# A Linux rootkit improvement based on inline hook

Jun Gu[1], Ming Xian[2,a], Tian Chen[3,b], Ruixiang Du[3,c]

CEMEE national key laboratory, National University of Defense Technology

Changsha 410000, China

email: 54185011@163.com

**Keywords:** Linux Rootkit; Inline hook; VFS

**Abstract.** Rootkit and its detection technology promoted mutually. This paper proposes a kernel rootkit level division method based on function call relationship. The deeper rootkit's level is, the more difficult its detection becomes. We put forward and implement a rootkit method based on inline hook. It pushes the rootkit down a lower level in the kernel space. Experimental results show our method has excellent performance on hiding malicious programs.

## Introduction

Rootkit first appeared in early nineties of the twentieth century, mainly deployed on the target machine to hide malicious code traces, such as process, file, backdoor port, the kernel module inserted by attackers. The users and the security software can't be aware of it and so that malicious code can be hidden in the target machine for long time [1]. The Linux kernel operating system is widely used as server system and handheld device system, Due to its open source and perfect system performance. However, compared with the windows system, the security of the Linux system is less researched. Security corporations take the commercial interests for consideration, so they do not put enough effort on the Linux security issues. The Linux Rootkit is not rare, such as adore, adore-ng[2], Knark[3], etc. If these rootkits are used for malicious attacks, the Linux system will be under threats. The purpose of this paper is analyzing the rootkit technology, understanding the principle and method of rootkit implementation and the shortcomings of the current detection technology, so as to prepare for the research of anti-rootkit technology.

This paper proposes a kernel rootkit level division method based on function-call relationship and puts forward a method based on inline hook to make a more covert rootkit.

## Related work and basic knowledge

**Inline hook technology.**

Inline hook is a function redirection method that we insert our own "JMP" instructions in the function start position and change the flow of execution. We can also use some other instructions, such as "PUSH EIP, RET" instructions pair, "CALL" instructions, etc., as indirect replacements for "JMP" instructions. In this paper, we use the JMP instructions to hook VFS level file and node operation functions and to change the execution flow.

**Linux rootkit technology.**

We can sort the Linux rootkit into two types based on its work mechanism: application level rootkit and kernel level rootkit [4]:

***1) Application rootkit***

Early rootkit works in the application level, Linux system general integrated PS, LS, netstat tool, used to view system processes, files, opened ports and other information. For example, user terminal inputs the PS command, system shell in the / bin directory finds and executes the PS executable file, then print out the information of the current active processes. Executable files integrated by the system are not only the basement of shell scripts, but also are important tools of Linux users to view, and maintain system. The earliest rootkit[4] replaces login, PS, nestat and other system tools, allowing attackers to login system as root user, to open ports and to hide the

© 2016. The authors - Published by Atlantis Press 

operation processes. This type rootkit allows the attackers to use their tools to replace system tools. When users view system information, malicious information is filtered out by rootkit. But there are some shortcomings with this type of rootkit: system update will restore replaced tools; third-party tools can bypass the rootkit to obtain complete information. It is very easy to detect this type of rootkit through comparison of file hash values.

***2) Kernel level rootkit***

Kernel level rootkit embeds code in the system kernel, usually by loading the kernel module into kernel, or modifying the kernel memory area such as /dev/mem, /dev/kmem etc. Kernel level rootkit can sort into two types: kernel hook and direct kernel object operation. Kernel hook is also referred as kernel function redirection. User mode programs usually should call the kernel functions, such as system call, virtual file system functions etc. Kernel hook is to rewrite the kernel functions or to make the kernel functions redirected to attacker's new functions. When users call these functions, the actual execution flow will be hacked.

**Linux VFS file system.**

A significant feature of Unix is that everything is a file. As the most successful "variant" version of Unix, Linux is completely inherited this idea. File system is a method collection of operating systems used to organize, manage and maintain of storing files on a computer storage medium[5]. Modern operating system also supports several different file systems. In order to keep the Linux operating system's modularization and provide a unified programming interface for different file systems. Linux provides a pure software based on virtual file system called VFS. VFS kernel supports three types of file system [6]:

***Disk based file system:*** it is a traditional file management system stored on the hard disk, such as floppy disk and CD-ROM. Its typical representatives include ext2 / 3, ReiserFS, XFS, UFS, is09660 etc.

