Введение в PWN

Спикер: Павел Блинников

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

@pturtle

whoami

- Играю CTF много-много лет
- Иногда читаю доклады на конфах
- 3 года в комп. криминалистике, год в Vulnerability Research
- Fuzzing enjoyer
- Выпускник 42 кафедры 2023 года

Зачем нам пывн?

1. Самый хакерский способ похека
2. Memory-unsafe языки всё ещё везде
3. В России на CTF очень плохо решают пывн
4. При некоторой доработке напильником из пывнера на CTF достаточно быстро получается ресерчер
5. Пывн – это путь самурая. Очень сложно разобраться, но усилия быстро окупятся

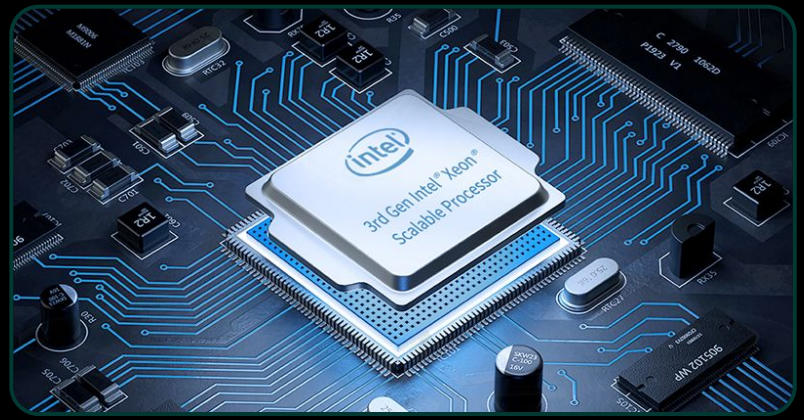
Из чего будет состоять курс?

1. Разберемся с тем как работают программы
2. Посмотрим на основные простые уязвимости
3. Поковыряемся в куче, разберем how2heap
4. Посмотрим на фаззинг
5. Если останется время, то разберем интро в сложные таргеты: ядро Linux и (возможно) браузеры

Как работает компьютер?

