Wait Wait …
Don‘t shell me!

2018, PlaidCTF
Theodor Mittermair

About

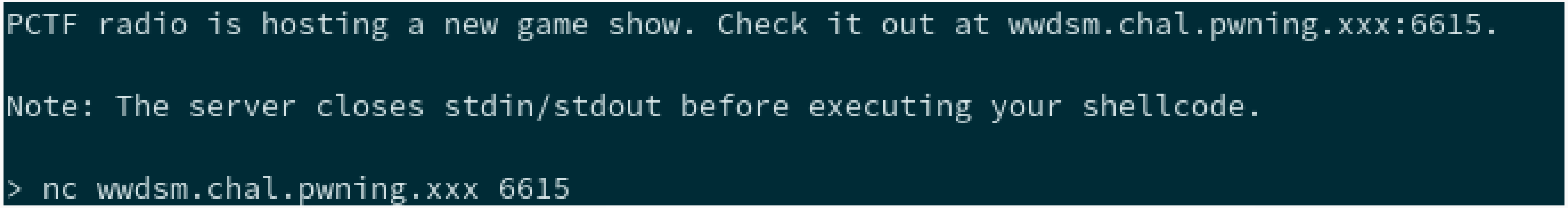
- Theodor Mittermair
  - Almost BsC Computer Engineering
  - Interested in hardware & low level stuff
- Not a complete success story,
  - but a learning experience.
- Questions during presentation are welcome!

Overview

- Challenge
- Attempt
  - Idea
  - Trial and Error
  - Realizations
- Solution
- Analysis

