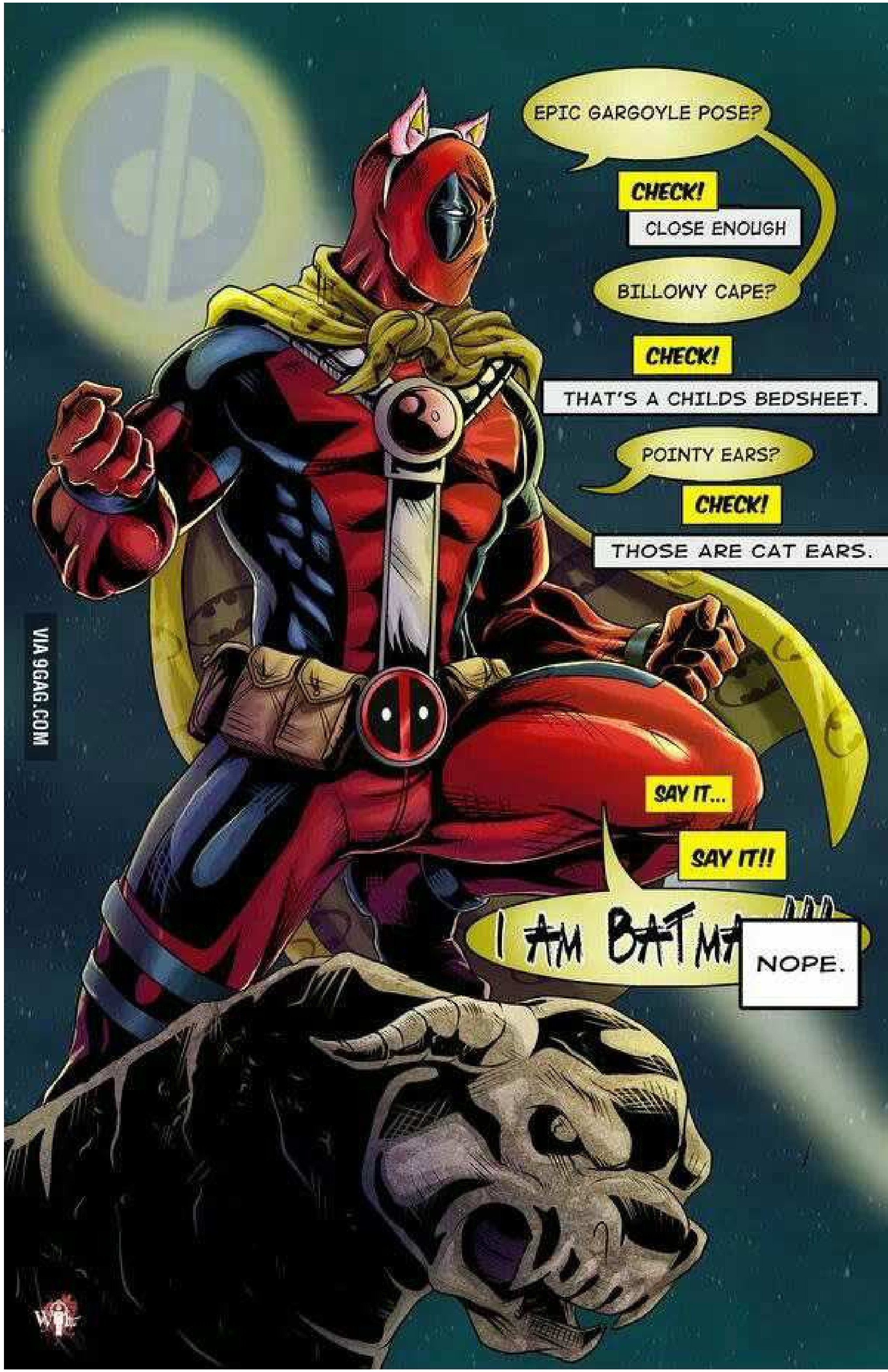
# Linux
# Exploit Mitigation

Dobin Rutishauser

V1.3, March 2016



# About me

At Compass Security since 2011

Spoke at OWASP Zürich, Bsides Vienna

On the internet:

- www.broken.ch, www.haking.ch, www.r00ted.ch, phishing.help
- @dobinrutis
- github.com/dobin

# About this presentation

To understand exploit mitigations

Need to understand exploit techniques

I'll lead you all the way, from zero

In 45 minutes!

# About this presentation

- Content of 8 hours for BFH
- It will get very technical
- Not possible to:
  - Cover all the topics
  - And be easy to understand
  - And handle all the details
- This should give more of an … overview
- Don’t worry if you don’t understand everything

# Overview of the presentation


Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

# Overview

1. Memory Layout
2. Stack
3. Exploit Basics
4. Exploit Mitigation
   - DEP
   - Stack Protector
   - ASLR
5. Contemporary Exploiting
6. Hardening
7. Container
8. Kernel

# Attacker wants:

- Execute his own code on the server
  - rm –rf /
  - Connect-back shellcode
  - echo “sysadmin:::” >> /etc/passwd

# Attacker needs:

- Be able to upload code to execute
- Be able to hijack instruction flow

# Memory Corruptions

## Buffer Overflow

- strcpy / strcat / memcpy
- Write past allocated buffer
- Minimum: 1 byte (off by one)

### Alternative: Arbitrary Write

- Can write certain memory area
- Not in scope in this presentation

# History intermezzo

Morris Worm in 1988, overflow in sendmail and finger

Around year 2000: Golden age of remote exploits

Team Teso (formatstring vulnerabilities)

- PHP, Apache, telnetd, wu-ftpd, qpopper, …

w00w00 (heap overflows)

- Efnet ircd, norton antivirus, AOL messenger, unixware stuff

Gobbles (putting the lulz in)

- Apache Scalp

ADM (high quality exploits)

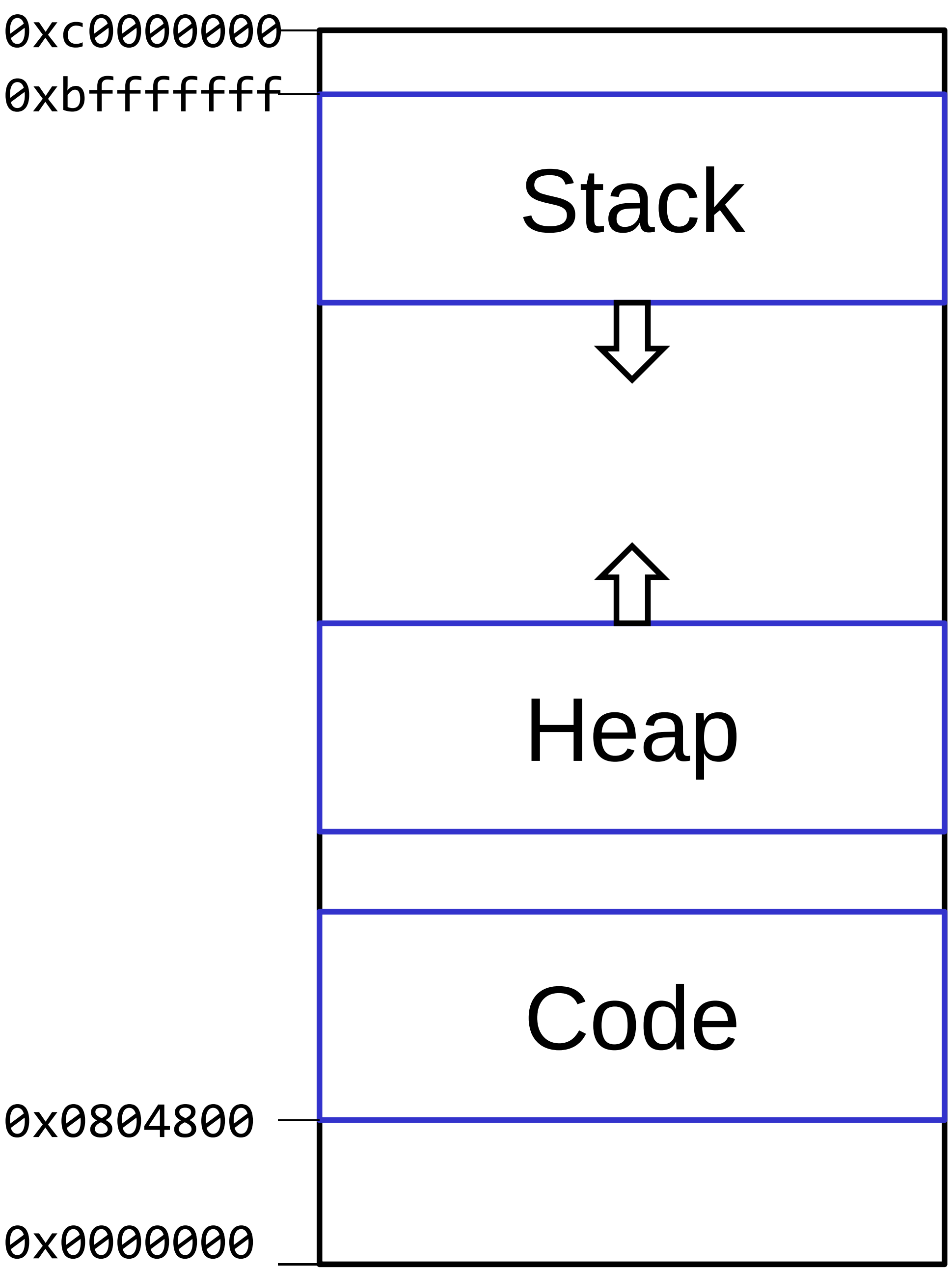
- Bind, wu-imapd, …

# Userspace Memory Layout

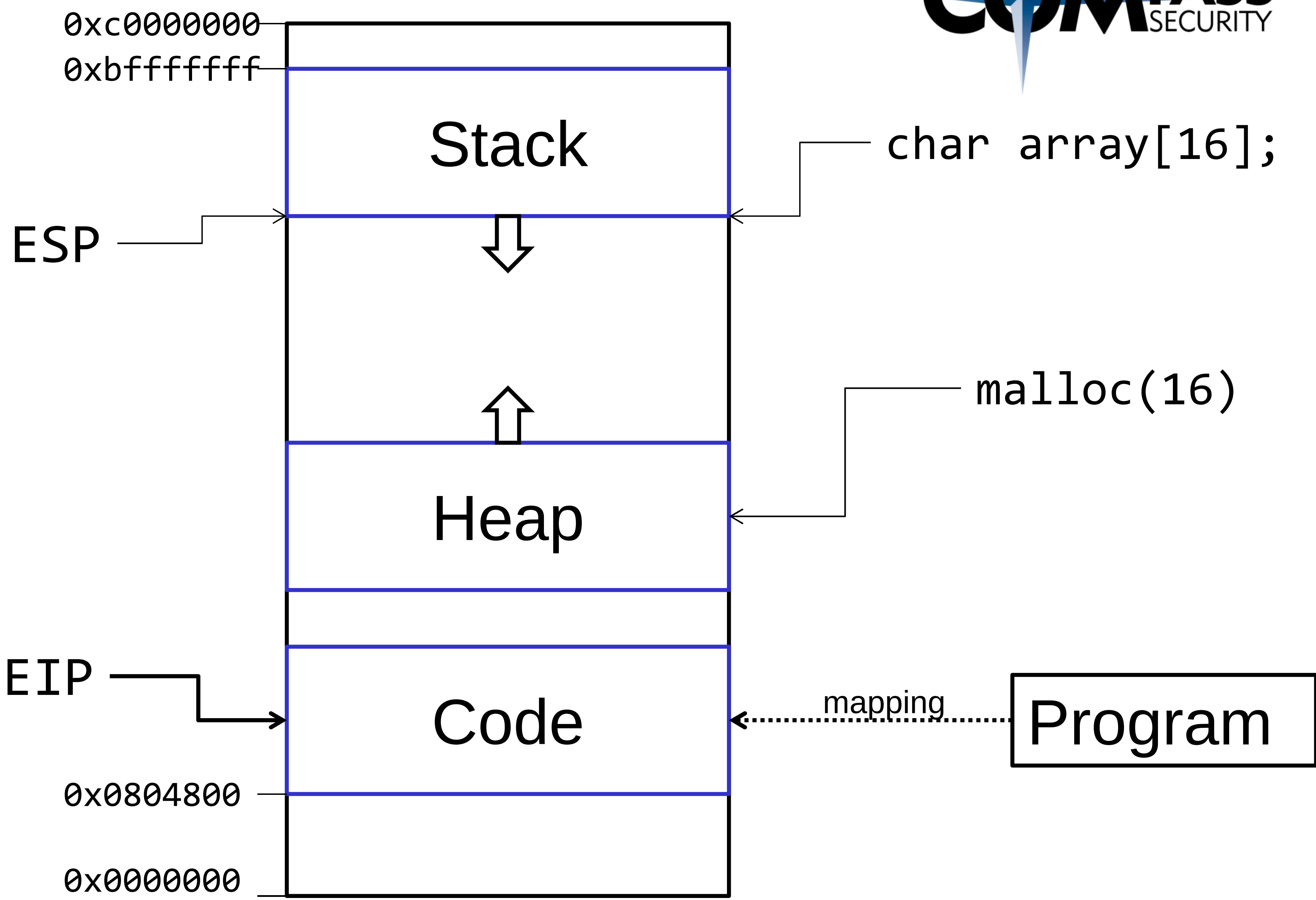
Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