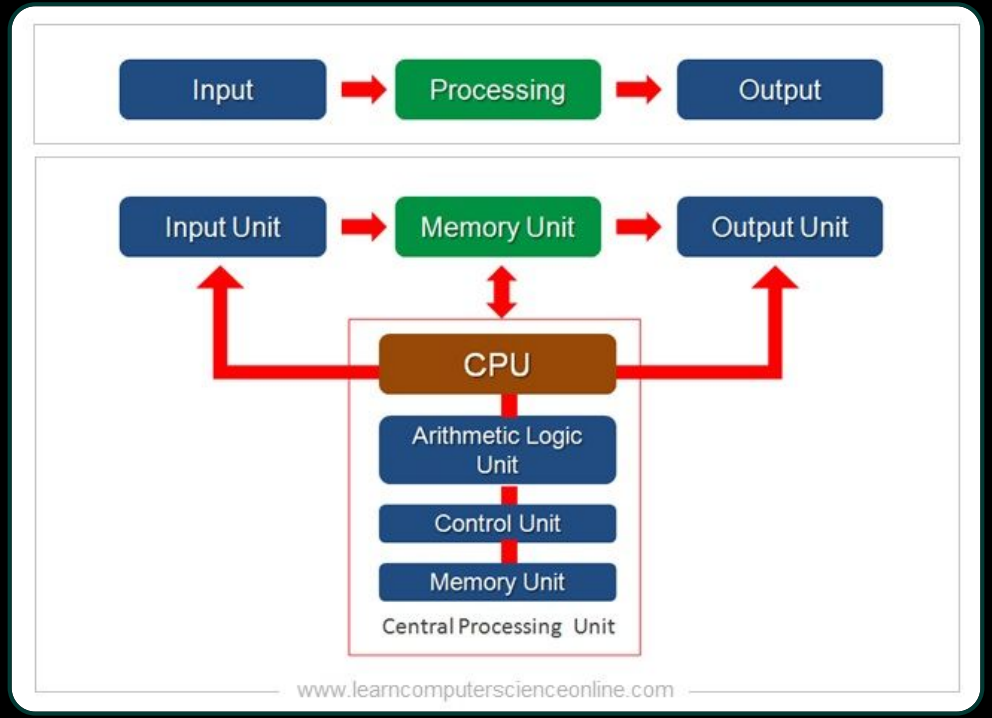
НИЯУ МИФИ BI.ZONE

Процессор



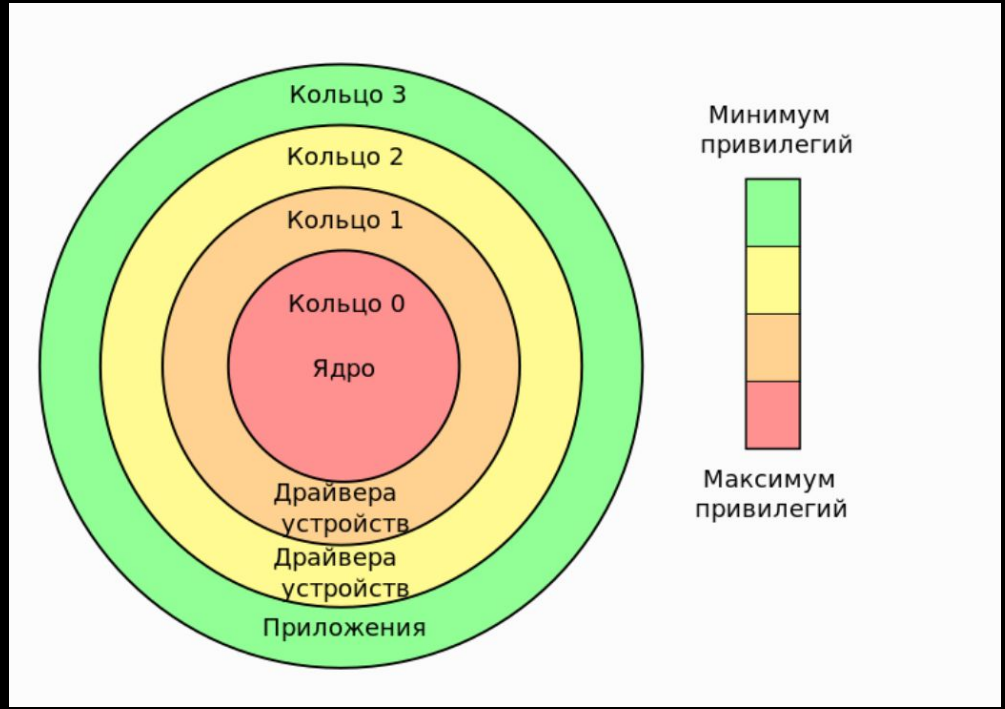
Оперативная память

Как работает компьютер?



Кольца защиты (Security Rings)