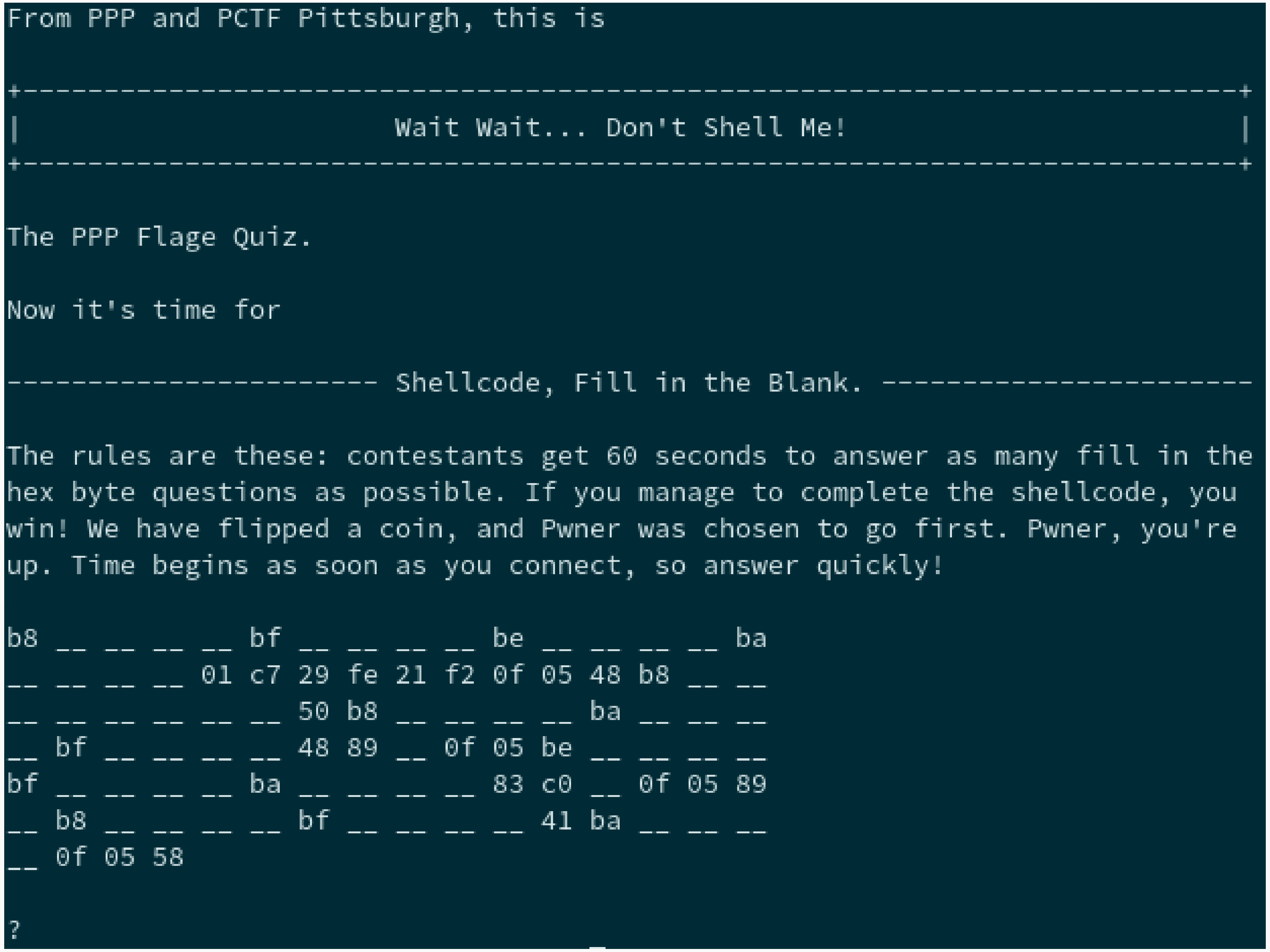
Challenge

- Wait Wait … Don‘t shell me! [1]
  - The hint was released only later



```
PCTF radio is hosting a new game show. Check it out at wwdsm.chal.pwning.xxx:6615.

Note: The server closes stdin/stdout before executing your shellcode.

> nc wwdsm.chal.pwning.xxx 6615
```

Challenge



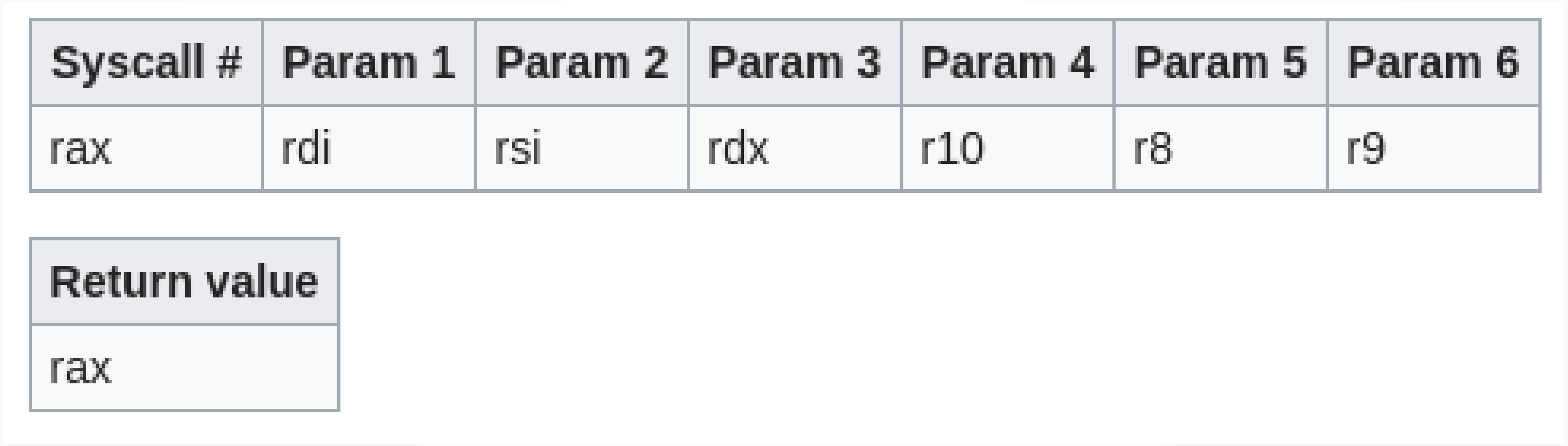
```
From PPP and PCTF Pittsburgh, this is

+------------------------------------------------------------------------------+
|                        Wait Wait... Don't Shell Me!                          |
+------------------------------------------------------------------------------+


The PPP Flage Quiz.

Now it's time for

----------------------- Shellcode, Fill in the Blank. -----------------------

The rules are these: contestants get 60 seconds to answer as many fill in the
hex byte questions as possible. If you manage to complete the shellcode, you
win! We have flipped a coin, and Pwner was chosen to go first. Pwner, you're
up. Time begins as soon as you connect, so answer quickly!

b8 __ __ __ __ bf __ __ __ __ be __ __ __ __ ba
__ __ __ __ 01 c7 29 fe 21 f2 0f 05 48 b8 __ __
__ __ __ __ __ __ 50 b8 __ __ __ __ ba __ __ __
__ bf __ __ __ __ 48 89 __ 0f 05 be __ __ __ __
bf __ __ __ __ ba __ __ __ __ 83 c0 __ 0f 05 89
__ b8 __ __ __ __ bf __ __ __ __ 41 ba __ __ __
__ 0f 05 58


?
```