***Virtual file system:*** Instead of storing on the hardware, it is just communication method between application program and users. Proc is a typical representative of VFS file system. Though looking up the proc file system size, we can found that its size is 0. Only by reading the contents of the file, can the system generate corresponding information.

***Network based file system:*** if Linux needs to access and process the data on other computers by network, it also need transparent the difference between local and network documents by VFS. Figure 1 shows the location of VFS in Linux system.

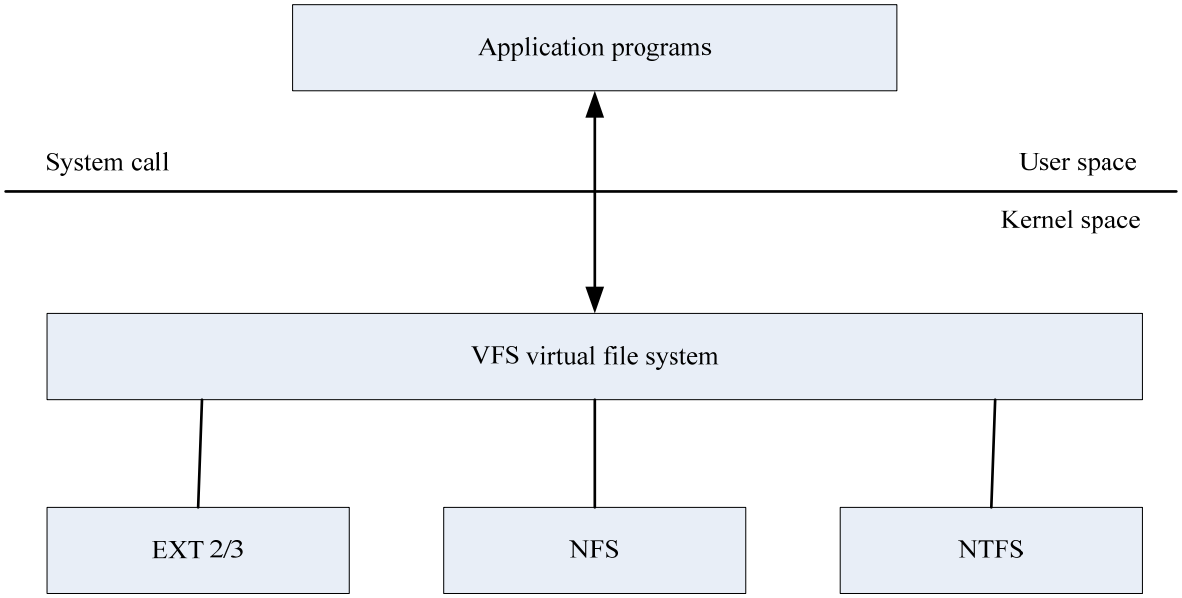


Fig.1 VFS file system

From the view of programming, VFS can be interpreted as an interface for system call. The specific file systems supported by VFS is different implementations of this interface.

## VFS level rootkit technology based on inline hook

**Function call relationship.**

While application level program invokes interface function, for example, read function. The system will invoke the "sys_read" system call function. Figure 2 shows the code of "sys_read". In function "sys_read", "fget_light" function is called to get "file" structure pointer and "vfs_read" function is called to realize real read operation.

```
asmlinkage ssize_t sys_read(unsigned int fd, char __user * buf, size_t count)
{
        struct file *file;
        ....
        file=fget_light(fd,&fput_needed);
        if (file) {
        loff_t pos = file_pos_read(file);
        ret = vfs_read(file, buf, count, &pos);
        ...
        }
        ...
}
```

Fig.2 "sys_read" function core code

Figure 3 shows the code of "vfs_read". When excution flow enter into "vfs_read" function, system will call read function pointers in f_op(a file_operations structures) bases on different file system (ext, nfs, proc, proc/net etc.).

```
ssize_t vfs_read(struct file *file, char __user *buf, size_t count, loff_t *pos)
{
        ...
        if (!ret) {
                if (file->f_op->read)
                        ret = file->f_op->read(file, buf, count, pos);
                else
                        ret = do_sync_read(file, buf, count, pos);
                        ....
                }
        ...
}
```

