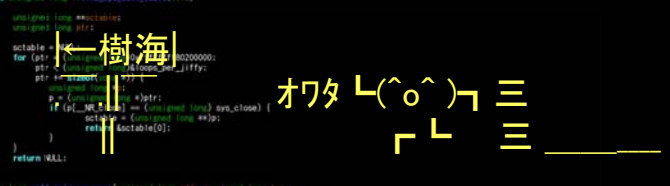
テストのためには手段を選ばず Linux編

株式会社エス・キュー・シー　市川正美
michikawa@sqc.co.jp

テストケースは書いたけど

- テスト対象に対して
  - メモリが確保できない状況なんてあまりない
  - ファイル読み込みに失敗とかまずしない
- 異常系のテストケースを書いても実行するのが困難だったりする

このときの気持ちを表すと



どうにかしてテストする

- 基本方針
  - テスト対象のソースコードは極力いじらない
  - バイナリエディタでバイナリをいじらない
- 何かしらのコードを書いて、エラーを起こしたいシステムコールをフックする

使えそうな手段

- LD_PRELOAD
- LD_AUDIT
- カーネルモジュールでシステムコールをフック
- *ionotify(7)*も使えるのでは

LD_PRELOAD＆LD_AUDIT

- 起動の仕方を変える必要あり
- ユーザ空間で動くからお手軽
  - setuidされているプログラムでは、LD_*系の環境変数は無視されるので使えない
- *dlopen(3)*、*dlsym(3)*、*dlclose(3)*と*__attribute__((constractor))*を使う
- 大概のことはこれでいける

libcフック

```
static const char const so_file[] = "/lib/libc.so.6";

void *(*orig_mmap)(void *start, size_t length, int prot, int flags,
                   int fd, off_t offset);

static void *handle;

static void __attribute__((constructor))
mmap_hook_init(void)
{
        handle = dlopen(so_file, RTLD_LAZY);
        orig_mmap = dlsym(handle, "mmap");
}

static void __attribute__((destructor))
mmap_hook_close(void)
{
        dlclose(handle);
}
```

カーネルモジュール

- *insmod(8)/modprobe(8)*を使う必要あり
- カーネル空間で動くから、ちょっとのミスで即死
  - *open*、*close*、*read*、*write*、*fork*、*mmap*あたりの扱いは気をつけましょう
- 2.6系のカーネルでは、2.4系の時のようにシステムコールテーブルを簡単に参照できない
  - カーネルのconfigで設定を変える
  - メモリ空間からアドレスを探す

カーネルモジュールフック

```
static unsigned long **find_sys_call_table(void)
{
        unsigned long **sctable;
        unsigned long ptr;

        sctable = NULL;
        for (ptr = (unsigned long) 0xffffffff80200000;
             ptr < (unsigned long) &loops_per_jiffy;
             ptr += sizeof(void *)) {
                unsigned long *p;
                p = (unsigned long *) ptr;
                if (p[__NR_close] == (unsigned long) sys_close) {
                        sctable = (unsigned long **)p;
                        return &sctable[0];
                }
        }
        return NULL;
}
```

カーネルモジュールフック

```
int __init syscall_hook_init(void)
{
        syscalls = find_sys_call_table();
        if (!syscalls) {
                printk("Couldn't find a system call table¥n");
                return -1;
        }

        /* set our own system call */
        orig_sys_mmap_addr = (unsigned long) syscalls[__NR_mmap];
        syscalls[__NR_mmap] = (void *) &syscall_hook_sys_mmap;

        /* remember original address */
        orig_sys_mmap = (long (*)(unsigned long, unsigned long, unsigned long,
                                  unsigned long, unsigned long, unsigned long))
                        orig_sys_mmap_addr;

        printk("syscall_hook  kernel module loaded with pid=[%i]¥n", current->pid);
        printk("sys_mmap's address is %lx¥n", orig_sys_mmap_addr);

        return 0;
}
```

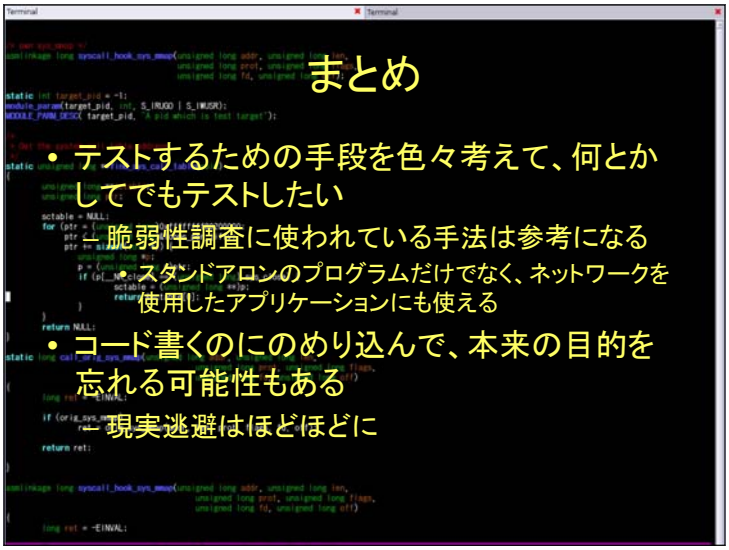
カーネルモジュールフック

```
void __exit syscall_hook_exit(void)
{
        syscalls[__NR_mmap] = (long *) orig_sys_mmap_addr;

        printk("syscall_hook kernel module ended with pid=[%i]¥n", current->pid);
        printk("sys_mmap's address is %p¥n", syscalls[__NR_mmap]);
}
```

ionotify(7)

- ファイルへのアクセスを見張ることができる
- 使いどころは難しいかも
  - *access(2)*から*open(2)*の間を狙って何かをするとか
    - アクセス権チェックとファイル操作までに時間があることを狙われた脆弱性が過去にある



まとめ

- テストするための手段を色々考えて、何とかしてでもテストしたい
  - 脆弱性調査に使われている手法は参考になる
  - スタンドアロンのプログラムだけでなく、ネットワークを使用したアプリケーションにも使える
- コード書くのにのめり込んで、本来の目的を忘れる可能性もある
  - 現実逃避はほどほどに