Idea

- remote code execution
- analyze existing instructions
  - 4 syscalls (socket, connect, dup2, execve, ...)
  - we seem to control (at least some) arguments
  - variable length instructions?
- Later: no stdin/stdout → no shell
  - Flag on filedescriptor 3?

Calling Conventions

- Syscalls & Parameters [4]

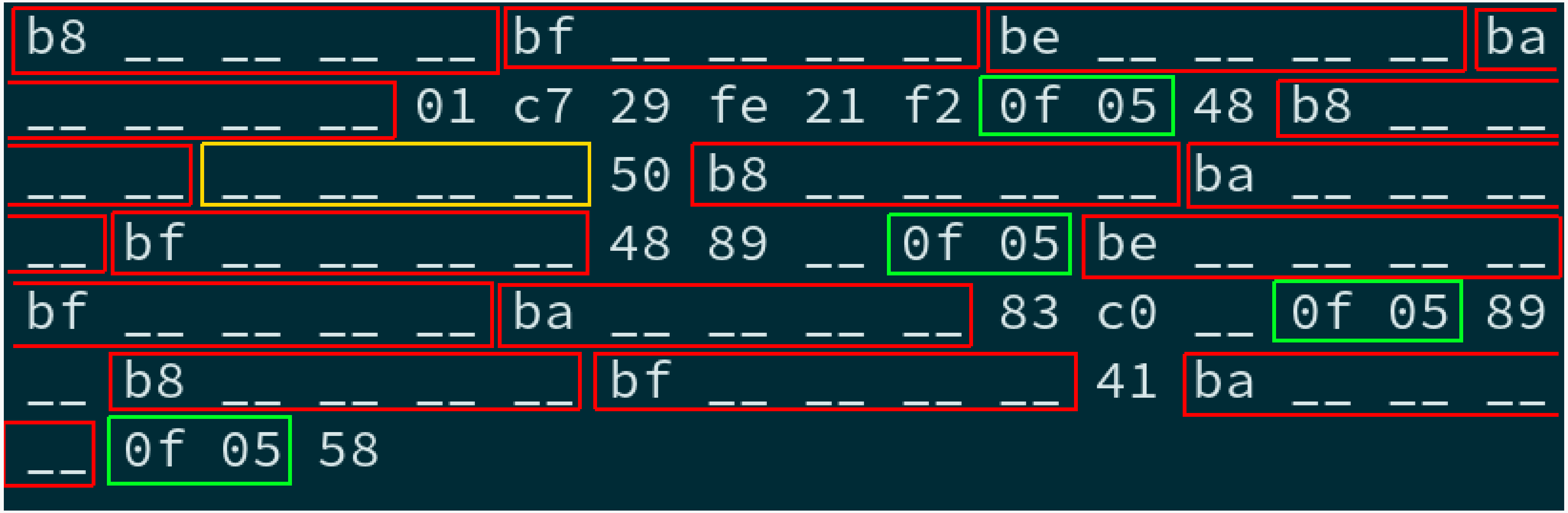


Syscall #	Param 1	Param 2	Param 3	Param 4	Param 5	Param 6
rax	rdi	rsi	rdx	r10	r8	r9

Return value
rax

main() { while(!solved()); }

- What do we have?