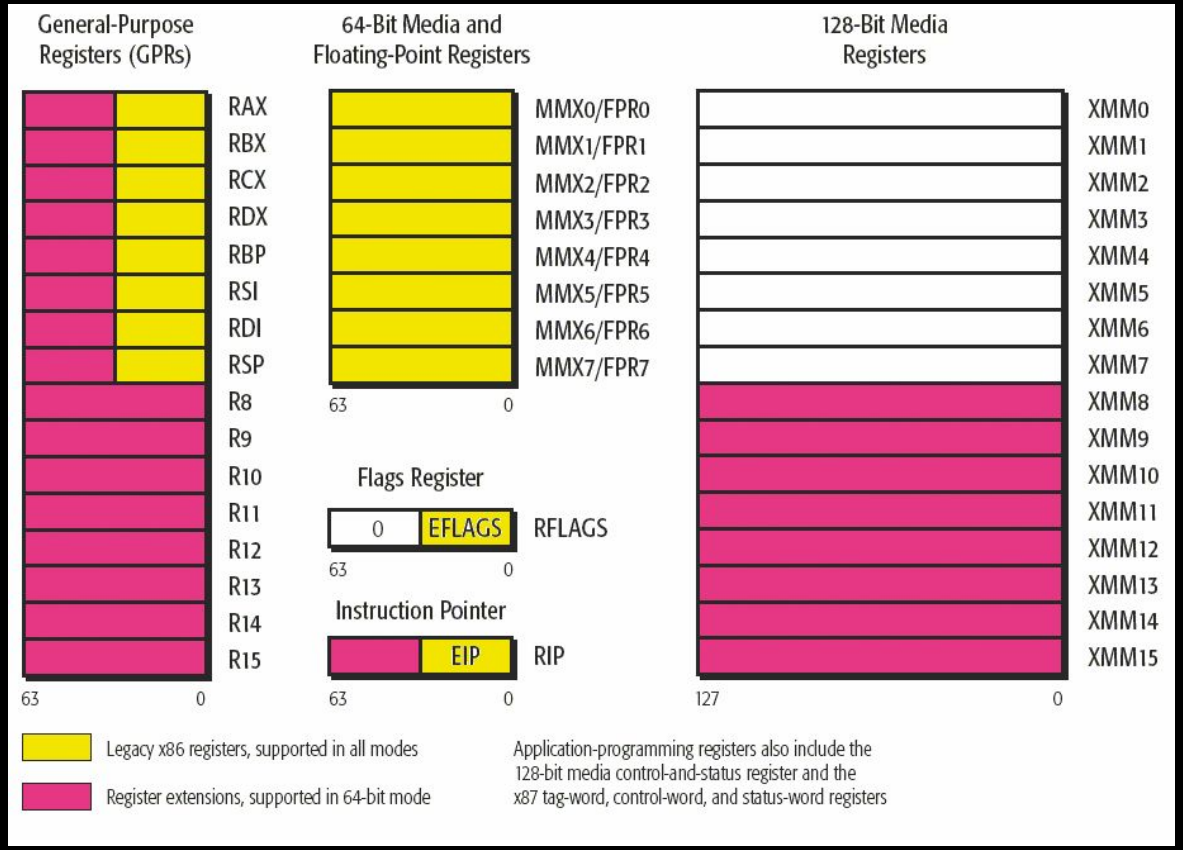
- Ядро в 0
- 1 и 2 на x86_64 не используются
- Юзерские приложения в 3
- Есть еще отрицательные, но не будем их трогать



Ассемблер

```
mov eax, 0
mov ecx, 1
xor edx, edx
sub esp, 8
push ebp
pop ebp
```

Регистры


General-Purpose Registers (GPRs)
RAX
RBX
RCX
RDX
RBP
RSI
RDI
RSP
R8
R9
R10
R11
R12
R13
R14
R15
63
0
64-Bit Media and Floating-Point Registers
MMX0/FPR0
MMX1/FPR1
MMX2/FPR2
MMX3/FPR3
MMX4/FPR4
MMX5/FPR5
MMX6/FPR6
MMX7/FPR7
63
0
Flags Register
0
EFLAGS
RFLAGS
63
0
Instruction Pointer
EIP
RIP
63
0
128-Bit Media Registers
XMM0
XMM1
XMM2
XMM3
XMM4
XMM5
XMM6
XMM7
XMM8
XMM9
XMM10
XMM11
XMM12
XMM13
XMM14
XMM15
127
0
Legacy x86 registers, supported in all modes
Register extensions, supported in 64-bit mode
Application-programming registers also include the 128-bit media control-and-status register and the x87 tag-word, control-word, and status-word registers

