some computer language to finish it.

Perl is a powerful script language, which could do any things almost. Perl is used for dealing with WEB, database, XML and systems manage (Edward *et al.*, 1998). It is fast, simple and especially fit for dealing with character string. So we use Perl to realize this function. The basic work flow chart is shown in Fig. 5.

Test: We test this program on Red Hat 9, kernel version 2.4.20-8, Perl version 5.8.0.

Before loading LKMs we run this perl program, the result is shown in Fig. 6. The first column is the PID from traversing 1 to 20000 digital directories within /proc; the second column shows the current directory of process; the third column shows the complete executable path of

-r--r--r--	1 root	root	0 9□ 2	10: 43 cmdl I ne
l r wxrwxrwx	1 root	root	0 9□ 2	10: 43 cwd -> / cn_server
-r-------	1 root	root	0 9□ 2	10: 43 environ
l r wxrwxrwx	1 root	root	0 9□ 2	10: 43 exe -> / cn_server/server
dr-x------	2 root	root	0 9□ 2	10: 43 f d
-r--r--r--	1 root	root	0 9□ 2	10: 43 maps
-rw-------	1 root	root	0 9□ 2	10: 43 mem
-r--r--r--	1 root	root	0 9□ 2	10: 43 mount s
l r wxrwxrwx	1 roor	root	0 9□ 2	10: 43 root -> /
-r--r--r--	1 root	root	0 9□ 2	10: 43 st at
-r--r--r--	1 root	root	0 9□ 2	10: 43 st at m
-r--r--r--	1 root	root	0 9□ 2	10: 43 st at us

Fig. 4: Structure of process server

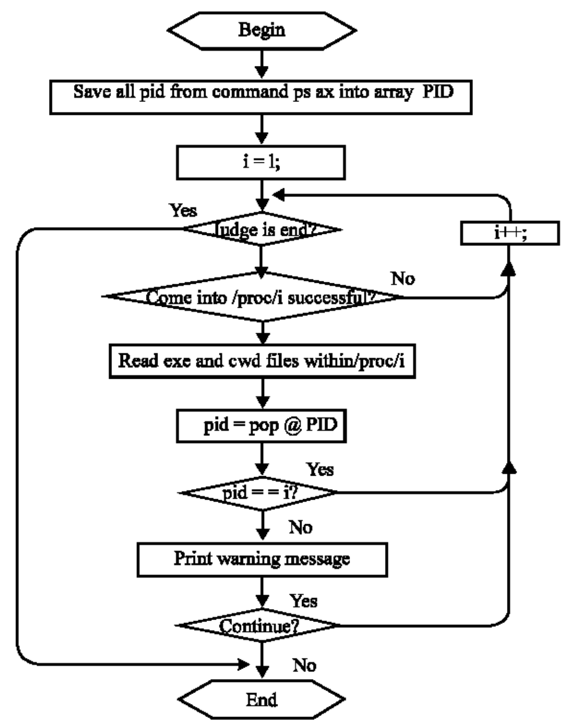

Fig. 5: Work flow chart

PI D	cwd	exe	pi d	I nf
=======	============	==========================	===========	=========
1	/	/ sbi n/ I ni t	1	
2	/	exe	2	
3	/	exe	3	
4	/	exe	4	
5	/	exe	5	
… …				
3215	/home/zhout ao	/usr/local/samba/sbi n/smbd	3215	
3257	/cn_server	/cn_server/server	3257	
3258	/	/usr/sbi n/i n. tel net d	3258	
3259	/	/bi n/login	3259	
3260	/home/ cn	/bi n/ bash	3260	
3289	/home/ cn	/bi n/ su	3289	
3290	/home/yuan	/bi n/ bash	3290	
… …				

Fig. 6: Output before loading LKMs

Pl D	ced	exe	pi d	1 nf
1	/	/sbi n/i ni t	1	
2	/	exe	2	
3	/	exe	3	
4	/	exe	4	
5	/	exe	5	
... ...				
3215	/home/zhoutao	/usr/local/samba/sbi n/smbd	3215	
3257	/cn_ server	/cn_server/server		warning
3258	/	/usr/sbi n/i n. tel net d	3258	
3259	/	/bi n/login	3259	
3260	/home/cn	/bi n/bash	3260	
3289	/home/cn	/bi n/su	3289	
3290	/home/yuan	/bi n/bash	3290	
... ...				

Fig. 7: Output after loading LKMs

program; the forth column is the PID number coming from command ps ax; the last column shows the warning message. The two methods get same result in this time, so there is not warning message.

After loading LKMs, process server has been hidden. But the Perl program comes into /3257 directory and gets relative information successfully. By contrasting to output of ps ax, it finds there is no process whose PID is equal to 3257, so the warning message is printed. The result is shown in Fig. 7.

From the above figure, system manager could get enough information: there is a hided process running in system, its name is server; the source code is located in /cn_server/. According to this information the manager can find and analyze the code to confirm the attributes of program.

CONCLUSION

Now, there are many tools used to check LKMs backdoors like KSTAT, Chkroot and so on. Among them KSTAT is more excellent. By comparing the address of current system call with the original one, it could find whether the system has been loaded rootkits or not. It offers many parameters and supports lots of functions. But the existent question is that it has to be complied on a newly complied and secure kernel for obtaining the original address of system calls. If KSTAT is used after the system having been intruded in, it does nothing. However, the method presented in this article utilizes the flaw that the hided files and directories could be accessed and the processes directories are number. By comparing the processes list got from exhaustion method to that one from command ps ax, the hided process would be found. And it could be used after the system having been loaded LKMs. The test shows that the result is correct. Moreover, by printing the existing directory and the complete running path of every process, it is helpful for system manager to locate quickly and analyze the program with threat.

REFERENCES

Edward, S. Peschko, 1998. Michele DeWolfe. Perl 5 Complete. McGraw-Hill, pp: 35-36.

Pragmatic, 1999. Complete Linux Loadable Kernel Modules version1.0. http:// www. xfocus. net/ articles/200008/47.html.

Rob Flickenger, 2003. Linux Server Hacks. O'Reilly, pp: 104-106.

Shi Jinqiao, 2003. Linux System Call Hijacking: Technical Principles, Application and Detection. The Computer Engineering and Applications, pp: 167-170.

Stealth, 2003. Kernel Rootkit Experiences. http://www.xfocus.net/articles.

YuanYuan, 2005. Design and Implementation of Linux's Security Monitor Base on LKM. Application Res. Computers, pp: 131-133.

Zhao Jong, 2004. The Complete Remark of Linux Kernel. China Machine Press, pp: 56-62.