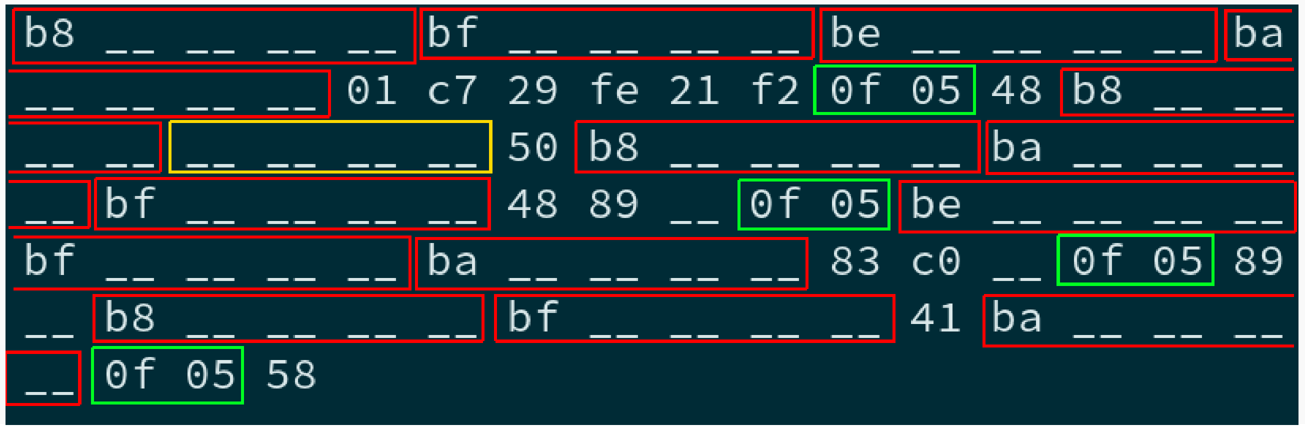


```
b8 __ __ __ __ bf __ __ __ __ be __ __ __ __ ba
__ __ __ __ 01 c7 29 fe 21 f2 0f 05 48 b8 __ __
__ __ __ __ __ __ 50 b8 __ __ __ __ ba __ __ __
__ bf __ __ __ __ 48 89 __ 0f 05 be __ __ __ __
bf __ __ __ __ ba __ __ __ __ 83 c0 __ 0f 05 89
__ b8 __ __ __ __ bf __ __ __ __ 41 ba __ __ __
__ 0f 05 58
```