# Process Memory Layout



# Process Memory Layout



# Userspace process data structures

## Stack

- There's one contiguous memory region containing the stack for the process
- LIFO – Last In, First Out
- Contains function local variables
- Also contains: Saved Instruction Pointer (SIP)
- Current function adds data to the top (bottom) of the stack

## Heap

- There's one contiguous memory region containing the heap
- Memory allocator returns specific pieces of the memory region
- For malloc()
- Also contains: heap management data

## Code

- Compiled program code

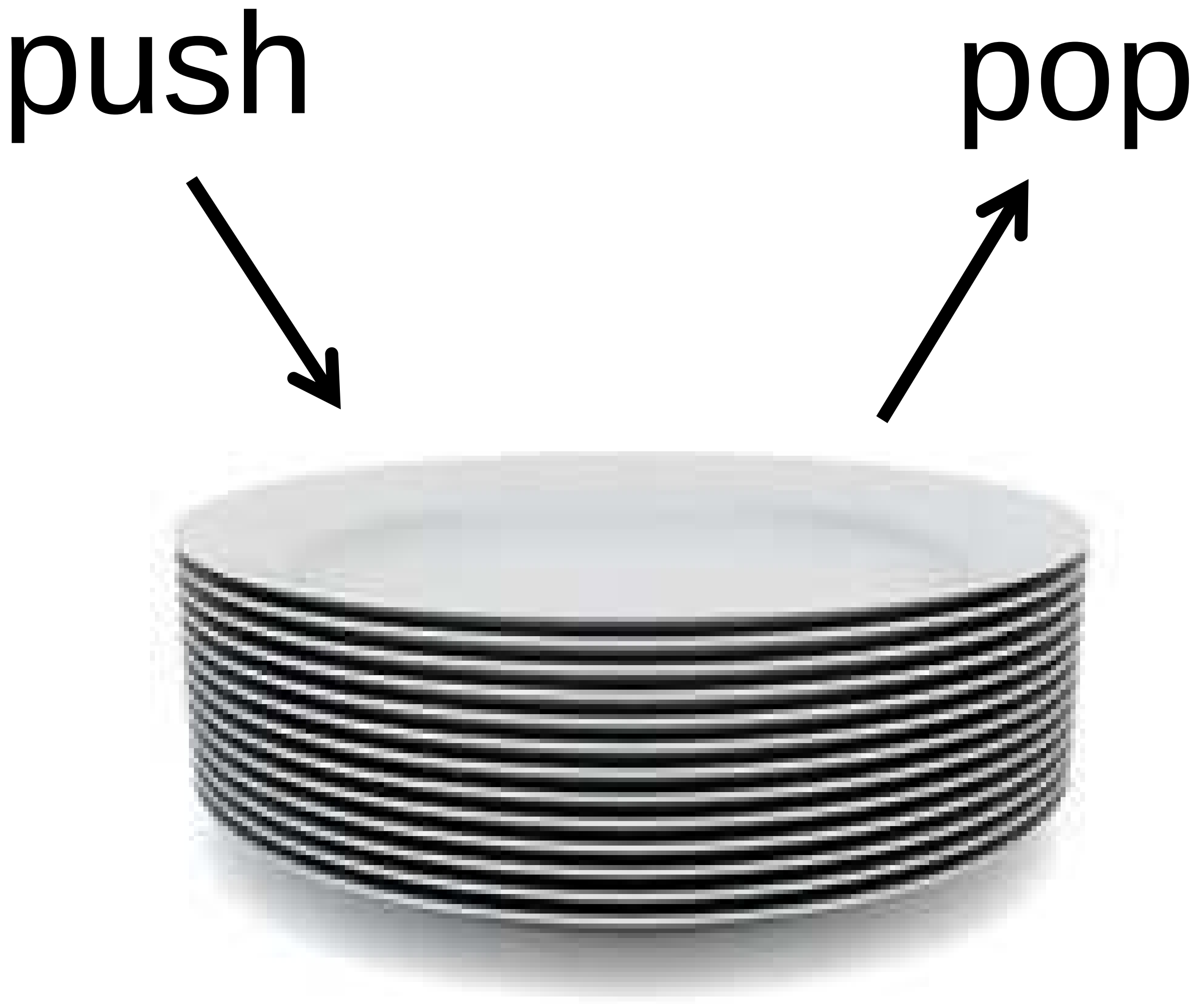
# Stacks

How do they work?

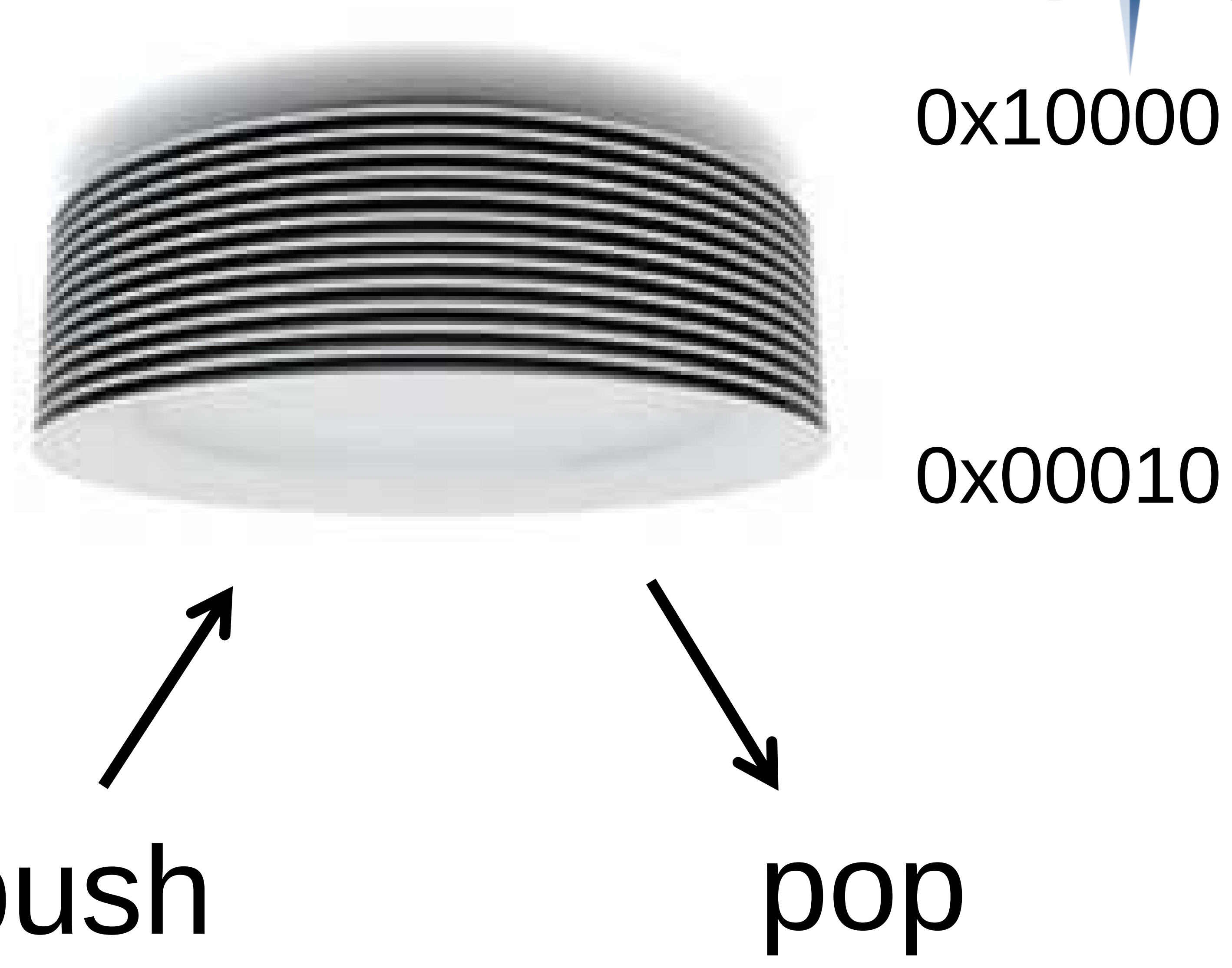
Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

# Stack



# Stack



```
void main(void) {
  int blubb = 0;
  foobar(blubb);
  return;
}

void foobar (int arg1) {
  char compass1[];
  char compass2[];
}
```

## Stack Layout

Saved IP (&__libc_start)

Saved Frame Pointer

Local Variables <main>

| |
|---|
| SIP |
| SFP |
| blubb |

Stack Frame <main>

Argument arg1 for <foobar>

**Saved IP (&return)**

Saved Frame Pointer

Local Variable 1

Local Variable 2

| |
|---|
| &blubb |
| **SIP** |
| SFP |
| **compass1** |
| **compass2** |

Stack Frame <foobar>

push pop

# Stack Layout

| | | |
|---|---|---|
| Saved IP (&__libc_start) | SIP | Stack Frame <main> |
| Saved Frame Pointer | SFP | |
| Local Variables <main> | blubb | |
| Argument arg1 for <foobar> | &blubb | Stack Frame <foobar> |
| **Saved IP (&return)** | **SIP** | |
| Saved Frame Pointer | SFP | |
| Local Variable 1 | **compass1** | |
| Local Variable 2 | **compass2** | |