Fig.3 vfs_read function core code

The file_operations is a structure that contains file operation function pointers. Figure 4 shows some of its members. The inode_operations is similar with file_operations.

```
struct file_operations {
        ...
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ...
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
        ...
        int (*readdir) (struct file *, void *, filldir_t);
        ...
        int (*open) (struct inode *, struct file *);
        ...
        ssize_t (*readv) (struct file *, const struct iovec *, unsigned long, loff_t *);
        ssize_t (*writev) (struct file *, const struct iovec *, unsigned long, loff_t *);
        ssize_t (*sendfile) (struct file *, loff_t *, size_t, read_actor_t, void *);
        ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
        ...
};
```

Fig.4 file_operations structure

Sometimes the function pointer needs a callback function pointer as a parameter. For example, “readdir”. This callback function does our real kernel operations.

By analyzing the system call function and kernel code, we can figure out the function call relationship in the system call. It is shown in Figure 5.

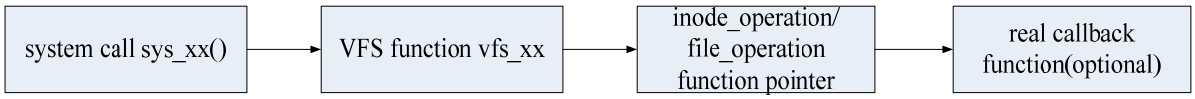


Fig. 5 function call relationship in the system call

**Kernel rootkit level division.**

According to the function call relationship. The kernel rootkit is divided into four levels.

The first level is system call table redirection .This method is to camouflage a system call table, replace the original system call table. The typical rootkits in this level is KIS[7].

The Second level is system call function redirection which replaces the system call function addresses in the system call table. The typical rootkit in this level is enyelkm[8].

The third level is the VFS function redirection, each system call completes the operation by calling VFS function, so the replacement of VFS function can also achieve hijacking the system. Paper[9] put forward this method.

The fourth level is the replacement of function pointers in inode_operations or file_operations structure based on different file system. Adore-ng[2] replaced the function pointers in these two structures and part of the callback function by this method. Adore-ng is hidden in a deep kernel level. This approach is very flexible and can’t be found easily. Anti-virus programs can monitor system call, but can’t be simply monitoring the function address. Because as long as the registration of a new file system, then there will generate a new address. The system is difficult to take on the anti-virus software’s overhead. But paper [10] proposed a method based on the limited range of normal kernel function pointer.

In this paper, we will propose and implement an improvement on the fourth level kernel rootkit by inline hook. We do not modify function pointers in inode_operations and file_operations structures, but lower down the level of rootkit by changing first five bytes of the address where the function pointers are located. Figure 6 shows the flows of the hook steps.

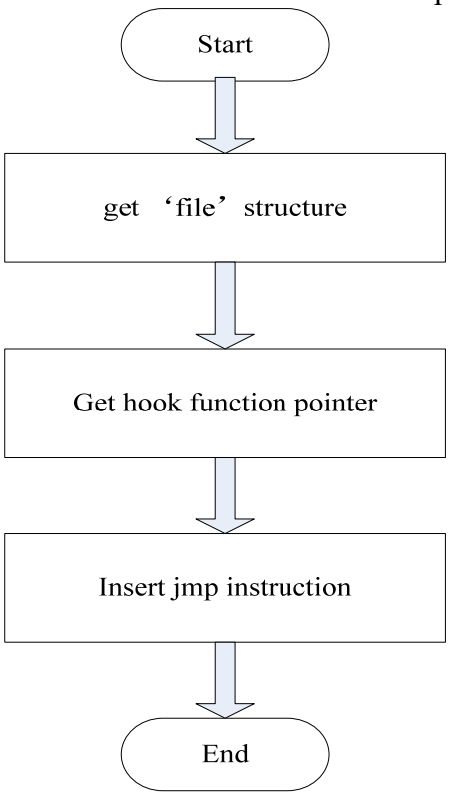


Fig.6 the hook steps

The steps to realize this method are as follows:

(1) Get the file structure pointer. We get this pointer by the “filp_open” kernel function. There are three parameters in this function. We need to pay more attention to the first parameter. It is a string pointer that points to the file directory we use.

(2) Get the function pointer we want redirect. By step 1, we have got the “file" structure pointers. There are two special structures in the “file” structure .One is inode_operations structure which contains node operation function pointers. The other is file_operations structure which contains file operation function pointers. We can find the needed function pointer from these two structures.