rasm2

- rasm2 -a x86 -b 64 -d "b801234567“
  - mov eax, 0x67452301

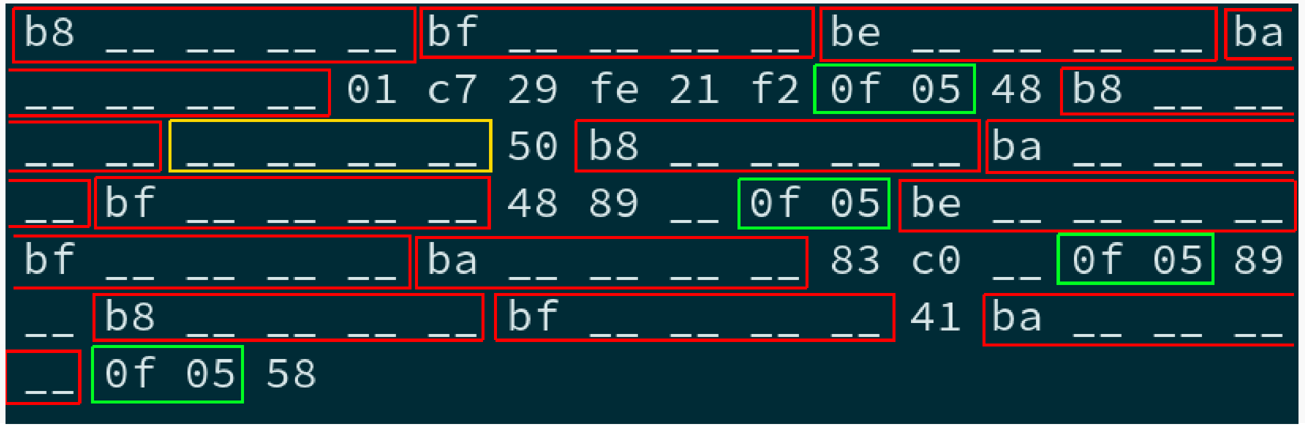
- rasm2 -a x86 -b 64 -D "01 c7 29 fe 21 f2 0f 05"
  - add edi, eax
  - add esi, edi
  - and edx, esi
  - syscall



```
b8 __ __ __ __ bf __ __ __ __ be __ __ __ __ ba
__ __ __ __ 01 c7 29 fe 21 f2 0f 05 48 b8 __ __
__ __ __ __ __ __ 50 b8 __ __ __ __ ba __ __ __
__ bf __ __ __ __ 48 89 __ 0f 05 be __ __ __ __
bf __ __ __ __ ba __ __ __ __ 83 c0 __ 0f 05 89
__ b8 __ __ __ __ bf __ __ __ __ 41 ba __ __ __
__ 0f 05 58
```

rasm2

- rasm2 -a x86 -b 64 -d
  - "83" → invalid
  - "83c0" → invalid
  - "83c000" → add eax, 0
  - "83c00000“ → add eax, 0; invalid

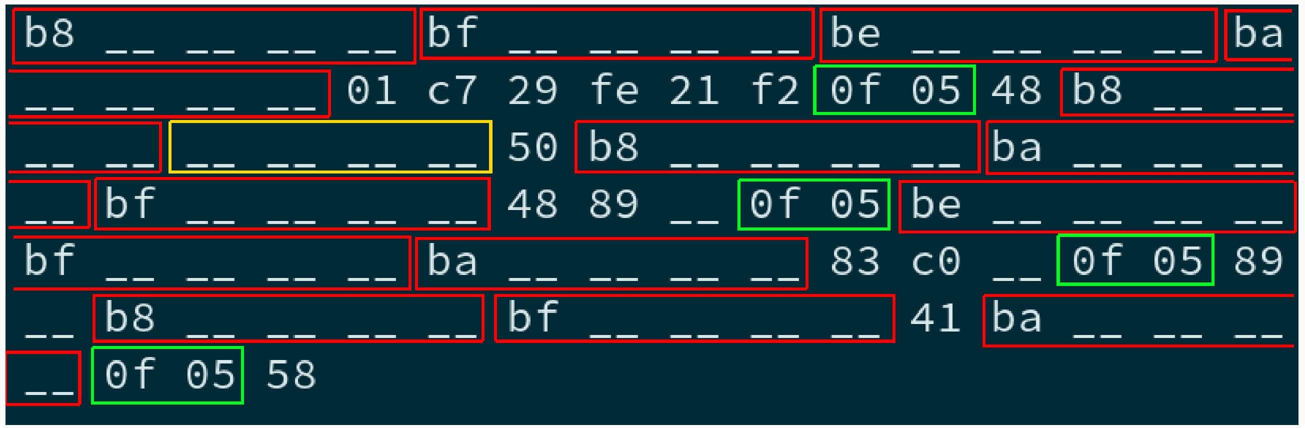


```
b8 __ __ __ __ bf __ __ __ __ be __ __ __ __ ba
__ __ __ __ 01 c7 29 fe 21 f2 0f 05 48 b8 __ __
__ __ __ __ __ __ 50 b8 __ __ __ __ ba __ __ __
__ bf __ __ __ __ 48 89 __ 0f 05 be __ __ __ __
bf __ __ __ __ ba __ __ __ __ 83 c0 __ 0f 05 89
__ b8 __ __ __ __ bf __ __ __ __ 41 ba __ __ __
__ 0f 05 58
```

rasm2

```
for i in {0..255}; do
    byte=$(printf "%02X" $i);
    echo "==== $byte";
    rasm2 -a x86 -b 64 -d "4889$byte";
done | less
```