push pop

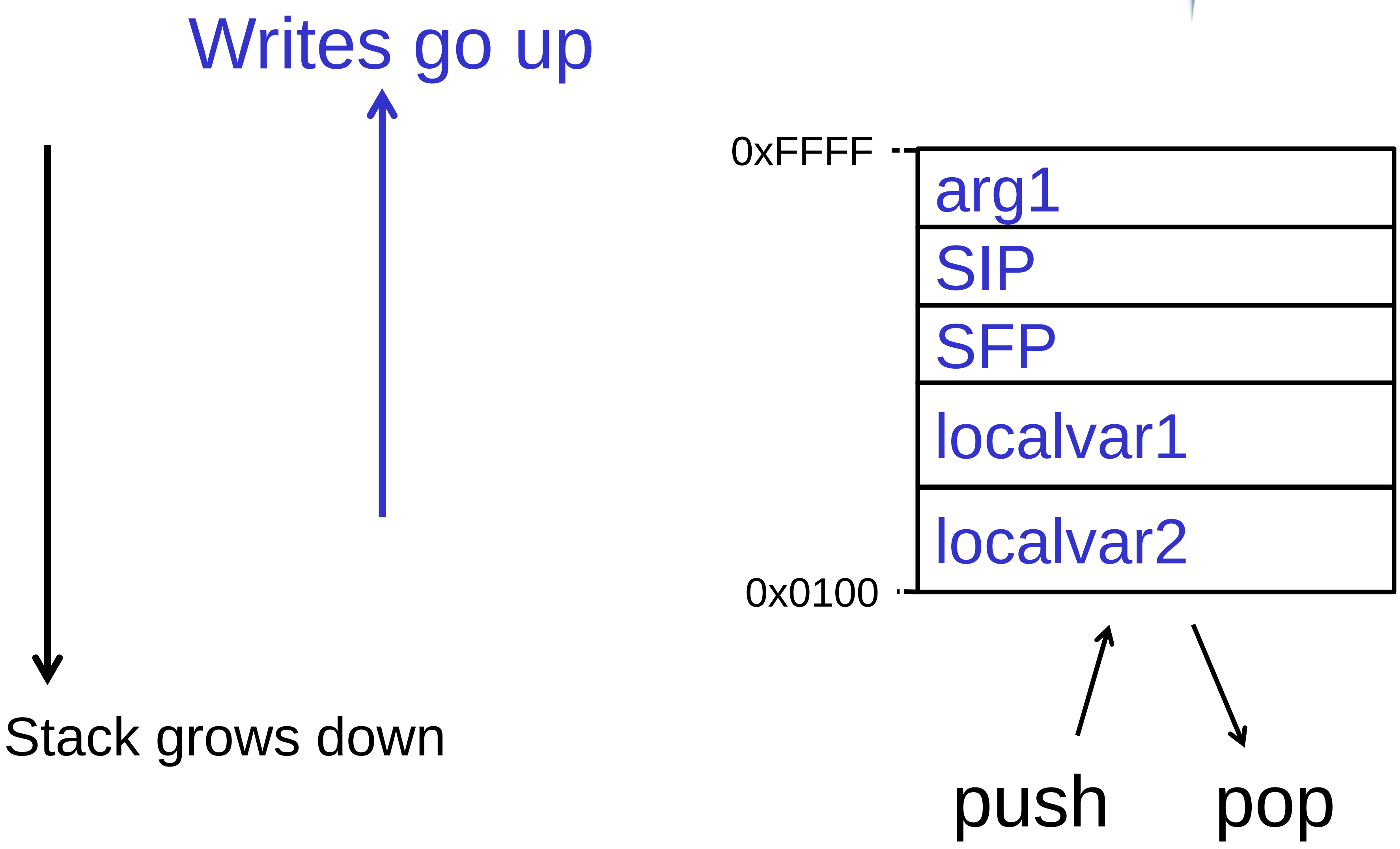
## Stack

```
void main(void) {
  int blubb = 0;
  foobar(blubb);
  return;
}

void foobar (int arg1) {
  char compass1[];
  char compass2[];
}
```

SIP

# SIP: Stored Instruction Pointer

- Copy of EIP
- Points to the address where control flow continues after end of function
  - (return, ret)
- Usually points into the code section


Writes go up
Stack grows down
0xFFFF
arg1
SIP
SFP
localvar1
localvar2
0x0100
push
pop

# Recap! Memory Layout

User data is on the stack

Also: important stuff is on the stack (Instruction Pointer, SIP)

Stack grows down ⇩

Writes go up ⇧

# Stack Overflow Exploitation

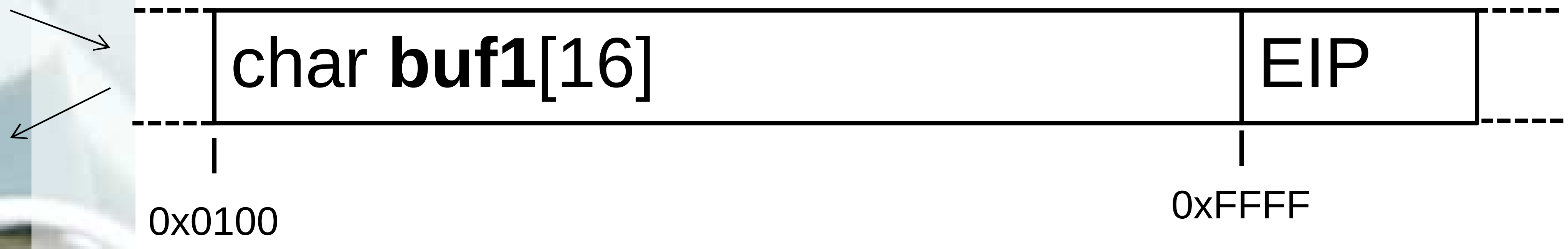
Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

# Exploitation Basics

- Program execution HIGHLY predictable/deterministic
  - Which is kind of surprising
- Stack, Heap, Code all start at the same address
- Same functions gets called in the same order
  - And allocate the same sized buffers
- "Error/Overflow in function X", every time:
  - Same call stack
  - Same variables
  - Same registers

# Buffer Overflow Basic Layout



| char **buf1**[16] | EIP |
|---|---|

strcpy(buf1, “AAAA AAAA AAAA AAAA”);

| AAAA AAAA AAAA AAAA | FF12 |
|---|---|

(0xFF12 = address of previous function)

Write up

# Buffer Overflow Basic Layout

| | |
|---|---|
| char **buf1**[16] | EIP |

strcpy(buf1, "AAAA AAAA AAAA AAAA BBBB");

| | |
|---|---|
| AAAA AAAA AAAA AAAA | BBBB |

Attacker can call any code he wants
But: What code?

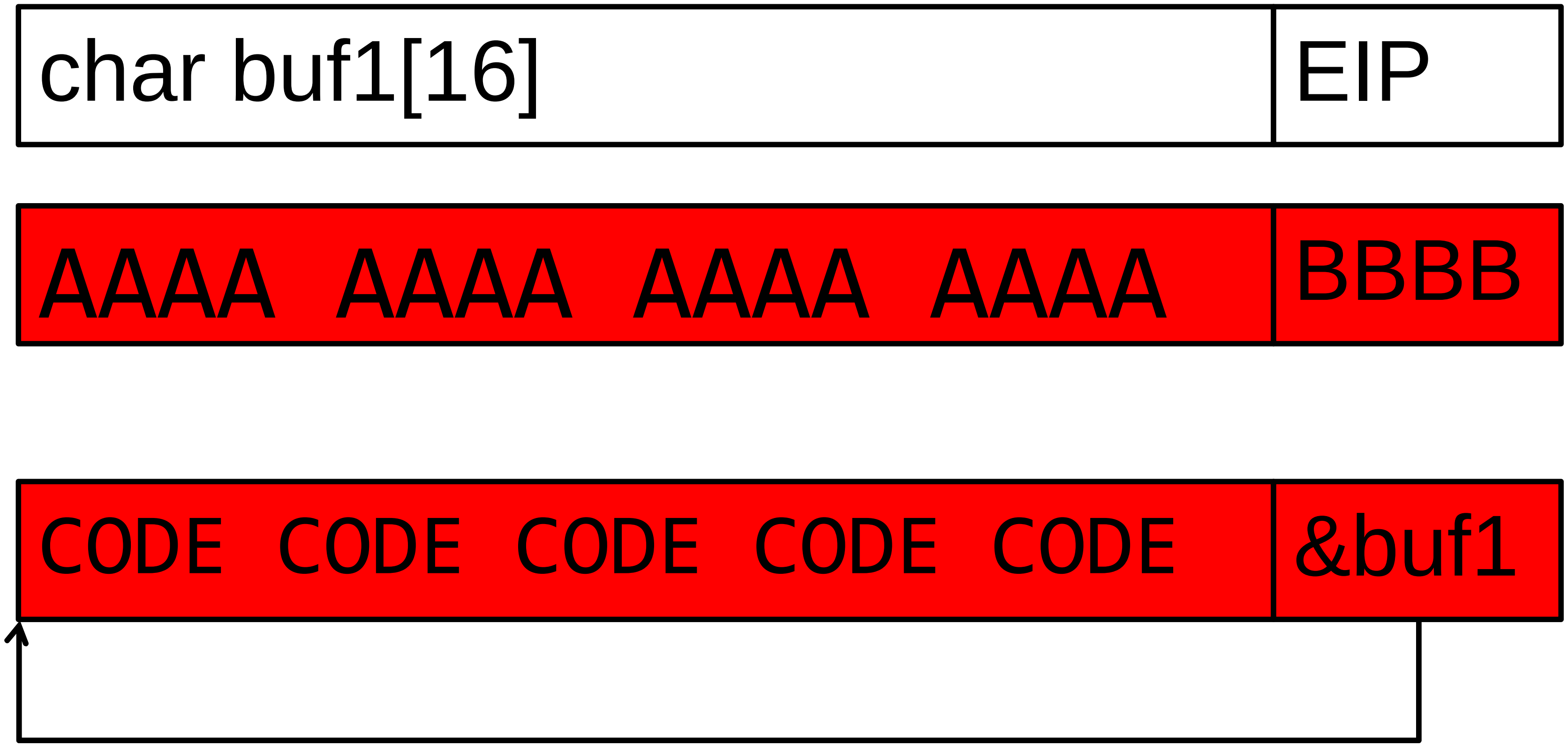
# Problem: In-band signaling

- Control data
- User data

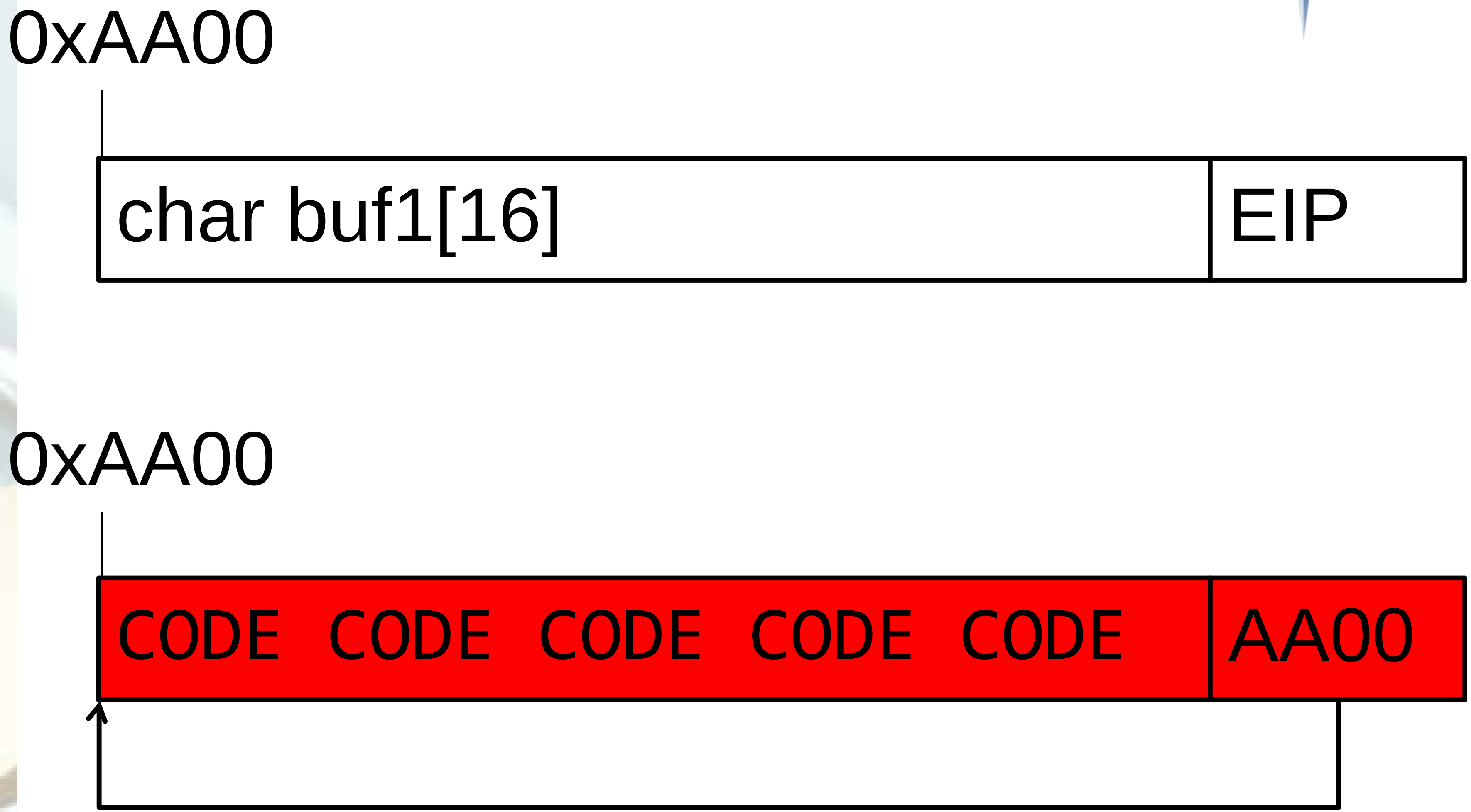
# Like old telephone networks

- 2600 hz: Indicate line is free
- With a 2600hz tone, you could phone anywhere, for free
- Oups, accidently created Legion of Doom

Return to Stack:




Jump to buffer with shellcode




Jump to buffer with shellcode

# How is shellcode formed?

Short description of shellcode

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

# Shellcode!

```
08048060 <_start>:
 8048060: 31 c0                 xor    %eax,%eax
 8048062: 50                    push   %eax
 8048063: 68 2f 2f 73 68        push   $0x68732f2f
 8048068: 68 2f 62 69 6e        push   $0x6e69622f
 804806d: 89 e3                 mov    %esp,%ebx
 804806f: 89 c1                 mov    %eax,%ecx
 8048071: 89 c2                 mov    %eax,%edx
 8048073: b0 0b                 mov    $0xb,%al
 8048075: cd 80                 int    $0x80
 8048077: 31 c0                 xor    %eax,%eax
 8048079: 40                    inc    %eax
 804807a: cd 80                 int    $0x80
```

```
char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                   "\x68\x68\x2f\x62\x69\x6e\x89"
                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                   "\xcd\x80\x31\xc0\x40\xcd\x80";
```

# Recap! Stack Overflow Exploit

Write past buffer on stack

Overwrite sIP

Point sIP to beginning of buffer

Place shellcode in buffer

Shellcode will be executed!

