BI.ZONE

НИЯУ МИФИ

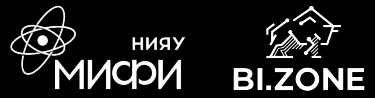
Введение в PWN 0x2

Спикер: Павел Блинников

Руководитель группы исследования уязвимостей BI.ZONE
Капитан SPRUSH
Админ MEPhI CTF

@pturtle

Что мы узнали на прошлом занятии?



1. Общая информация о компьютере, ОС и процессах
2. Как работает stack-based buffer overflow
3. Как работают простые ROP-чейны
4. Как писать простые эксплойты на pwntools
5. Как дебажить в gdb с pwndbg

Что мы узнаем сегодня?

НИЯУ МИФИ BI.ZONE

1. Похекаем ещё ропов
2. Посмотрим ещё на один класс уязвимостей
3. Потыкаемся с тулингом
4. Получим ещё

Recall: что такое ROP?

ROP – техника, при которой мы собираем свою программу из кусков взламываемой

Немного ROP-гаджетов есть в обычных бинарях, но очень много – в libc

libc – основная системная либа для исполняемых файлов на Си

Recall: ASLR

ASLR – защитный механизм, который загружает все библиотеки и регионы памяти по почти случайному базовому адресу

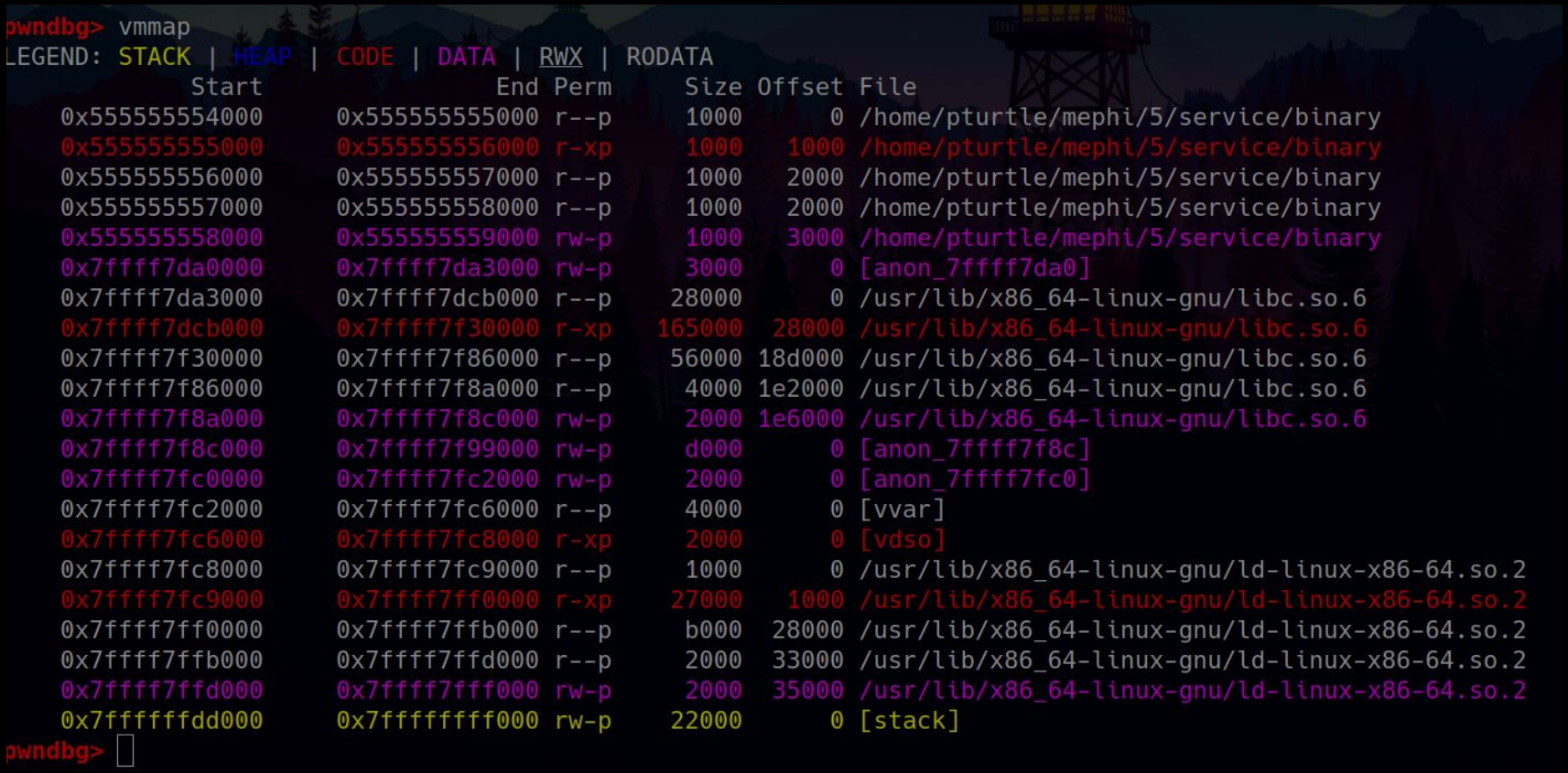
Угадать ASLR в userspace фактически невозможно, без дополнительной утечки сделать почти ничего нельзя (но кое-что можно, таск gparted)