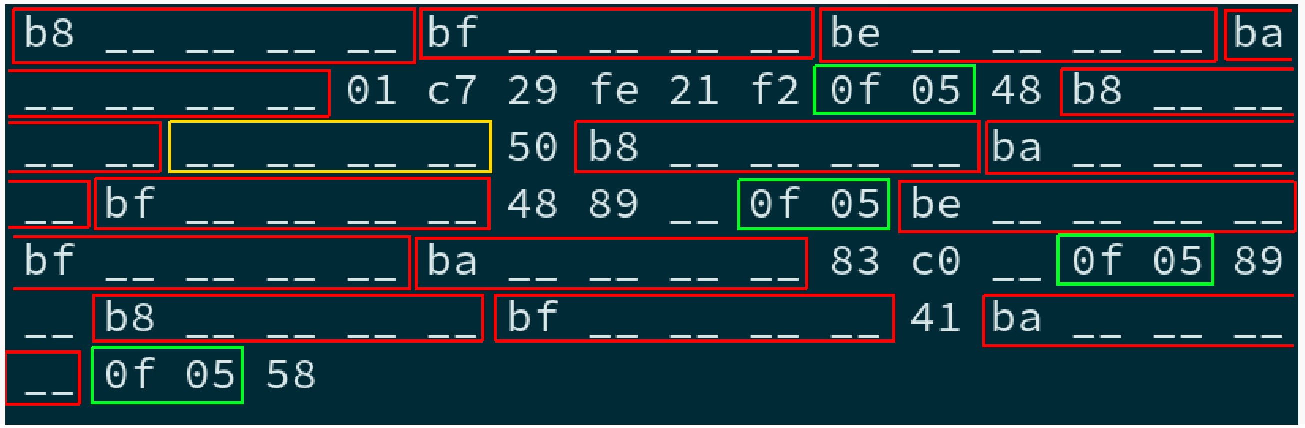
- → variants of
  - „mov [reg1], reg2“
  - „mov reg1, reg2“



```
b8 __ __ __ __ bf __ __ __ __ be __ __ __ __ ba
__ __ __ __ 01 c7 29 fe 21 f2 0f 05 48 b8 __ __
__ __ __ __ __ __ 50 b8 __ __ __ __ ba __ __ __
__ bf __ __ __ __ 48 89 __ 0f 05 be __ __ __ __
bf __ __ __ __ ba __ __ __ __ 83 c0 __ 0f 05 89
__ b8 __ __ __ __ bf __ __ __ __ 41 ba __ __ __
__ 0f 05 58
```

rasm2

- rasm2 -a x86 -b 64 -d "41"
  - Invalid ???
  - Part of previous or next Instruction?
- Got stuck.



```
b8 __ __ __ __ bf __ __ __ __ be __ __ __ __ ba
__ __ __ __ 01 c7 29 fe 21 f2 0f 05 48 b8 __ __
__ __ __ __ __ __ 50 b8 __ __ __ __ ba __ __ __
__ bf __ __ __ __ 48 89 __ 0f 05 be __ __ __ __
bf __ __ __ __ ba __ __ __ __ 83 c0 __ 0f 05 89
__ b8 __ __ __ __ bf __ __ __ __ 41 ba __ __ __
__ 0f 05 58
```

break;