# Exploit Mitigations

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

# Exploit Mitigations

- DEP
- Stack Canary
- ASLR

# Exploit Mitigation: DEP

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

# DEP – Data Execution Prevention

- Aka: No-Exec Stack
- Aka: W^X (Write XOR eXecute)(OpenBSD)
- Aka: NX (Non-Execute) Bit

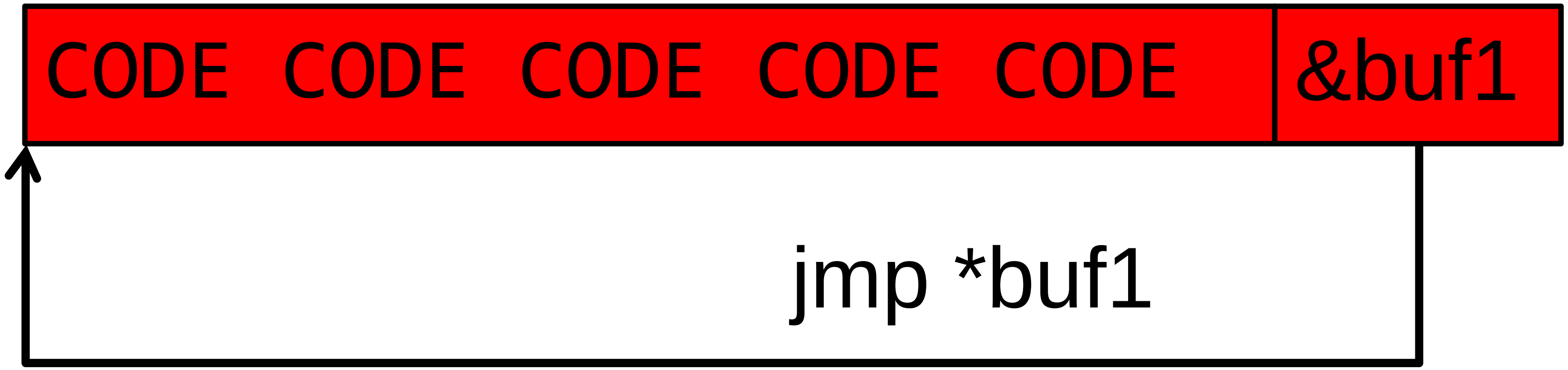
# AMD64 (x86-64) introduced NX bit in HW

- Intel 32 bit architecture (starting from 80386) “saved” Xecute bit
- For 32 bit, need PAE (Physical Address Extension, 32->36bit)
- Or kernel patches like PaX

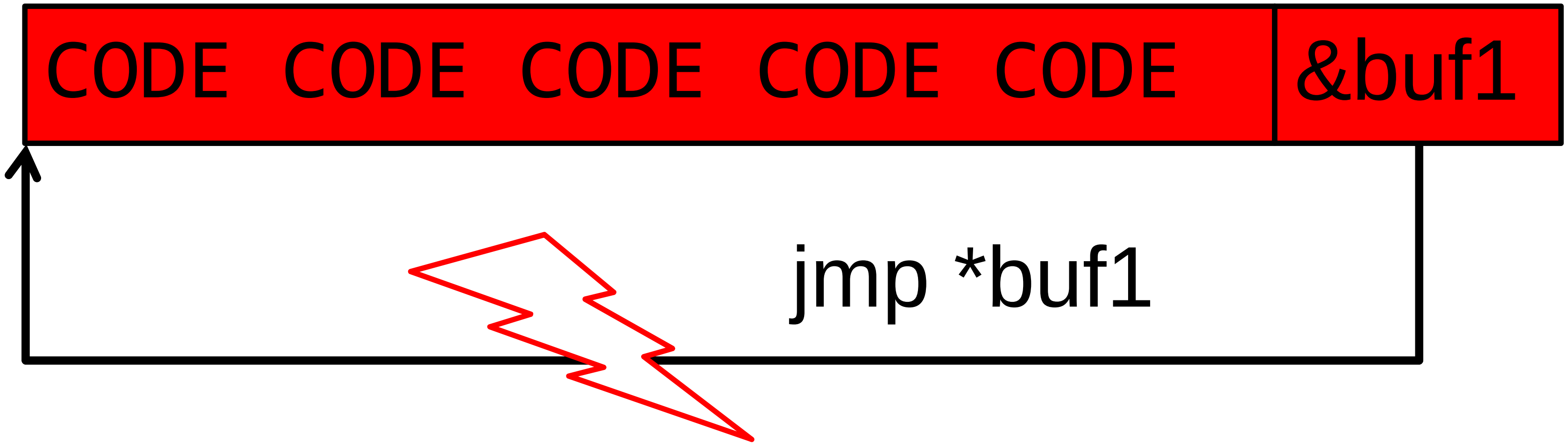
# Linux:

- Support in 2004, Kernel 2.6.8, default

Permissions: rwx



Permissions: rw-



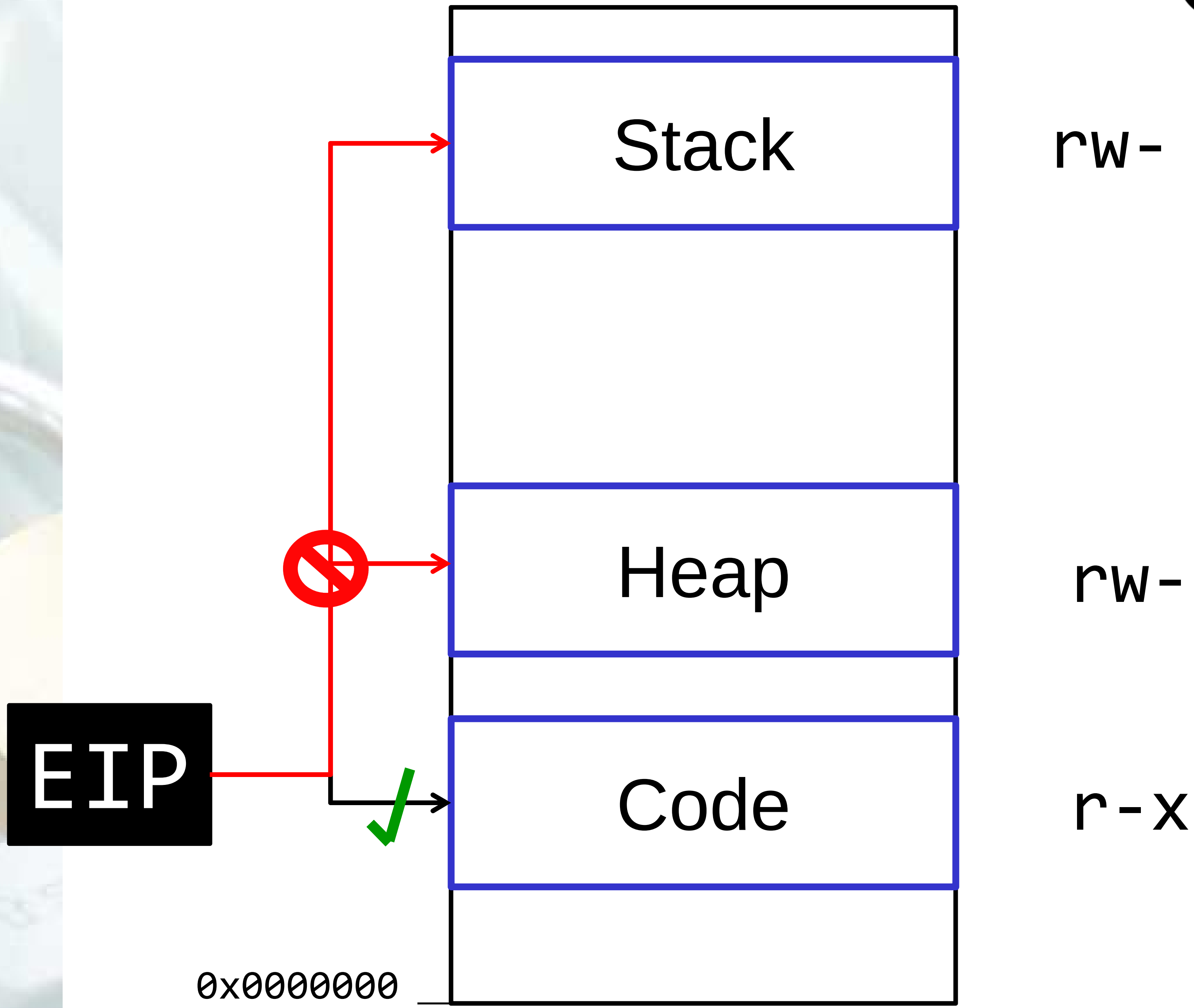

"Segmentation Fault"

# Read Only Stack

```
Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x00120 0x00120 R E 0x4
  INTERP         0x000154 0x08048154 0x08048154 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x005d0 0x005d0 R E 0x1000
  LOAD           0x000f14 0x08049f14 0x08049f14 0x00100 0x00108 RW  0x1000
  DYNAMIC        0x000f28 0x08049f28 0x08049f28 0x000c8 0x000c8 RW  0x4
  NOTE           0x000168 0x08048168 0x08048168 0x00044 0x00044 R   0x4
  GNU_EH_FRAME   0x0004d8 0x080484d8 0x080484d8 0x00034 0x00034 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x4
  GNU_RELRO      0x000f14 0x08049f14 0x08049f14 0x000ec 0x000ec R   0x1
```

```
Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x00120 0x00120 R E 0x4
  INTERP         0x000154 0x08048154 0x08048154 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x005d0 0x005d0 R E 0x1000
  LOAD           0x000f14 0x08049f14 0x08049f14 0x00100 0x00108 RW  0x1000
  DYNAMIC        0x000f28 0x08049f28 0x08049f28 0x000c8 0x000c8 RW  0x4
  NOTE           0x000168 0x08048168 0x08048168 0x00044 0x00044 R   0x4
  GNU_EH_FRAME   0x0004d8 0x080484d8 0x080484d8 0x00034 0x00034 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RWE 0x4
  GNU_RELRO      0x000f14 0x08049f14 0x08049f14 0x000ec 0x000ec R   0x1
```

