2016 International Conference on Applied Mathematics and Mechanics (ICAMM 2016)
ISBN: 978-1-60595-399-1

Intrusion Prevention Method on LKM (Loadable Kernel Module) Backdoor Attack

Ji-Ho CHO, Han LEE, Jeong-Min KIM and Geuk LEE*

Dept. of Computer Engineering, Hannam University, Dae-Jeon, South Korea

*Corresponding author

Keywords: Loadable kernel module, Backdoor, Intrusion prevention.

Abstract. The current backdoor program is executed in user mode, which is called application mode, it is possible to find backdoors by the integrity check of system file. However, for the backdoor program is executed in kernel module, it is impossible to find its existence by the integrity check of system file. Current detection system has limitation to detection this LKM (Loadable Kernel Module) backdoor because they just examine the changes on the System Call Table. In this paper, we suggest the method using log file and password to overcome the limitation which the current integrity check system can't prevent attack using the kernel module.

Introduction

With the development of computer and networking technology and rapid diffusion of internet culture, the society is rapidly changing to the knowledge and information society. Our country, being equipped with strong internet infrastructure compared to other country, has various and numerous user. While sharing information, extending social, cultural and economic exchange and providing conveniences of life through internet, those users are improving the quality of life in various aspects and get affirmative effect.

Due to openness in design, fundamental problem of TCP/IP protocol and vulnerability of information system itself, there are misuse and tapping of accessed information and fabrication and alteration of information by malicious intrusion. These problems may have possibility to be expanded to serious social problem such as causing paralysis of information system of individual, enterprise or government or can be used as tool for a war further. Recently, the cases of intrusion to Linux, which is open operating system having with increased number of users, are likely to be increasing. Linux works by system call. Creation and deletion of file and instruction for the user permission are performed by the system call. Therefore, intrusion to Linux system can be made through system call [1]. So, the invader attempts several approaches to modify the system call table. One of them is to install backdoor using LKM (Loadable Kernel Module) secretly in order for re-access to the modified kernel without the password. The existing backdoor used the method changing the key system files such as ls, ps and netstat, but this method could be easily detected by the integrity check tool like Tripwire and MD5. However, this LKM backdoor, as a high level kernel backdoor, is impossible to be detected using the existing detecting tool or method. Therefore, this study will focus on the detection and interception of Kernel backdoor. Through the research related to Kernel LKM backdoor, the technique for detection and interception can be found and the system damage resulted from modified Kernel can be protected.

Related Study

Kernel

Linux kernel, as core element of operating system, is a control program to allow the user programs to use the resources in computer, and its functions comprise process management, management of file system and memory and network.

Most users make system call by using C library. Processes control file sub system through system calls related to the file such as open, close, read, write, stat, chown and chmod. The file system makes an access to the data flow between Kernel and secondary memory device through buffer. The block input/output device means random access memory device. As the device driver which does not use Cache feature is text device driver, those except block device belong to this. The system calls used by processor control sub system comprise fork, exec, exit, wait, brk, signal, kill, setpgrp, and setuid. These control communication between processors, scheduling and memory.

Consequently, we can find that Linux system works through the system calls. It reads, writes the files and controls the processor by using system call.