- The beautiful pattern broke
- Complexity in our minds jumped

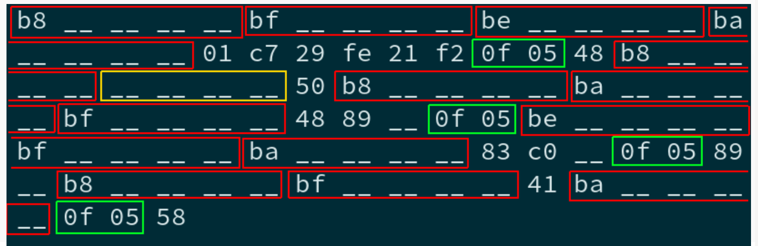
```
b8 __ __ __ __ bf __ __ __ __ be __ __ __ __ ba
__ __ __ __ 01 c7 29 fe 21 f2 0f 05 48 b8 __ __
__ __ __ __ __ __ 50 b8 __ __ __ __ ba __ __ __
__ bf __ __ __ __ 48 89 __ 0f 05 be __ __ __ __
bf __ __ __ __ ba __ __ __ __ 83 c0 __ 0f 05 89
__ b8 __ __ __ __ bf __ __ __ __ 41 ba __ __ __
__ 0f 05 58
```

Problems Encountered

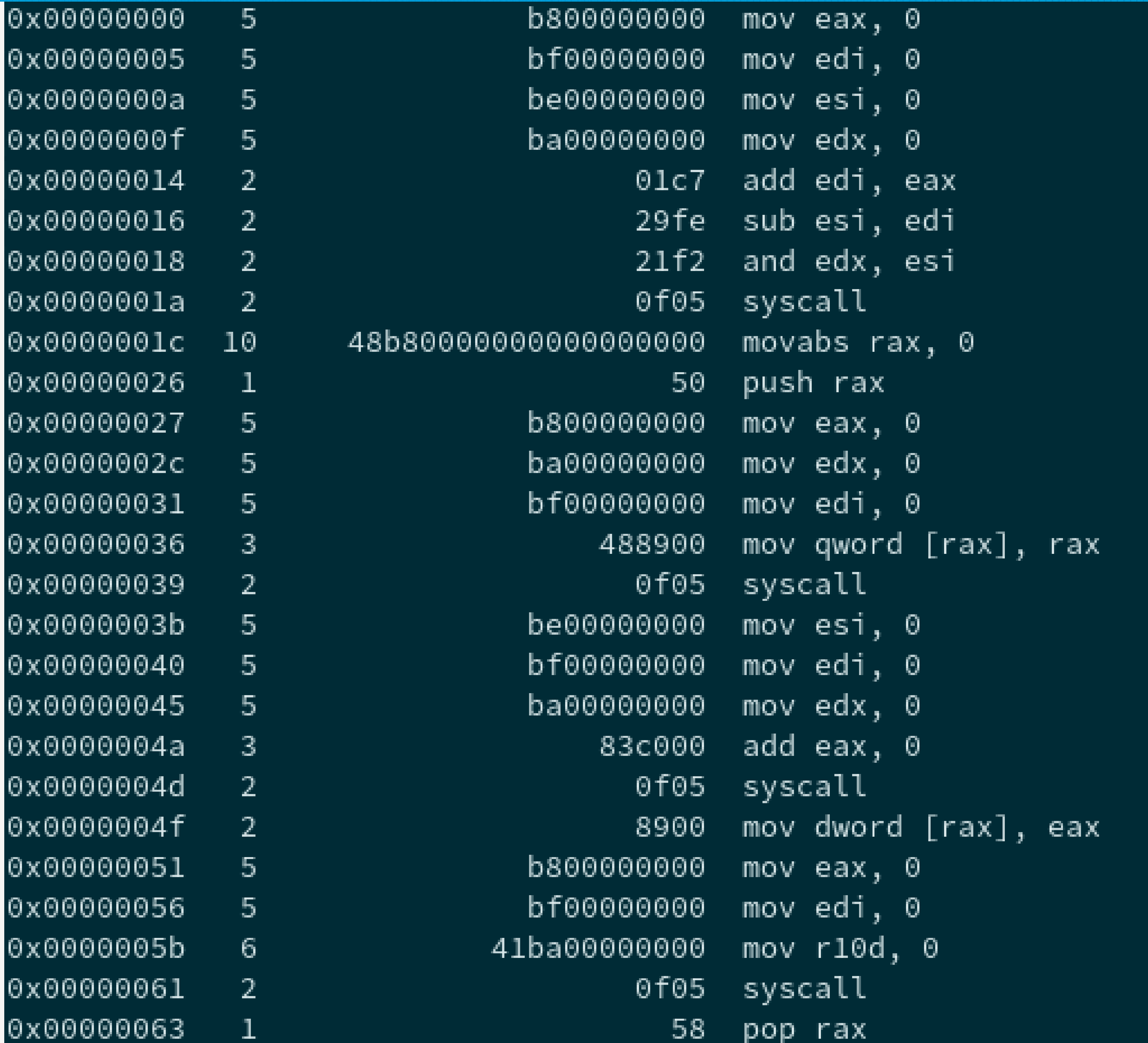
- Socket Struct: sa_length?
- architecture differences: x86 / amd64?
- variable length instructions?
- intermediate step necessary?