Регионы памяти

В рантайме память разделена регионы.

Регион памяти – несколько логически соединенных страниц памяти.

Страница памяти – 0x1000 байт (4 Кб) идущих подряд


pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
Start End Perm Size Offset File
0x555555554000 0x555555555000 r--p 1000 0 /home/pturtle/mephi/5/service/binary
0x555555555000 0x555555556000 r-xp 1000 1000 /home/pturtle/mephi/5/service/binary
0x555555556000 0x555555557000 r--p 1000 2000 /home/pturtle/mephi/5/service/binary
0x555555557000 0x555555558000 r--p 1000 2000 /home/pturtle/mephi/5/service/binary
0x555555558000 0x555555559000 rw-p 1000 3000 /home/pturtle/mephi/5/service/binary
0x7ffff7da0000 0x7ffff7da3000 rw-p 3000 0 [anon_7ffff7da0]
0x7ffff7da3000 0x7ffff7dcb000 r--p 28000 0 /usr/lib/x86_64-linux-gnu/libc.so.6
0x7ffff7dcb000 0x7ffff7f30000 r-xp 165000 28000 /usr/lib/x86_64-linux-gnu/libc.so.6
0x7ffff7f30000 0x7ffff7f86000 r--p 56000 18d000 /usr/lib/x86_64-linux-gnu/libc.so.6
0x7ffff7f86000 0x7ffff7f8a000 r--p 4000 1e2000 /usr/lib/x86_64-linux-gnu/libc.so.6
0x7ffff7f8a000 0x7ffff7f8c000 rw-p 2000 1e6000 /usr/lib/x86_64-linux-gnu/libc.so.6
0x7ffff7f8c000 0x7ffff7f99000 rw-p d000 0 [anon_7ffff7f8c]
0x7ffff7fc0000 0x7ffff7fc2000 rw-p 2000 0 [anon_7ffff7fc0]
0x7ffff7fc2000 0x7ffff7fc6000 r--p 4000 0 [vvar]
0x7ffff7fc6000 0x7ffff7fc8000 r-xp 2000 0 [vdso]
0x7ffff7fc8000 0x7ffff7fc9000 r--p 1000 0 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
0x7ffff7fc9000 0x7ffff7ff0000 r-xp 27000 1000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
0x7ffff7ff0000 0x7ffff7ffb000 r--p b000 28000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
0x7ffff7ffb000 0x7ffff7ffd000 r--p 2000 33000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
0x7ffff7ffd000 0x7ffff7fff000 rw-p 2000 35000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
0x7ffffffdd000 0x7ffffffff000 rw-p 22000 0 [stack]
pwndbg>

Какие структуры данных критичны?

- GOT – Global Offset Table, секция для линковки между исполняемыми файлами в рантайме
- Всевозможные vtable: таблицы функций для виртуальных методов в C++
- Индексы и размеры массивов в структурах

За что мы боремся?

1. Перезапись критических данных (указатели на функции, размеры массивов, указатели на указатели)
2. Захват RIP
3. Создание RWX регионов для постэксплуатации

Как мы можем выиграть?

- Вызвать system(“/bin/sh”)
- Вызвать execve(“/bin/sh”, 0, 0)
- Создать ROP на чтение флага и вывод на экран
- Создать ROP на изменение прав на страницу памяти на RWX, записать свой шеллкод и вызвать его

GOT

Очень полезная для нас секция файлов, необходимая для линковке в рантайме.