Ассемблер

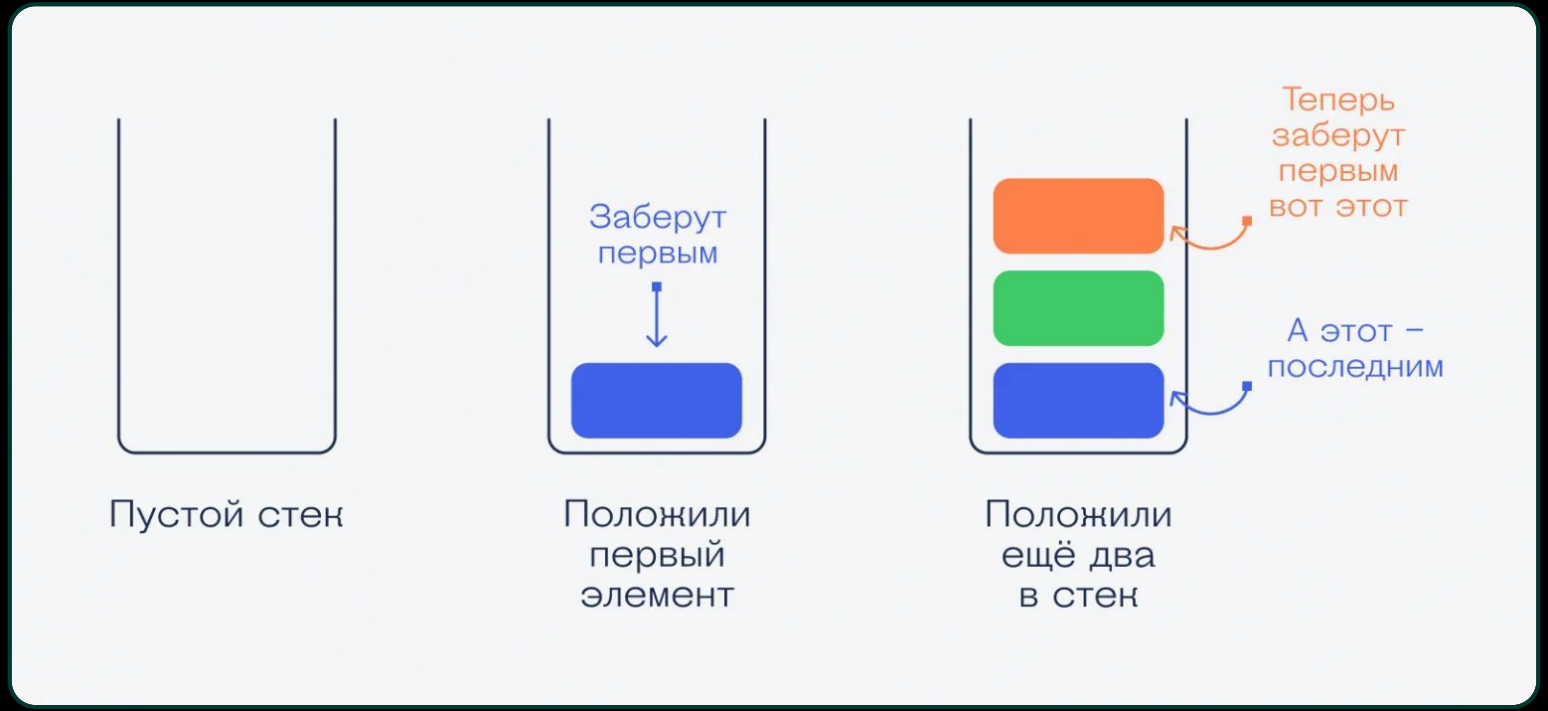
mov x, y — переместить значение из y в x