# DEP: Memory Layout



# Read Only Stack

```
compass@ubuntu:~/bof$ cat shellcode.c
int main(void)
{
    char shellcode[] =
    "\x48\x31\xd2"                                  // xor    %rdx, %rdx
    "\x48\xbb\x2f\x2f\x62\x69\x6e\x2f\x73\x68"      // mov      $0x68732f6e69622f2f, %
rbx
    "\x48\xc1\xeb\x08"                              // shr    $0x8, %rbx
    "\x53"                                          // push   %rbx
    "\x48\x89\xe7"                                  // mov    %rsp, %rdi
    "\x50"                                          // push   %rax
    "\x57"                                          // push   %rdi
    "\x48\x89\xe6"                                  // mov    %rsp, %rsi
    "\xb0\x3b"                                      // mov    $0x3b, %al
    "\x0f\x05";                                     // syscall

    (*(void (*)()) shellcode)();

    return 0;
}
compass@ubuntu:~/bof$ gcc shellcode.c -o sh && ./sh
Segmentation fault (core dumped)
compass@ubuntu:~/bof$ gcc -z execstack shellcode.c -o sh-execstack && ./sh-execstack
$ id
uid=1000(compass) gid=1000(compass) groups=1000(compass),4(adm),24(cdrom),27(sudo),30(
dip),46(plugdev),108(lpadmin),124(sambashare)
$
```

# Recap! DEP

## Exploit Mitigation – DEP

- Makes it impossible for an attacker to execute his own shellcode
- Code: eXecute (no write)
- Heap, Stack: Write (no execute)

- No-no: Write and Execute
  - Sometimes necessary
  - Interpreted Languages
  - E.g. Java
  - Or JavaScript
  - Ähem *Browser* ähem

# Exploit Mitigation – Stack Protector

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

- Aka:
  - SSP: Stack Smashing Protector
  - Stack Cookie
  - Stack Canary
- Secret value in front of control data
- Generated per-process
  - Not per-function

# Exploit Mitigation – Stack Protector

| | |
|---|---|
| char buf1[16] | EIP |

# Exploit Mitigation – Stack Protector

| | |
|---|---|
| char buf1[16] | EIP |

| | | |
|---|---|---|
| char buf1[16] | secret | EIP |

# Exploit Mitigation – Stack Protector

| | | |
|---|---|---|
| char buf1[16] | secret | EIP |

| | | |
|---|---|---|
| char buf1[16] | 55667 | FF12 |

| | | |
|---|---|---|
| CODE CODE CODE CODE | BBBB | AA00 |

| | | |
|---|---|---|
| char buf1[16] | 55667 | FF12 |

| | | |
|---|---|---|
| CODE CODE CODE CODE | BBBB | AA00 |

"Segmentation Fault"

BBBB != 55667

# Stack Protector

- GCC patch
  - First: StackGuard in 1997
  - Then: ProPolice in 2001, by IBM
- Finally: Re-implement ProPolice in 2005 by RedHat
  - introduced in GCC 4.1
  - -fstack-protector
- Update:  Better implementation by Google in 2012
  - -fstack-protector-strong
- Enabled since like forever by default
  - most distributions
  - most packages

# Exploit Mitigation – Stack Protector

# Exploit Mitigation: ASLR

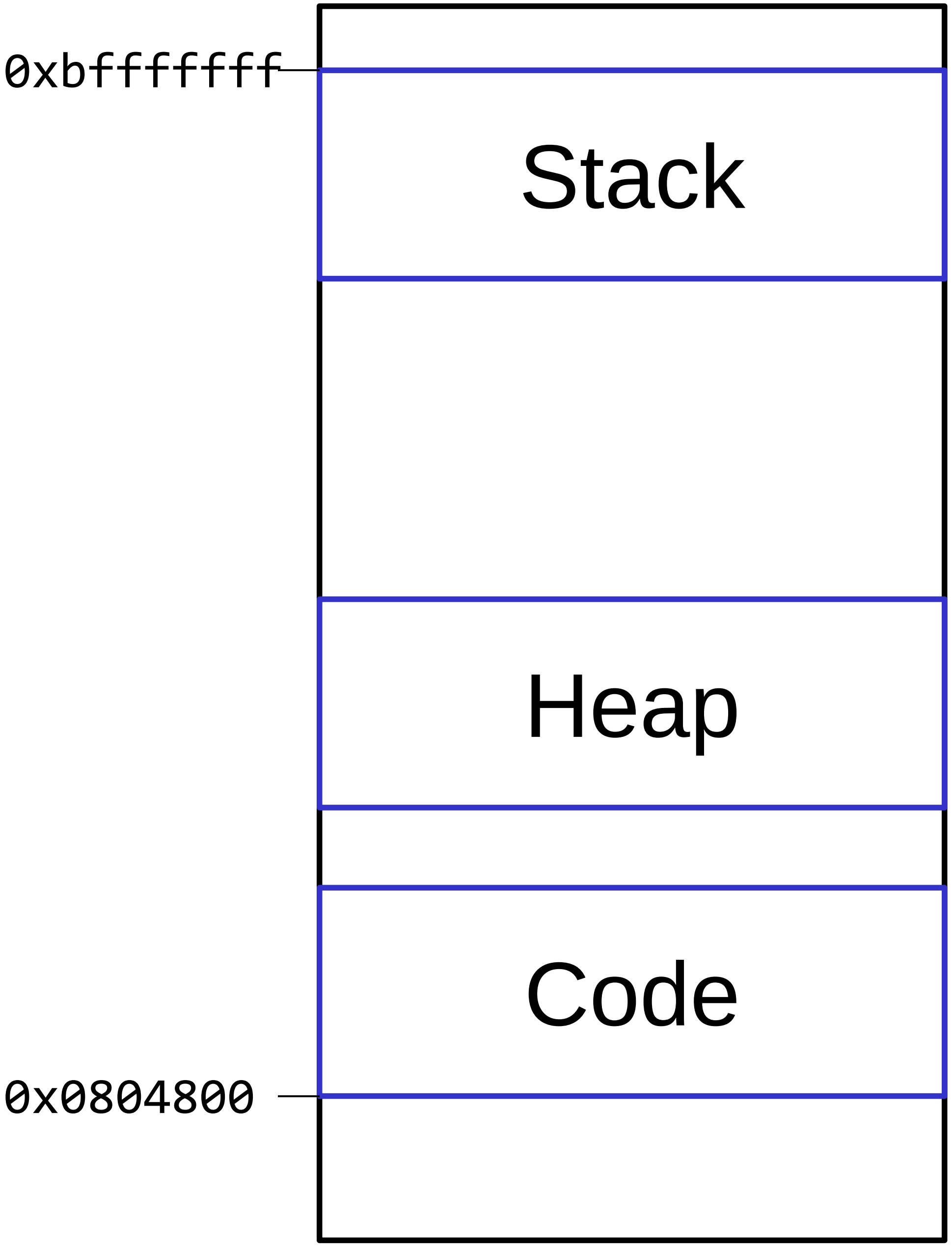

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

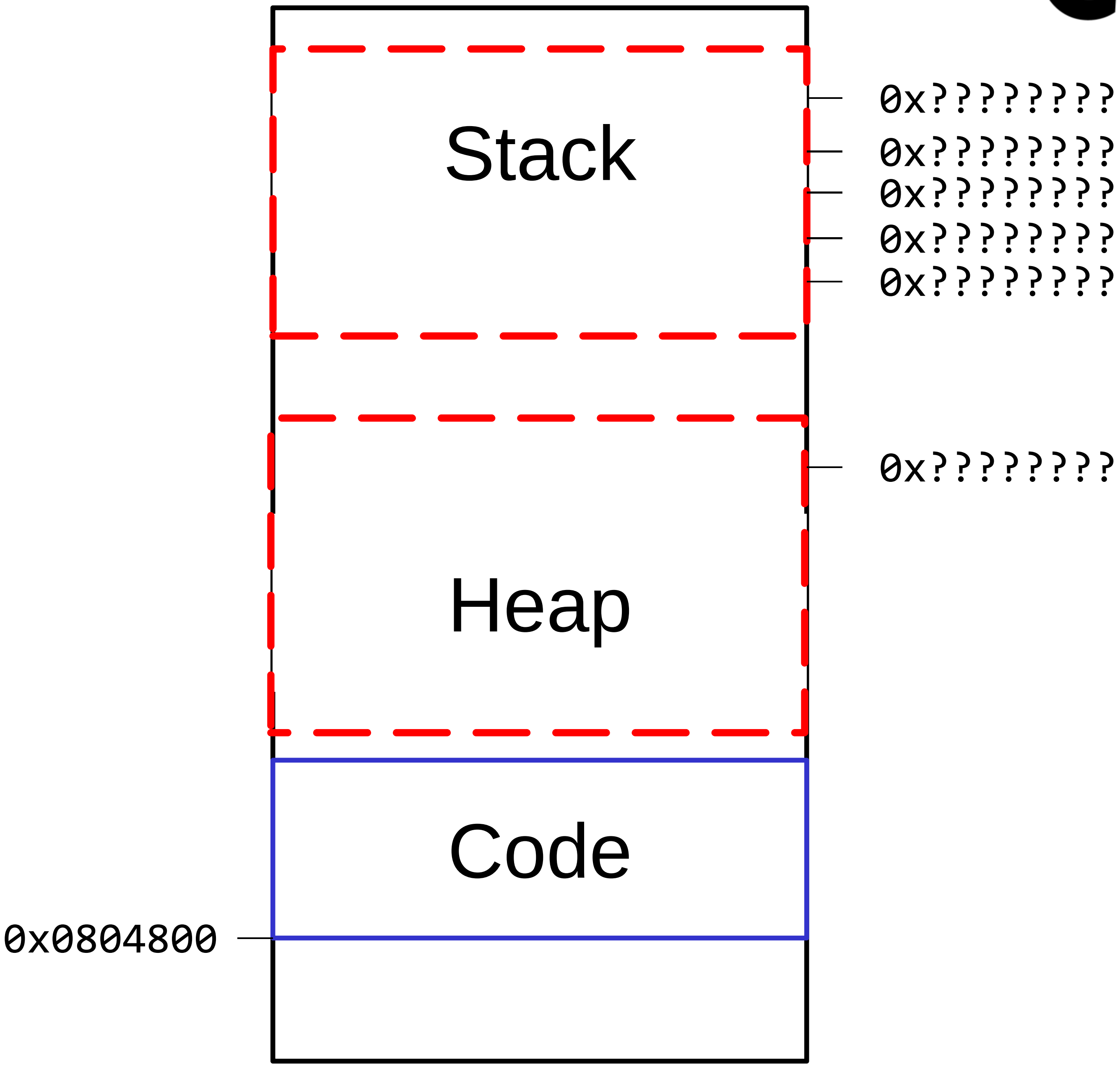
# Exploit Mitigation - ASLR

- Code execution is surprisingly deterministic
- E.g. Network service:
  1. fork()
  2. Parse incoming data
  3. Buffer Overflow is happening at module X line Y
- On every exploit attempt, memory layout looks the same!
  - Same stack/heap/code layout
  - Same address of the buffer(s)
- ASLR: Address Space Layout Randomization
  - Introduces randomness in memory regions