В GOT есть указатели на функции в подгруженной либе (got.plt) и обертки для их вызова (plt)

```
pwndbg> x/32i puts
   0x1030 <puts@plt>:    jmp    QWORD PTR [rip+0x2fca]        # 0x4000 <puts@got.plt>
   0x1036 <puts@plt+6>: push   0x0
   0x103b <puts@plt+11>:         jmp    0x1020
```

GOT: RELRO

Partial RELRO – создает GOT перед .bss

Full RELRO – делает GOT полностью read-only

```
pwndbg> checksec
[*] '/home/pturtle/mephi/5/service/binary'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

DEMO домашки

НИЯУ МИФИ BI.ZONE

Немножко отдохнем

Исполняемые файлы бывают двух типов: статические и динамические

До этого мы работали с динамическими, но теперь посмотрим на статические

В статических libc вкомпилена в бинарь, поэтому не надо ликать адрес. А ещё...

Можно автоматически создать ROP!

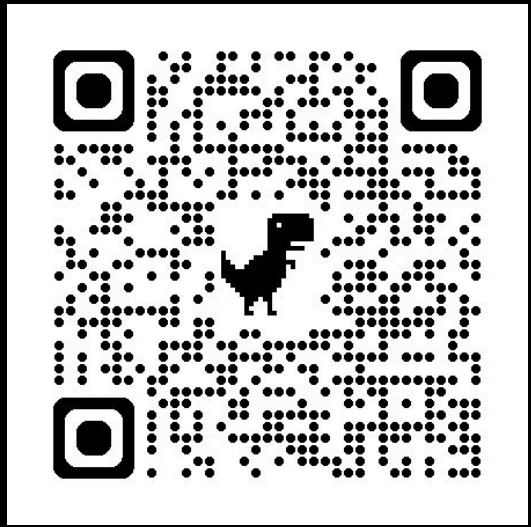
ROPgadget --binary ./binary --ropchain

DEMO!

Recall: системные вызовы

Системные вызовы – один из основных интерфейсов взаимодействия user-space и kernel-space.

Calling convention: номер в RAX, затем аргументы: RDI, RSI, RDX, R10, R8, R9



Seccomp

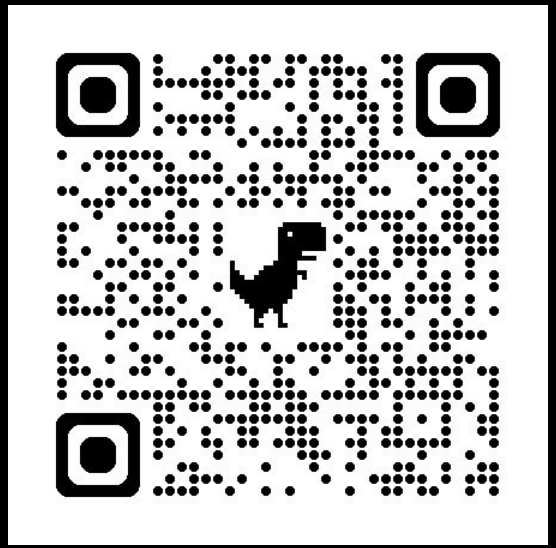
Функциональность Linux для ограничения возможности вызова сисколлов.

```
void seccomp_start() {
  scmp_filter_ctx ctx = seccomp_init(SCMP_ACT_ALLOW);
  seccomp_rule_add(ctx, SCMP_ACT_KILL, SCMP_SYS(execve), 0);
  seccomp_rule_add(ctx, SCMP_ACT_KILL, SCMP_SYS(execveat), 0);
  seccomp_load(ctx);
  seccomp_release(ctx);
}
```

Что такое шеллкод?

Шеллкод – небольшая часть кода, написанная на ассемблере, разработанная для постэксплуатации бинарных уязвимостей.

Пример шеллкода:



DEMO: пишем шеллкод

НИЯУ МИФИ BI.ZONE

Imagine arb read and arb write...

Что если бы у нас был arbitrary read и arbitrary write?

Imagine arb read and arb write...

Что если бы у нас был arbitrary read и arbitrary write?

Мы бы что-нибудь переписали, например GOT

DEMO

НИЯУ МИФИ

BI.ZONE