?

- rasm2



```
b8 __ __ __ __ bf __ __ __ __ be __ __ __ __ ba
__ __ __ __ 01 c7 29 fe 21 f2 0f 05 48 b8 __ __
__ __ __ __ __ __ 50 b8 __ __ __ __ ba __ __ __
__ bf __ __ __ __ 48 89 __ 0f 05 be __ __ __ __
bf __ __ __ __ ba __ __ __ __ 83 c0 __ 0f 05 89
__ b8 __ __ __ __ bf __ __ __ __ 41 ba __ __ __
__ 0f 05 58
```



```
0x00000000    5              b800000000  mov eax, 0
0x00000005    5              bf00000000  mov edi, 0
0x0000000a    5              be00000000  mov esi, 0
0x0000000f    5              ba00000000  mov edx, 0
0x00000014    2                    01c7  add edi, eax
0x00000016    2                    29fe  sub esi, edi
0x00000018    2                    21f2  and edx, esi
0x0000001a    2                    0f05  syscall
0x0000001c   10    48b80000000000000000  movabs rax, 0
0x00000026    1                      50  push rax
0x00000027    5              b800000000  mov eax, 0
0x0000002c    5              ba00000000  mov edx, 0
0x00000031    5              bf00000000  mov edi, 0
0x00000036    3                  488900  mov qword [rax], rax
0x00000039    2                    0f05  syscall
0x0000003b    5              be00000000  mov esi, 0
0x00000040    5              bf00000000  mov edi, 0
0x00000045    5              ba00000000  mov edx, 0
0x0000004a    3                  83c000  add eax, 0
0x0000004d    2                    0f05  syscall
0x0000004f    2                    8900  mov dword [rax], eax
0x00000051    5              b800000000  mov eax, 0
0x00000056    5              bf00000000  mov edi, 0
0x0000005b    6            41ba00000000  mov r10d, 0
0x00000061    2                    0f05  syscall
0x00000063    1                      58  pop rax
```

continue;

- Write-Up [2]
  - Two-Step Procedure
  - Get pointer to „flag.txt“:
    - Socket → Connect → Write
  - open and send flag file contents
    - Socket → Connect → Open → Sendfile

DEMO TIME

MAN, I SUCK AT THIS GAME. CAN YOU GIVE ME A FEW POINTERS?
0x3A28213A 0x6339392C, 0x7363682E.
I HATE YOU.

Analysis - Risk

- crafted & unrealistic, but ...
- ... not-so-distant real-world similarities ...
- -> unsanitized user input.
- -> access to currently unused resources.

Analysis - Mitigation

- Well … don‘t let others tell you assembly code you are going to execute :)
- → ASLR
- → Sanitize Input.
- → Limit access to minimum necessary.

Thank you for your attention!

References

- [1] https://ctftime.org/task/6070
- [2] https://fortenf.org/e/ctfs/pwn/2018/05/07/plaidctf-2018-waitwait.html
- [3] https://www.amd.com/system/files/TechDocs/24594.pdf
- [4] https://en.wikibooks.org/wiki/X86_Assembly/Interfacing_with_Linux#Syscalls

\0