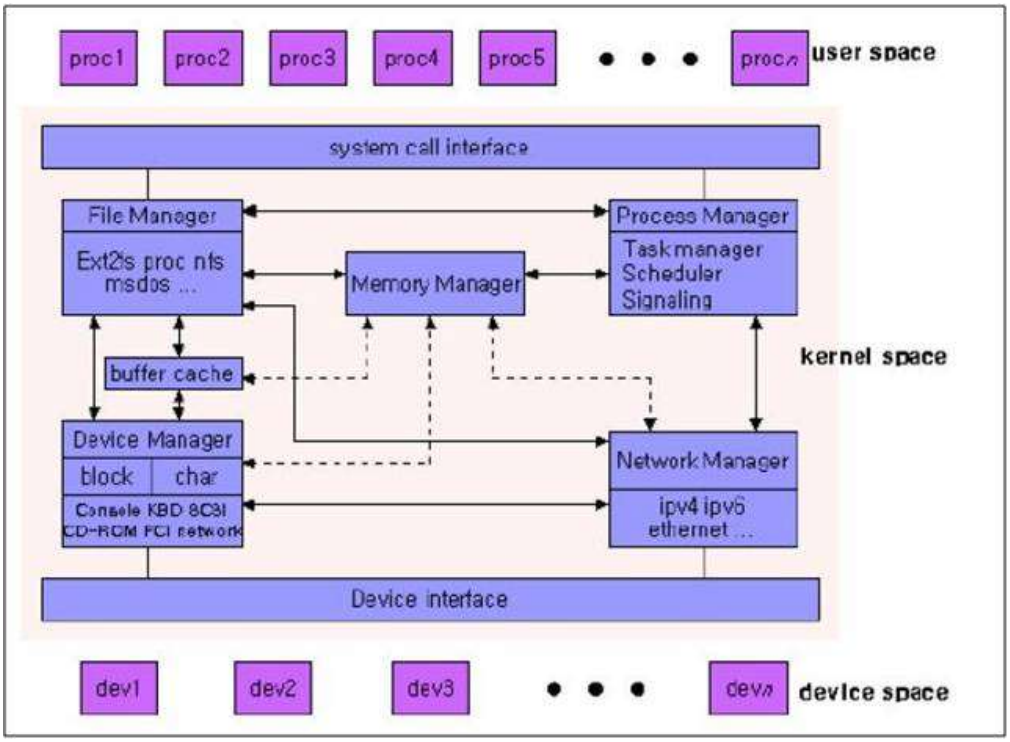

Figure 1. Linux Kernel.

System

The System call corresponds to the interface between Kernel area and User area. The Kernel uses separate memory space to protect the Kernel area from the mistake by single user or multiple users. Therefore, to control the hardware in user area, the system call which is an interface provided by the Kernel should be used because of inaccessible to the hardware directly. If a certain program opens a file to edit, for example, it reads inode information from file system and uses sys_open() internally which is a system call to return the writer of file [3].

LKM (Loadable Kernel Module)

Sometimes, it is necessary for the Kernel, the core element of operating system, to add a new module for additional device. For this reason, LKM (Loadable Kernel Module) could be supported and the typical case is Darwin, a kernel of Mac OS X. In this case, most of Kernel consists of modules so that it can be loaded and unloaded if necessary. This is called as Micro Kernel, while the traditional Unix (including Linux, too) is called Monolithic Kernel, which is impossible to load new module. However, most of Monolithic Kernel had been changed to make some modules to be loaded for Kernel expansion. The module used at this time is called as LKM module [4].

This LKM can be linked to the Kernel any time after system boot and the loaded module becomes part of Kernel like the other Kernel. This function of LKM is provided to various Unix systems such as Linux, FreeBSD, and Solaris. The module has same authority and responsibility as Kernel code. In other words, the Unix Kernel module may destroy the Kernel like all Kernel codes or device driver.

LKM Kernel Backdoor

Kernel Backdoor Using LKM

The backdoor installed at Kernel area misuses the feature of LKM which was developed for Kernel expansion. The fundamental concept of LKM backdoor is that the invader makes his own Kernel module and lets the module intercept the normal system call by insmod into Kernel area to alter normal operation such as hiding a certain process.

If the system that a conventional RootKit is installed as shown in figure 2 has a file with right checksum value and checksum program such a s tripwire with secured integrity, it will be able to check the abnormalities by calculating the checksum valve of major programs such as login and ps. Further, if it terminates the process and deletes the file further, it can be simply eliminated. But, the problem of Kernel backdoor exists in that it makes normal program works as a backdoor without change on application in user area [5]. As no change is made in application program, the symptom of abnormality cannot be detected by the checksum value of the program.

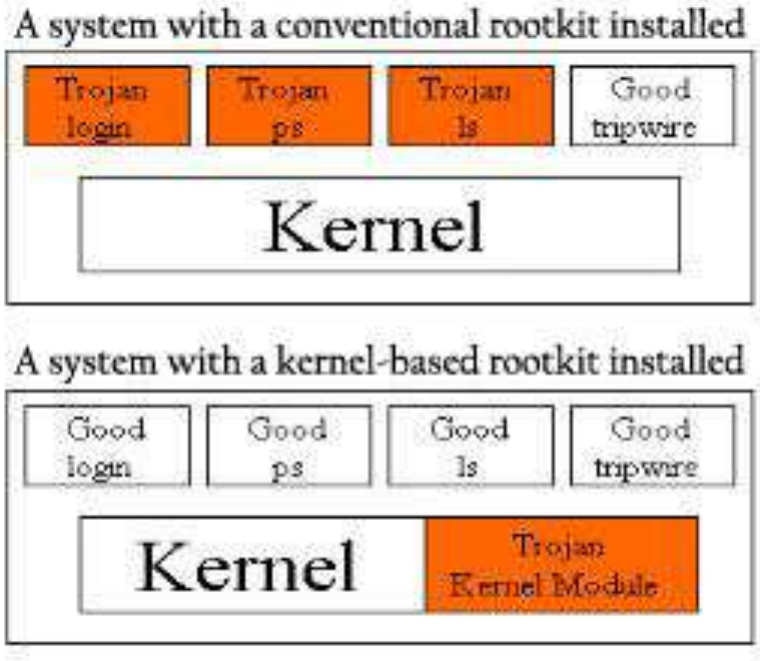

Figure 2. Kernel backdoor.