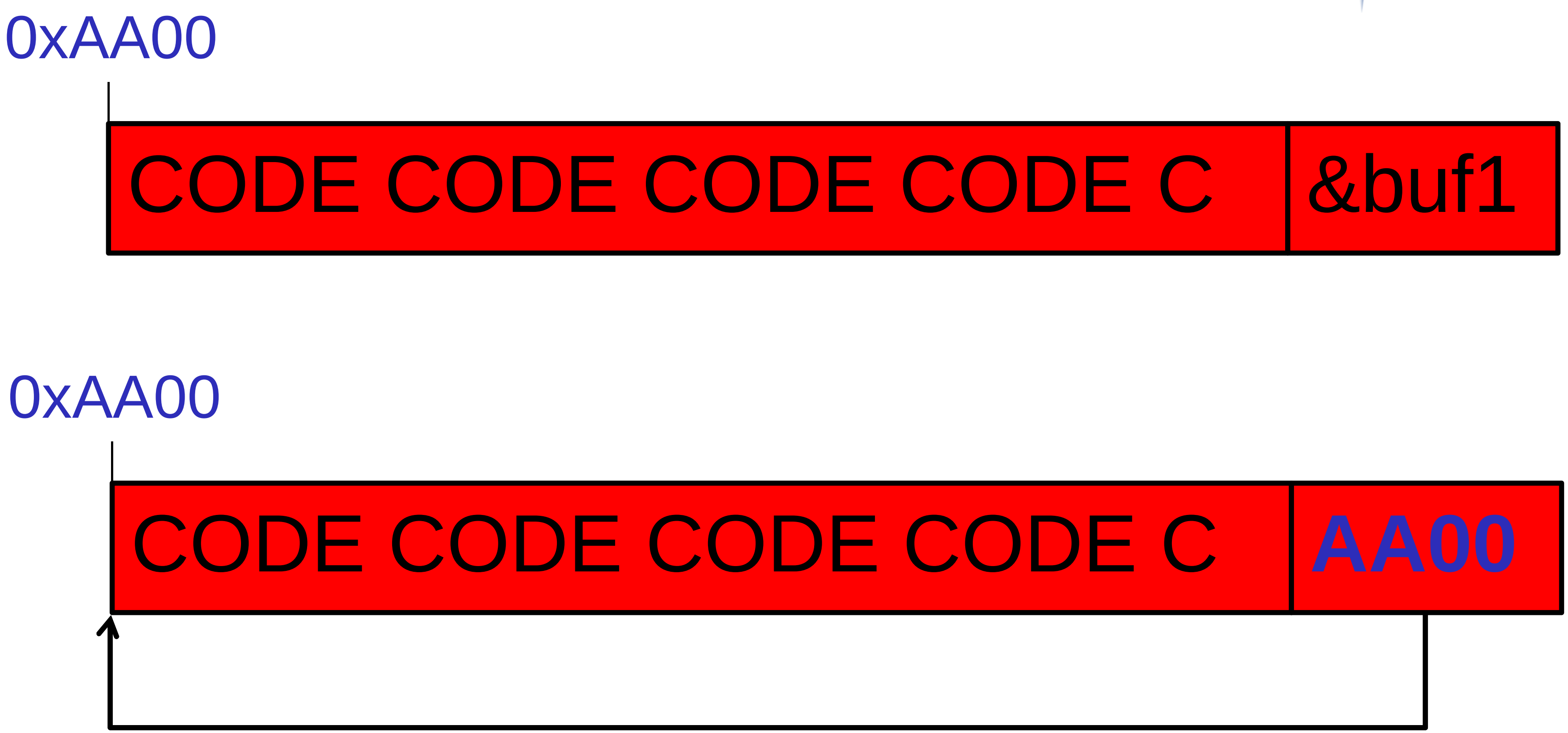
# Memory Layout



# Memory Layout



# Exploit Mitigation - ASLR



0xBB00

| CODE CODE CODE CODE C | &buf1 |
|---|---|

0xBB00

| CODE CODE CODE CODE C | AA00 |
|---|---|

"Segmentation Fault"

AA00 != **BB00**

# Randomness is measured in entropy

- Several restrictions
  - Pages have to be page aligned: 4096 bytes = 12 bit
- Very restricted address space in x32 architecture
  - ~8 bit for stack (256 possibilities)
- Much more space for x64
  - ~22 bit for stack

# Re-randomization

- ASLR only applied on exec()
  - With some bugs…
- Not on fork()

# Recap! ASLR

Randomize Memory Layout

Attacker can't call/reference
- what he cant find

Default ASLR randomizes:
- Writeable locations
- Stack
- Heap

Stack canary: detects/blocks overflows

DEP: makes it impossible to execute uploaded code

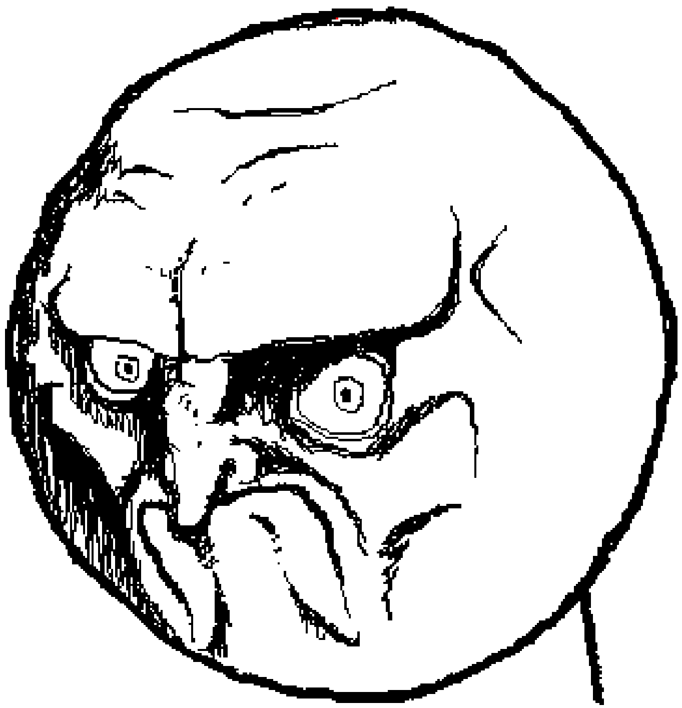
ASLR: makes it impossible to locate data

# Stack canary, DEP, ASLR…
# so many protections…
# now I'm secure!

Time for beer and pizza!

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch



NO.

# This was the simple part…


Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

# Contemporary exploiting

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

# Contemporary exploiting

## Defeating: Stack Canary

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

- Stack canary protects only overflows

- Arbitrary write!

```
char array[16];
array[userIndex] = 0;
```

- Also: Heap is not protected

- Also: Local Vars (function ptr) not protected

Or… just bruteforce it!

- 32 bit value, so 2^32 =~ 4 billion possibilities?
- Example: 0x42A1B2C3

| | | | | | |
|---|---|---|---|---|---|
| AAAAAAA | 0x42 | 0xA1 | 0xB2 | 0xC3 | |
| AAAAAAA | 0x41 | 0xA1 | 0xB2 | 0xC3 | A -> Crash |
| AAAAAAA | 0x42 | 0xA1 | 0xB2 | 0xC3 | B -> No crash |
| AAAAAAA | 0x42 | 0x41 | 0xB2 | 0xC3 | BA -> crash |
| AAAAAAA | 0x42 | 0x42 | 0xB2 | 0xC3 | BB -> crash |

- So: not 2^32 = 4 billion possibilities
- But: 4 * 8 = 4 * 256 = 1024 possibilities
  - 512 on average

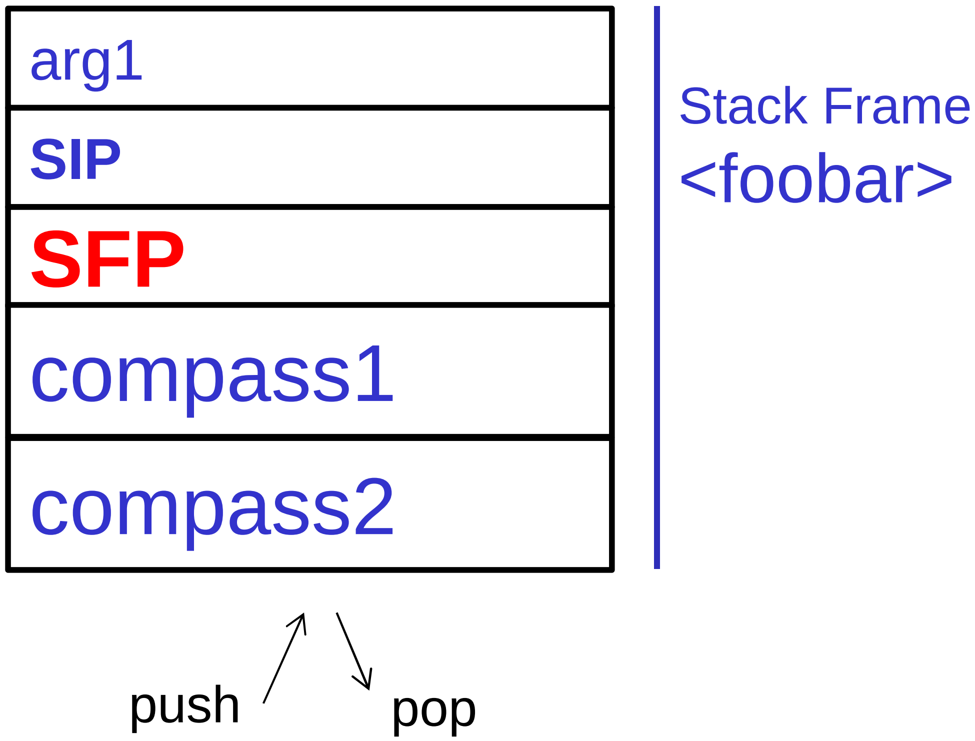
I lied a bit!

Argument for <foobar>

**Saved IP (&main)**

**Saved Frame Pointer**

Local Variables <func>



# Recap: Defeating Stack Canary

- Defeat ASLR for free, because brute force sFP ☺
- Conclusion: Stack Canary is can be brute forced, or just circumvented

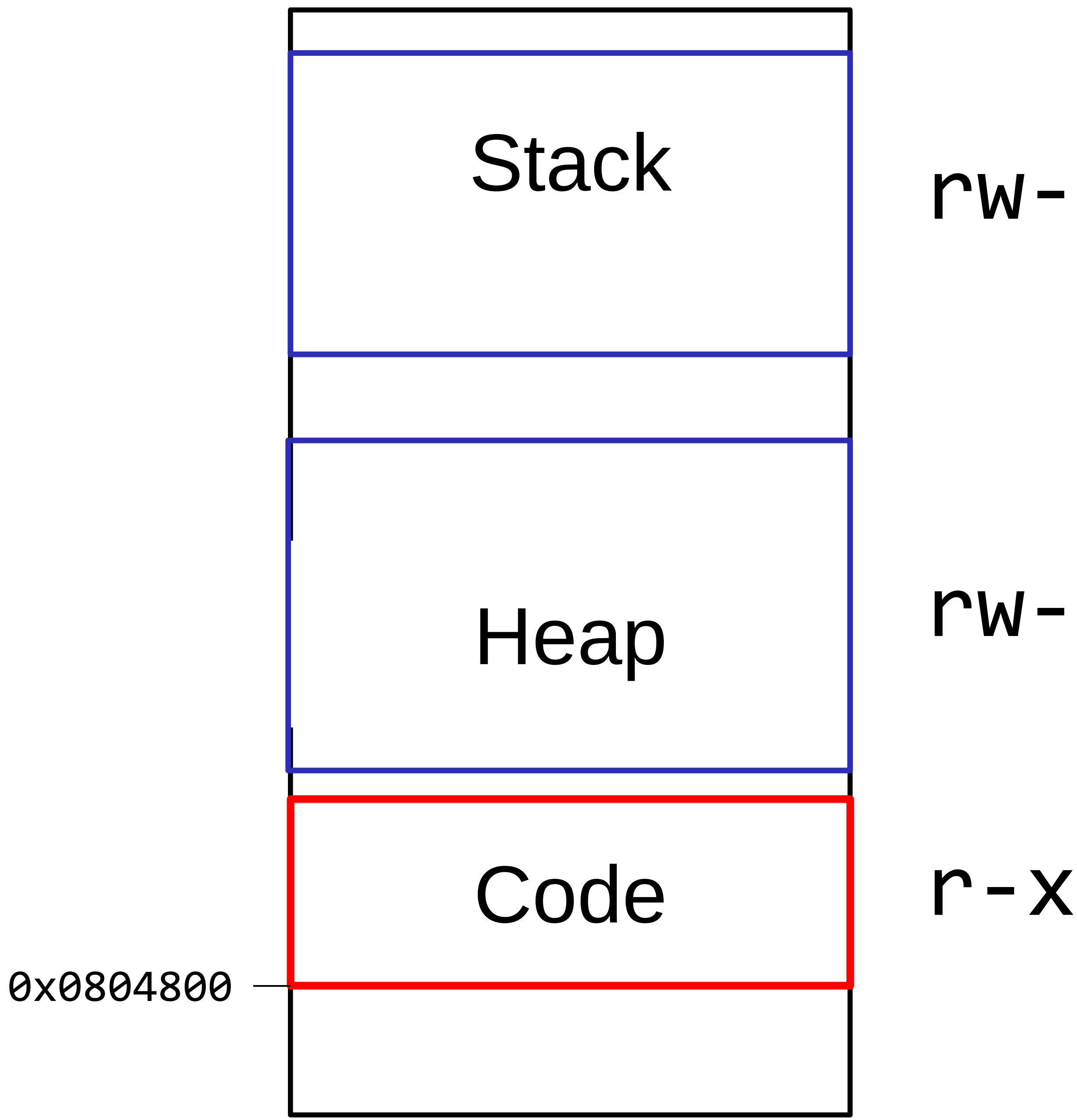
# Contemporary exploiting

## Defeating: DEP

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

# Exploiting: DEP - Memory Layout



- DEP does not allow execution of our own code
- But what about existing code?
- Code from binary, followed by a RET
  - Called "gadgets"
- Return Oriented Programming