xor x, y — провести операцию XOR между y и x, результат положить в x

push x — положить на стек значение x

pop x — взять со стека верхнее значение и положить в x

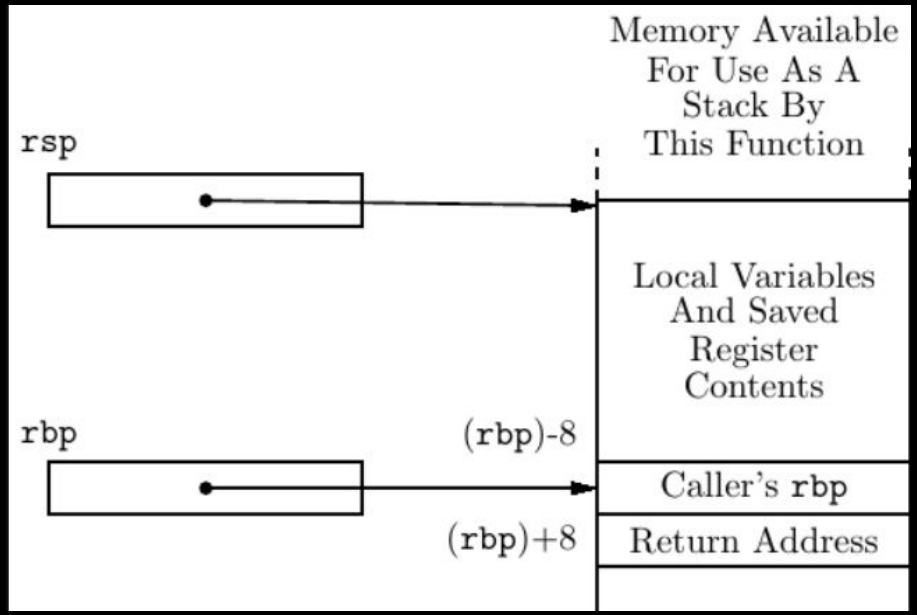
Стек


Заберут
первым
Теперь
заберут
первым
вот этот
А этот –
последним
Пустой стек
Положили
первый
элемент
Положили
ещё два
в стек

Стековый фрейм

- Самый первый элемент в стеке лежит по самому большому адресу
- Нам нужно хранить локальные переменные
- Давайте скажем, что есть регистр RBP, который указывает на самое нижнее место стека
- Тогда все, что выше, функция может использовать для хранения данных

Стековый фрейм



Соглашения о вызовах

- Очень много разных
- Зависят от платформы, разрядности, security ring
- Стандартный userspace Linux:

  RDI, RSI, RDX, RCX, R8, R9, stack

Square

C++ source #1

```cpp
int square(int num) {
    return num * num;
}
```

x86-64 gcc 14.2 (Editor #1)

x86-64 gcc 14.2

Compiler opt

```
square(int):
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-4], edi
        mov     eax, DWORD PTR [rbp-4]
        imul    eax, eax
        pop     rbp
        ret
```

Ещё стековый фрейм

C++ source #1

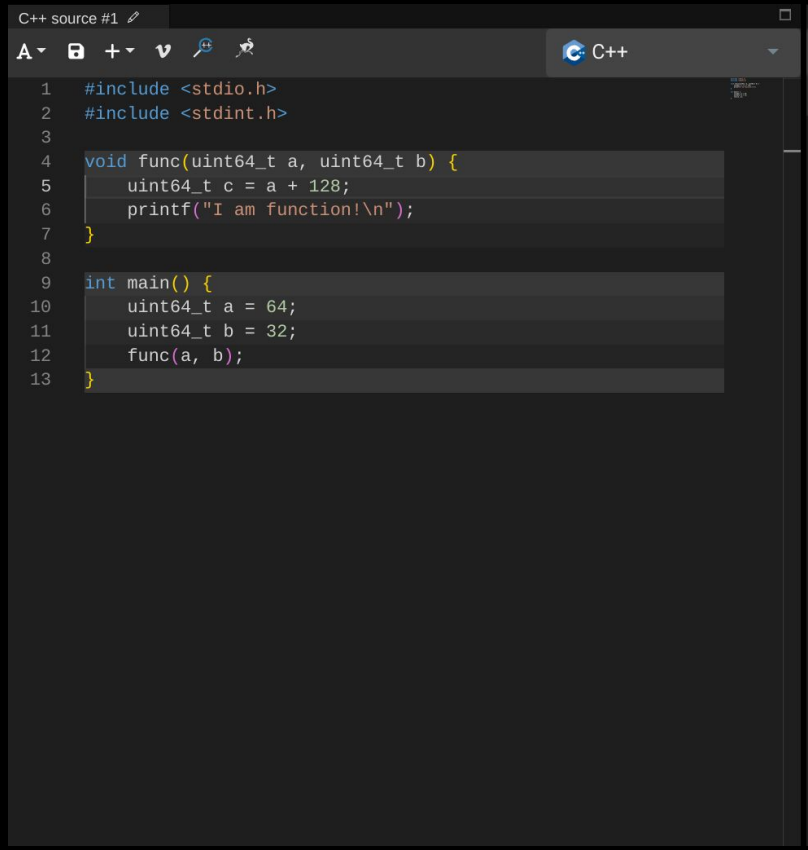
```
1  int square(int num) {
2      int a = 32;
3      int b = 64;
4      return 0;
5  }
```