Therefore, the backdoor inserted in the Kernel cannot be easily detected and eliminated. In addition to that, for the case of Kernel module, it is not deleted until reboot the system, and it is difficult to remove because the Kernel backdoor changes the system call related not to be removed. In the case of system which should be running for 24 hours, this problem is more serious. To settle down this problem, the system should be shut down and rebooted for temporary service suspension, and should remove the script which loads the Kernel backdoor onto the Kernel possibly hidden in the starting script.

Intercepting the System Call of Kernel Backdoor

As the system call should be called in the user area not in the Kernel area which it is created, the interface of system call was exported. As discussed previously, the system call has its own number and stored own address on sys_call_table to be called. So, to call the system call from user area, it is called by input of system call name into sys_call_table[] and getting its address. For example, following code can be used to call sys_open() function.

$$openfunc = sys\_calll\_table[sys\_open] \quad (1)$$

Kernel Symbol

Up to now, the system call which has backdoor was replaced by intercepting the system call in the system call table. For this, the loadable Kernel module which the backdoor was planted was made to load the Kernel with root authority and replace the system call. There is lsmod commander is one of the commanders which control the Kernel modules. This command is to show the current module loaded. Therefore, the loaded module to plant the backdoor can be revealed by a single command of 1smod. So, it requires another way to hide it.

Kernel symbol table is an array of the loaded Kernel registered similar to the system call table. The module names listed in the Kernel symbol table can be found to be listed like a file under /proc/ksyms directory or /proc/module/.

Hiding the Kernel Module

The experienced manager will eliminate immediately the exceptional module when he sees the name of module listed on this. Therefore, it is necessary to hide the modules which make backdoor. We want to find how to hide it. There are various ways to hide the Kernels from Kernel symbol table. The fundamental method among them is to initialize the name of module as 0 or NULL during the course of module initialization. Let's look at the simple module initialization.

```
/*from Phrack // or whatever register it is in
*(char*)mp->name=0;
mp->size=0; mp->ref=0;
```
(2)

With above statement, when the module is queried, not only can it be visible but also removed. The better alternative is to intercept the system call, sys_query_module() which can be commonly found while tracing 1smod and rmmod of system call. Intercepting this system call is the same to that of do sys_write() call. It is to filter out all the processing requests related to the planted modules by intercepting nsys_query_module. As this is the system call used commonly by 1smod and remmod, if the system call is intercepted by this, it cannot be identified nor removed from /proc/module [6].

Intrusion Prevention Technique of LKM Kernel Backdoor

The basic method to protect the LKM based backdoor is to compile the monolithic Kernel which the loading function of compiler's module is completely deleted. This is simple and trustworthy but, in the case that the Kernel need to be expanded by adding new device, the total Kernels should be re-compiled and the system should be re-started. In most cases, the server may not be indeed restarted except in serious condition. In such situation, the server administrator may go through a rough patch. Therefore, the examples to be presented later will include how to protect the Kernel backdoor without removing of Kernel's module function and how to detect the backdoor if it is installed.

Logging Loaded Kernel Modules' Names to the Log File

For the loaded Kernel, its information is found under /proc/module/ but, it can be hid and removed by the various methods as described above. Therefore, it is of great advantage to maintain a separate log file. To record the name of loaded Kernel into log file, system call called sys_create_module () should be altered. As this can be called whenever the module is being loaded, as long as we input the code to record the name of loaded module, the existence of backload will be revealed no matter how the invader attempts to hide it.

Module Verification by a Password

This method is also to alter sys_create_module () which loads the module on the Kernel. Differently from previous method, however, the administrator or invader should follow the authentication procedure when they install new module on the Kernel. At this time, the user should be certified by the pre-defined password to load the module. The problem resides on how the input password by the administrator can be transmitted to the loaded verification module on the Kernel safely without being

tapped in the middle. To make it simple, the administrator may load the Kernel by keying in the password on the configuration file, but, the stability is very low because the invader can easily get the password.

```
#define MODULE
#define __KERNEL__

#include <linux/module.h>

#include <linux/malloc.h>

extern void* sys_call_table[];

int (*orig_create_module)(char*, unsigned long);

int hacked_create_module(char *name, unsigned long size)
{
 char *kernel_name;
 char hide[]="ourtool";
 int ret;

 kernel_name = (char*) kmalloc(256, GFP_KERNEL);
 memcpy_fromfs(kernel_name, name, 255);

 printk("<1>      SYS_CREATE_MODULE      :      %s\n",
kernel_name);

 ret=orig_create_module(name, size);
 return ret;
}

int init_module(void)                    /*module setup*/
{
 orig_create_module=sys_call_table[SYS_create_module];
 sys_call_table[SYS_create_module]=hacked_create_module;
 return 0;
}

void cleanup_module(void)          /*module shutdown*/
{
 sys_call_table[SYS_create_module]=orig_create_module;

}
```