# ROPgadget

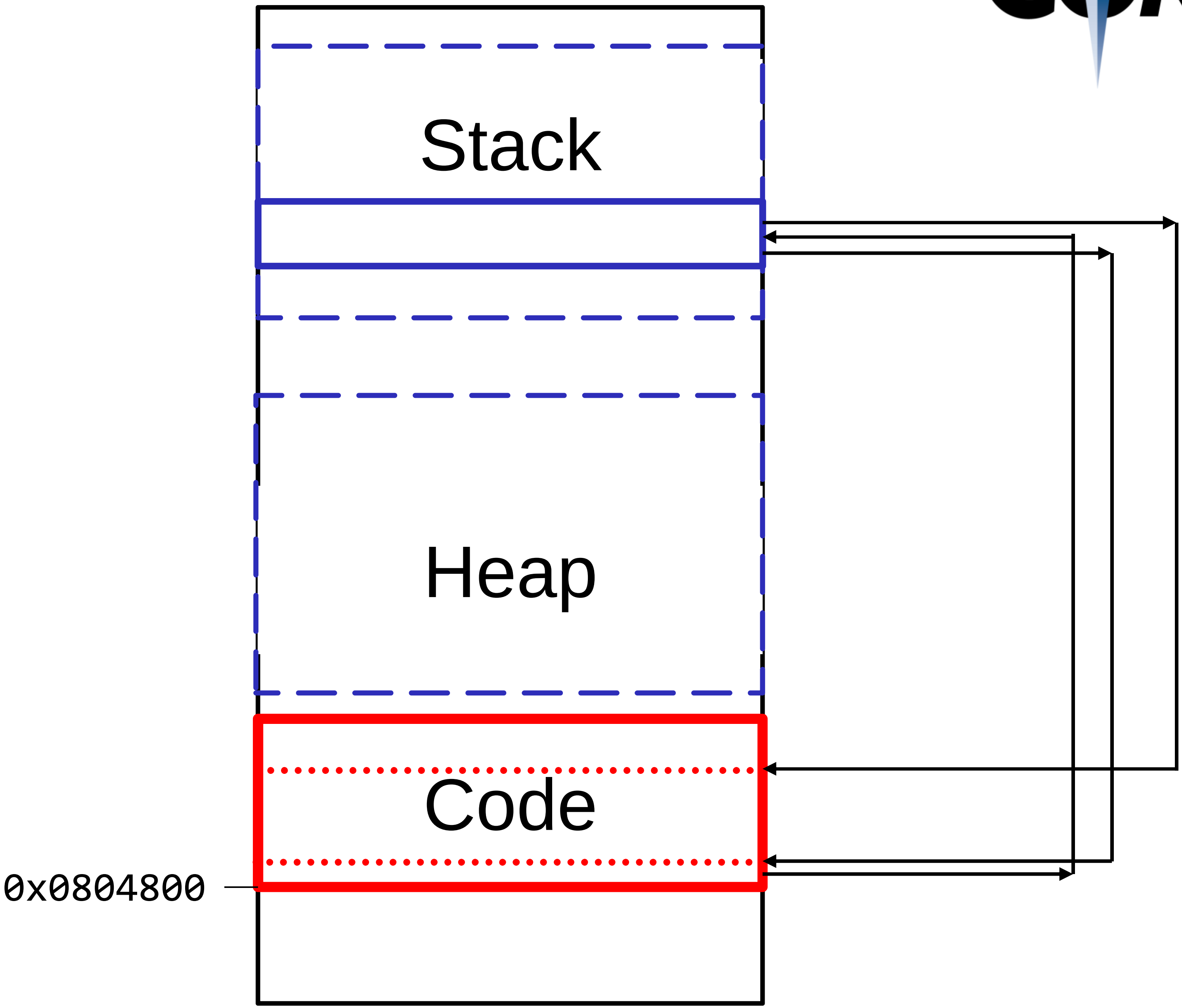
```
0x0000000000440608 : mov dword ptr [rdx], ecx ; ret
0x00000000004598b7 : mov eax, dword ptr [rax + 0xc] ; ret
0x0000000000431544 : mov eax, dword ptr [rax + 4] ; ret
0x000000000045a295 : mov eax, dword ptr [rax + 8] ; ret
0x00000000004a3788 : mov eax, dword ptr [rax + rdi*8] ; ret
0x0000000000493dec : mov eax, dword ptr [rdx + 8] ; ret
0x00000000004a36f7 : mov eax, dword ptr [rdx + rax*8] ; ret
0x0000000000493dc8 : mov eax, dword ptr [rsi + 8] ; ret
0x000000000043fbeb : mov eax, ebp ; pop rbp ; ret
0x00000000004220fa : mov eax, ebx ; pop rbx ; ret
0x0000000000495b90 : mov eax, ecx ; pop rbx ; ret
0x0000000000482498 : mov eax, edi ; pop rbx ; ret
0x0000000000437c11 : mov eax, edi ; ret
0x000000000042cfa1 : mov eax, edx ; pop rbx ; ret
0x000000000047d484 : mov eax, edx ; ret
0x000000000043de7e : mov ebp, esi ; jmp rax
0x0000000000499461 : mov ecx, esp ; jmp rax
0x00000000004324fb : mov edi, dword ptr [rbp] ; call rbx
0x0000000000443f34 : mov edi, dword ptr [rdi + 0x30] ; call rax
0x00000000004607e2 : mov edi, dword ptr [rdi] ; call rsi
0x000000000045c71e : mov edi, ebp ; call rax
0x0000000000491e33 : mov edi, ebp ; call rdx
0x00000000004a7a2d : mov edi, ebp ; nop ; call rax
0x000000000045c4c1 : mov edi, ebx ; call rax
```

# ROPgadget

ROPgadget.py --ropchain

```
ROP chain generation
===========================================================

- Step 1 -- Write-what-where gadgets

        [+] Gadget found: 0x806f702 mov dword ptr [edx], ecx ; ret
        [+] Gadget found: 0x8056c2c pop edx ; ret
        [+] Gadget found: 0x8056c56 pop ecx ; pop ebx ; ret
        [-] Can't find the 'xor ecx, ecx' gadget. Try with another 'mov [r], r'

        [+] Gadget found: 0x808fe0d mov dword ptr [edx], eax ; ret
        [+] Gadget found: 0x8056c2c pop edx ; ret
        [+] Gadget found: 0x80c5126 pop eax ; ret
        [+] Gadget found: 0x80488b2 xor eax, eax ; ret

- Step 2 -- Init syscall number gadgets

        [+] Gadget found: 0x80488b2 xor eax, eax ; ret
        [+] Gadget found: 0x807030c inc eax ; ret

- Step 3 -- Init syscall arguments gadgets

        [+] Gadget found: 0x80481dd pop ebx ; ret
        [+] Gadget found: 0x8056c56 pop ecx ; pop ebx ; ret
        [+] Gadget found: 0x8056c2c pop edx ; ret

- Step 4 -- Syscall gadget

        [+] Gadget found: 0x804936d int 0x80

- Step 5 -- Build the ROP chain

        #!/usr/bin/env python2
        # execve generated by ROPgadget v5.2

        from struct import pack

        # Padding goes here
        p = ''

        p += pack('<I', 0x08056c2c) # pop edx ; ret
        p += pack('<I', 0x080f4060) # @ .data
        p += pack('<I', 0x080c5126) # pop eax ; ret
        p += '/bin'
        p += pack('<I', 0x0808fe0d) # mov dword ptr [edx], eax ; ret
        p += pack('<I', 0x08056c2c) # pop edx ; ret
        p += pack('<I', 0x080f4064) # @ .data + 4
        p += pack('<I', 0x080c5126) # pop eax ; ret
        p += '//sh'
```

# Exploiting: DEP - Memory Layout



Stager:

- Allocate new RWX memory
- Copy rest of shellcode to newly allocated memory
- Execute it (jmp)
- Profit

# Recap: Anti-DEP

Conclusion:

Code section is not randomized

Just smartly re-use existing code

# Contemporary exploiting

## Defeating: ASLR

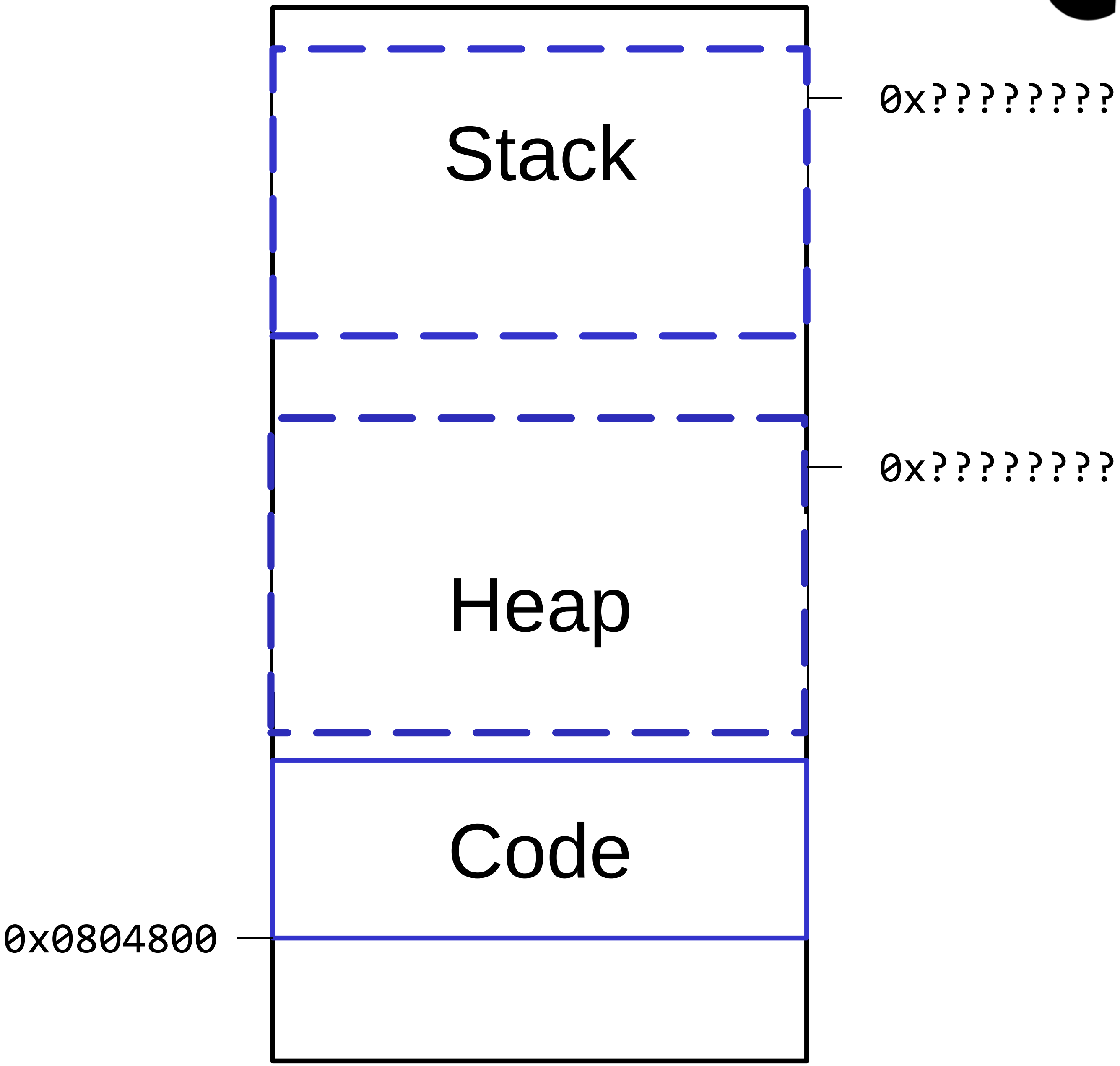
Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