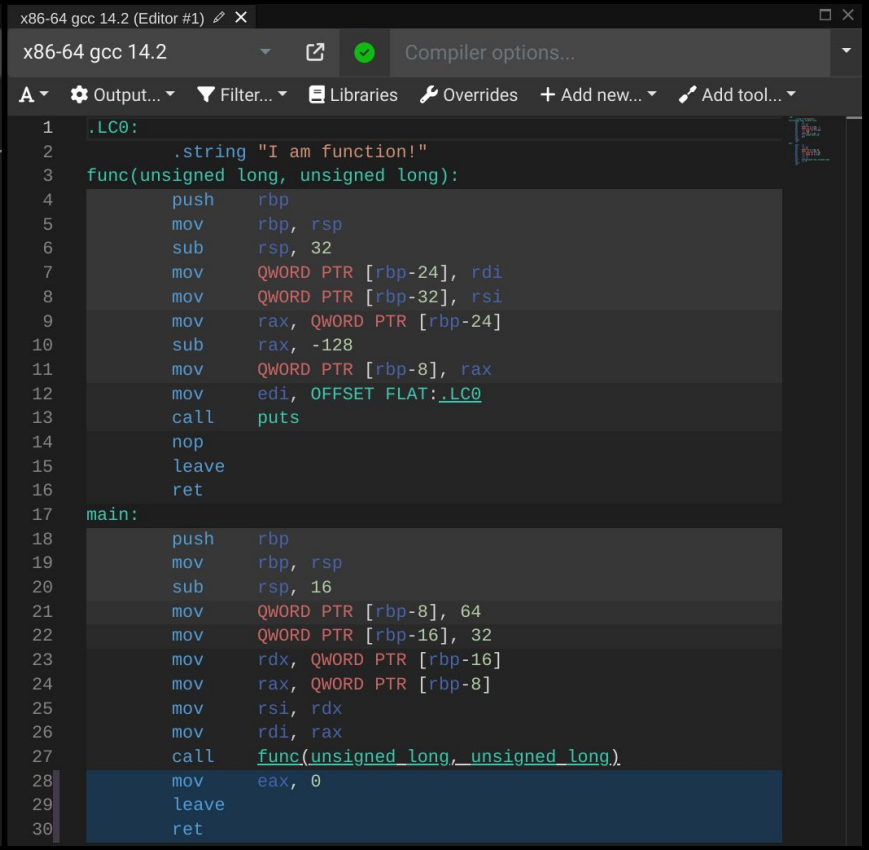
x86-64 gcc 14.2 (Editor #1)

x86-64 gcc 14.2

```
1  square(int):
2          push    rbp
3          mov     rbp, rsp
4          mov     DWORD PTR [rbp-20], edi
5          mov     DWORD PTR [rbp-4], 32
6          mov     DWORD PTR [rbp-8], 64
7          mov     eax, 0
8          pop     rbp
9          ret
```

Функции посложнее...





Что если мы почитаем...

C source #1

Save/Load + Add new... Vim C

```c
#include <unistd.h>

void func() {
    char buffer[32];
    read(0, buffer, 128);
}

int main() {
    int a = 64;
    func();
}
```

x86-64 gcc 14.2 (Editor #1)

x86-64 gcc 14.2 Compiler options...