Figure 3. Logging loaded kernel modules' names to the log file.

```
#define MODULE
#define __KERNEL__

#include <linux/module.h>
//~ 헤더파일 중략
#include <sys/stat.h>

extern void* sys_call_table[];

/*if lock_mod=1 THEN ALLOW LOADING A MODULE*/
int lock_mod=0;

int __NR_myexecve;

/*intercept create_module(...) and stat(...) systemcalls*/
int (*orig_create_module)(char*, unsigned long);
int (*orig_stat) (const char *, struct old_stat*);

char *strncpy_fromfs(char *dest, const char *src, int n)
{
char *tmp = src;
int compt = 0;

do {
dest[compt++] = __get_user(tmp++, 1);
}
while ((dest[compt - 1] != '\0') && (compt != n));

return dest;
}
int hacked_stat(const char *filename, struct old_stat *buf)
{
char *name;
int ret;
char *password = "password"; /*yeah, a great password*/

name = (char *) kmalloc(255, GFP_KERNEL);

(void) strncpy_fromfs(name, filename, 255);

/*do we have our password ?*/
if (strstr(name, password)!=NULL)
{
/*allow loading a module for one time*/
lock_mod=1;
kfree(name);
return 0;
}
else
{
kfree(name);
ret = orig_stat(filename, buf);
}
return ret;
}

int hacked_create_module(char *name, unsigned long size)
{
char *kernel_name;
char hide[]="ourtool";
int ret;
if (lock_mod==1)
{
lock_mod=0;
ret=orig_create_module(name, size);
return ret;
}
else
{
printk("<1>MOD-POL : Permission denied !\n");
return 0;
}
return ret;
}

int init_module(void) /*module setup*/
{
__NR_myexecve = 200;

while (__NR_myexecve != 0 && sys_call_table[__NR_myexecve] != 0)
__NR_myexecve--;
sys_call_table[__NR_myexecve]=sys_call_table[SYS_execve];

orig_stat=sys_call_table[SYS_prev_stat];
sys_call_table[SYS_prev_stat]=hacked_stat;

orig_create_module=sys_call_table[SYS_create_module];
sys_call_table[SYS_create_module]=hacked_create_module;

printk("<1>MOD-POL LOADED...\n");
return 0;
}

void cleanup_module(void) /*module shutdown*/
{
sys_call_table[SYS_prev_stat]=orig_stat;
sys_call_table[SYS_create_module]=orig_create_module;
}
```

Figure 4. Module verification by a password.

Conclusion

As the existing backdoor program is executed in user mode, which is application mode, it was possible to find the existence of backdoor by the integrity check of system file. However, for the backdoor using kernel module, it is impossible to find its existence by the integrity check of system file. Even various programs were presented to protect this LKM Kernel backdoor, there is limitation in protection as they examine the changes on the System Call Table. This paper, describing the risk of intrusion through the LKM Kernel backdoor, suggests a solution for the limitation which the existing integrity check could not prevent the attack through Kernel backdoor. Various new attacking methods by changing Kernel which is not mentioned in this paper may be under development. Further study on the various methods of intrusion prevention for those attacks would be necessary.

References

[1] Maurice J. Bach, The Design of the UNIX Operating System, Daeyoungsa, Seoul, 1992, pp.47-50.

[2] Aessandro rubini, Linux Device Drivers, 2nd Edition, O'Reilly, 2001.

[3] Daniel. P. Bover, Understanding the Linux Kernel, 2000.

[4] Pragmatic, Complete Linux Loadable Kernel Modules, 1999.

[5] HyunChul Jeong, Kernel based Rootkit Analyst Reports (2000).

[6] AEleen Frisch, Essential System administration, 3rd Edition, 2003.

[7] https://www.infoq.com/news/2013/10/Linux-Backdoor, A Look Back at the Linux Kernel Backdoor, 2013.

[8] https://www.skyhighnetworks.com/cloud-security-blog/, Protecting your company from backdoor attacks what you need to know, 2016.