(3) Modify first five bytes of the address where the function pointer points to. In this step, we should write a function for our “JMP” instruction. Then we should write “JMP" instruction to replace first five bytes. Particularly to our “JMP" instruction, it is start with the byte ‘0xe9’. So it is a short jump instruction. The address follows is a relative address. We should do some transformation. Formula 1 shows the transformation method.

JMP relative address=JMP absolute address - (JMP instruction address + 5) (1)

(4) Write our hook function based on our goals .We should decide whether return to original function or not. If it is not needed, the hook function can return immediately, otherwise the hook function should restore the first five bytes to the former bytes and JMP to the original address, then repeat step 3 for next time. The pseudo codes in figure 7 show the execution flow of our hook function:

```
Hook function pseudo code
bHide: a sign decide whether file/process should be hidden
pOriginal: the pointer of hooked function
pNewfun: our hook function,the pointer of this function
if bHide==false do
        return
else do
        Recover first five bytes of pOriginal
        CALL pOriginal
        Change first five bytes of pOriginal to  “JMP pNewfun-pOriginal-5”
        return
end if
```

Fig.7 The hook function

## Experiment

The Experiment platform is 32-bit CentOS 5.3, kernel version is Linux 2.6.18. Experiment, on the basis of Adore-ng 2.6, we modify the “write” and “readdir” function in file_operations structures, and the “lookup” function in inode_operations structure through inline hook. Then we use the method in paper[10] to make detection. The result is shown as Table 1:

**Table 1:** experiment result

| File system mount point | File system | Function pointer | Detection result (Adore-ng 2.6) | Detection result (Our improvement) |
|---|---|---|---|---|
| / | Ext3 | file_oporations ->readdir | success | failed |
| | | file_oporations ->write | success | failed |
| /proc | proc | file_operations ->readdir | success | failed |
| | Ext3 | Inode_operations -> lookup | success | failed |

Form Table 1, we can draw the conclusion: our improvement can escape from the method referred by paper[10], making the rootkit more difficult to be detected than adore-ng 2.6 .

## Concluding remarks

The method proposed in this paper studies a deeper hidden rootkit, increasing the difficulty of the detection. According to kernel rootkit level division, we can sort our method as the fifth level rootkit. But if anti-virus software uses both the method in paper [10] and inline hook detection technology, it still can be detected. However doing this will greatly increase the burden of operating system. We can also use other instructions such as “CALL” and “PUSH/RET” to hook. Furthermore, according to the assembly code of the function, we want to hook, we can predict the function stack. If stack is figured out, it is possible to hook any address in the function, not just first several bytes. (We hook first serval bytes, because we should use the parameters in original function). If inline hook is not limited to first several bytes, it is almost impossible to be detected.

## References


[1] Andreas B. UNIX and Linux based rootkits techniques and countermeasures [J]. DFN-CERT Services GmbH, 2004.

[2] Adore-ng [EB/OL]. http://stealth.7350.org/rootkits/adore-ng0.56.tgz.

[3] Knark [EB/OL]. http://packetstormsecurity.com/UNIX/penetration/rootkits.

[4] Alisa S. Rootkit Evolution[EB/OL]. http://www.viruslist.com/en/analysis?pubid=204792016.

[5] Ricardo galli. Joumal file systems in Linux [EB/OL]. http://foulma.net/body.html? nldNoticia= 1154,2002-01-24.

[6] Bovet D P, Cesati M. Understanding the Linux Kernel, 3rd edition [M]. 208 • Incremental Chec-kpointing for Grids, 2003.

[7] KIS [EB/OL]. http://packetstormsecurity.or KIS(Kernel g/cgi-bin/search/search.cgi? Intrusion System) searchtype=archives&counts=126&searchvalue=rootkit++.

[8] enyelkm [EB/OL]. http://download.csdn.net/detail/shadow20080578/6372647

[9] Analysis and implementation of advanced Linux Kernel Inline Hook[EB/OL].http://security.cto-cio.com. cn/ tips/285/8840785_2.shtml

[10] ZHUANG Sihu, WANG Jian, ZHANG Fuxin. Detecting VFS Kernel Backdoor Software in Linux [J]. Application Research Of Computers, 2005, 22:194-196.