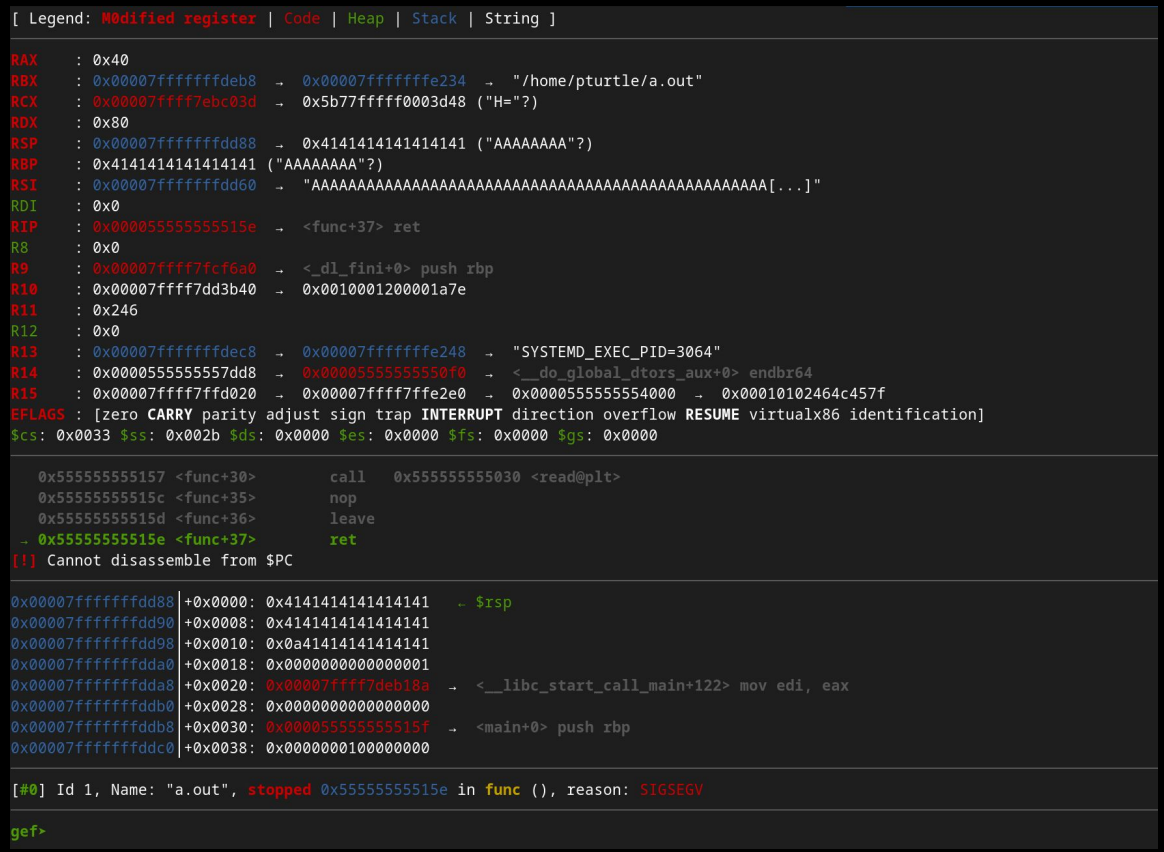
Output... Filter... Libraries Overrides + Add new... Add tool...

```asm
func:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        lea     rax, [rbp-32]
        mov     edx, 128
        mov     rsi, rax
        mov     edi, 0
        call    read
        nop
        leave
        ret
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], 64
        mov     eax, 0
        call    func
        mov     eax, 0
        leave
        ret
```

Стек:

0x30: buffer content (32bytes)
0x10: saved rbp
0x8 : return address
0x0 : 64

Stack buffer overflow



GDB

- r или run – запустить программу
- start – запустить и сразу остановиться
- ni – перейти на следующую инструкцию
- si – то же, что и ni, но если call, то зайти в функцию
- break *addr – остановиться на адресе
- x/gx addr – посмотреть 16bit-hex значенение по адресу
- cyclic – сгенерировать последовательность де Брёйна
- info functions – вывести все функции

DEMO

НИЯУ МИФИ

BI.ZONE

Что если нам не подыгрывают?

- Нам нужно как-то вызвать шелл
- Для этого нужна функция win
- Откуда её взять?...

Что если нам не подыгрывают?

- Нам нужно как-то вызвать шелл
- Для этого нужна функция win
- Откуда её взять?...
- libc!

libc

libc – основная системная библиотека в userspace linux для приложений на Си

Там огромное количество функций, которые мы можем вызывать с помощью ROP-цепочек

libc

libc – основная системная библиотека в userspace linux для приложений на Си

Там огромное количество функций, которые мы можем вызывать с помощью ROP-цепочек

Что такое ROP-цепочки?

ROP

ROP-цепочки – одна из мощнейших техник эксплуатации

Основная логика: из кусков программы сделать свою программу


```
pop rdi;    pop rsi;    call rax    add rsp, 8;
ret         ret                     ret
```

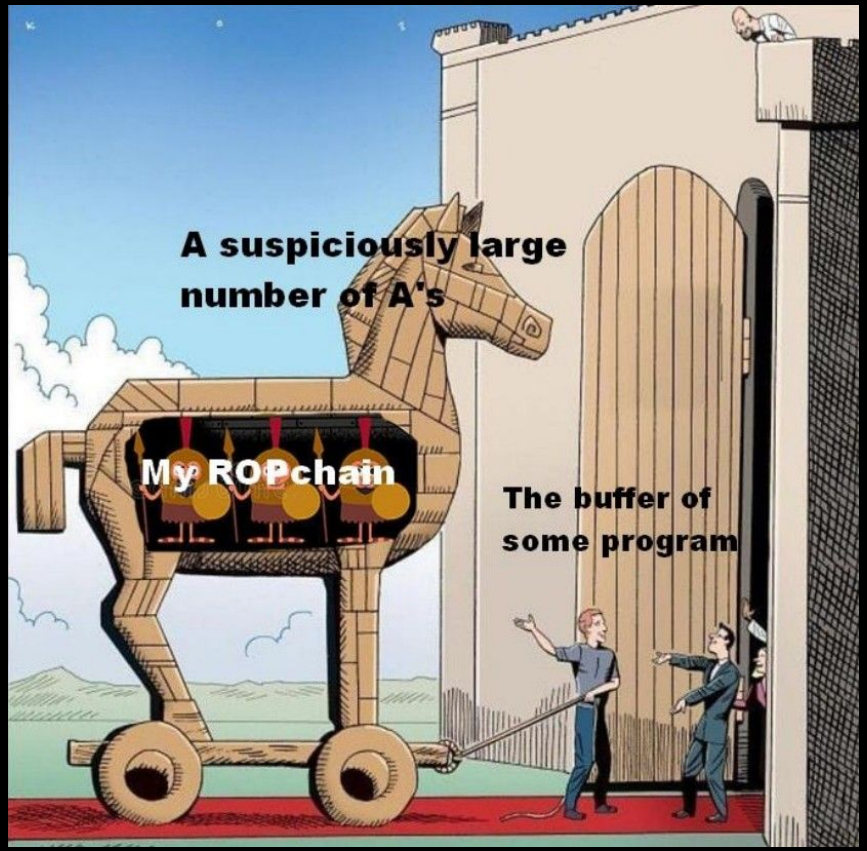
Митигации

Просто так ломать программы нам не дают. Есть защиты, которые сильно усложняют процесс пывна.

Самые нужные нам:

- ASLR (рандомизация базового адреса ELF библиотек)
- PIE (рандомизация базового адреса самого исполняемого файла)
- Full RELRO (read-only GOT)
- Stack Canary (секретное значение на стеке, которое мешает захватывать RIP)

DEMO


A suspiciously large number of A's
My ROPchain
The buffer of some program