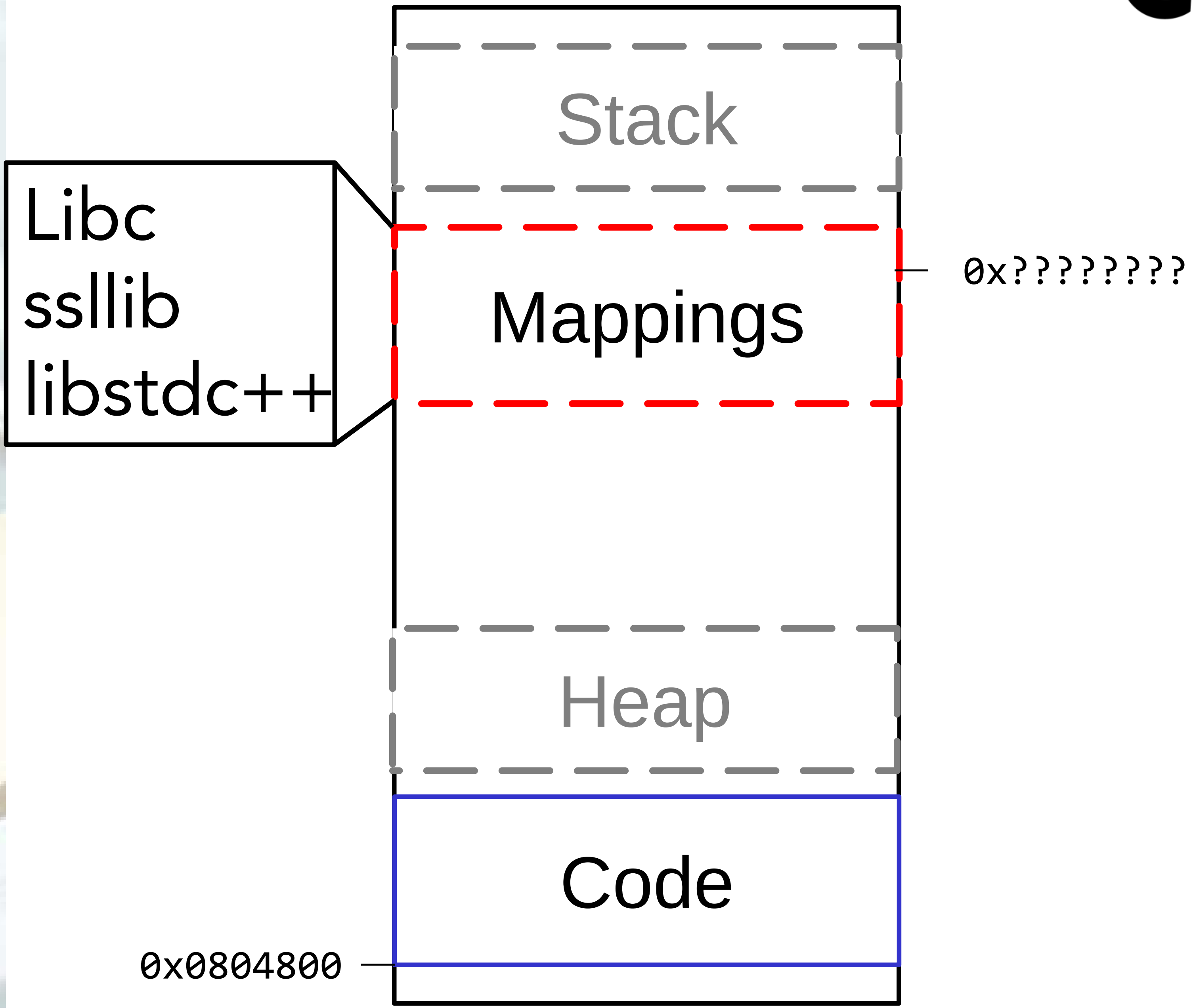
# Exploiting: ASLR

I lied… again

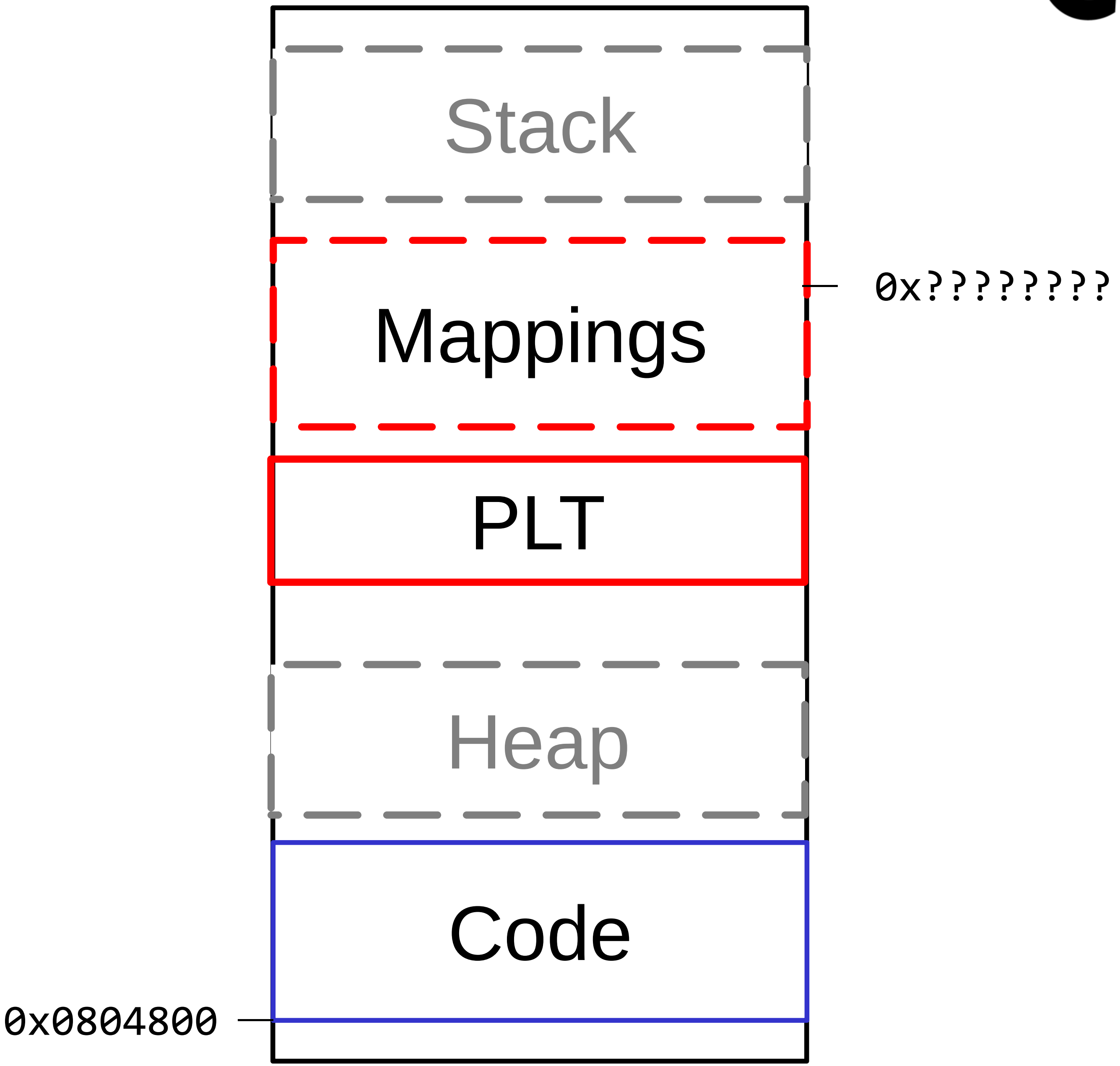
# Exploiting: ASLR – Memory Layout



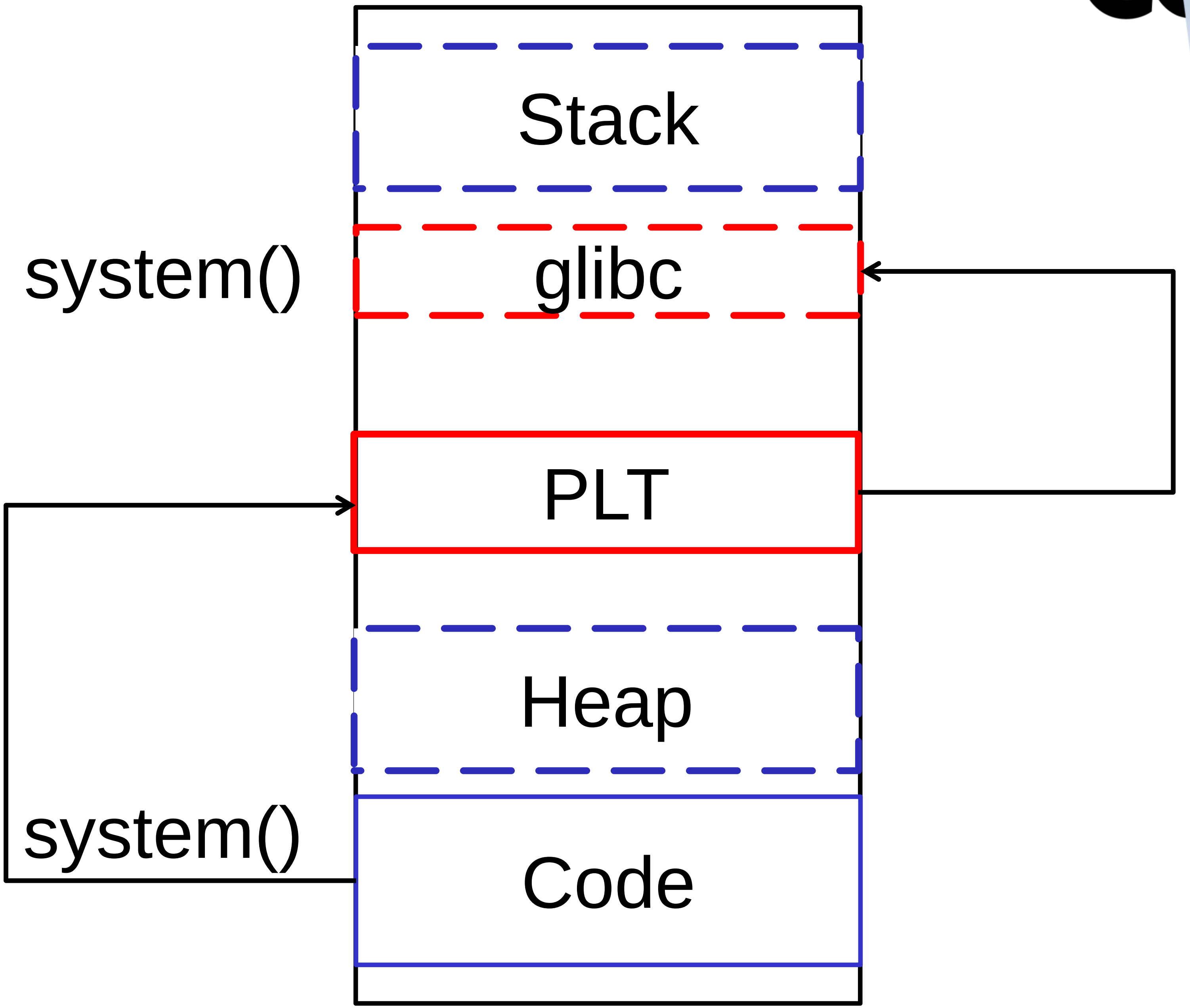
# Exploiting: ASLR – Memory Layout



# Exploiting: ASLR – Memory Layout



# Exploiting: ASLR – Memory Layout



# Exploiting: ASLR – ret2libc / ret2plt

- Defeats ASLR
- Also defeats DEP in one go ☺
- Just do:
  - EIP = &system@plt
  - arg = &meterpreter bash shellcode

  - system("/bin/bash nc –l –p 31337")

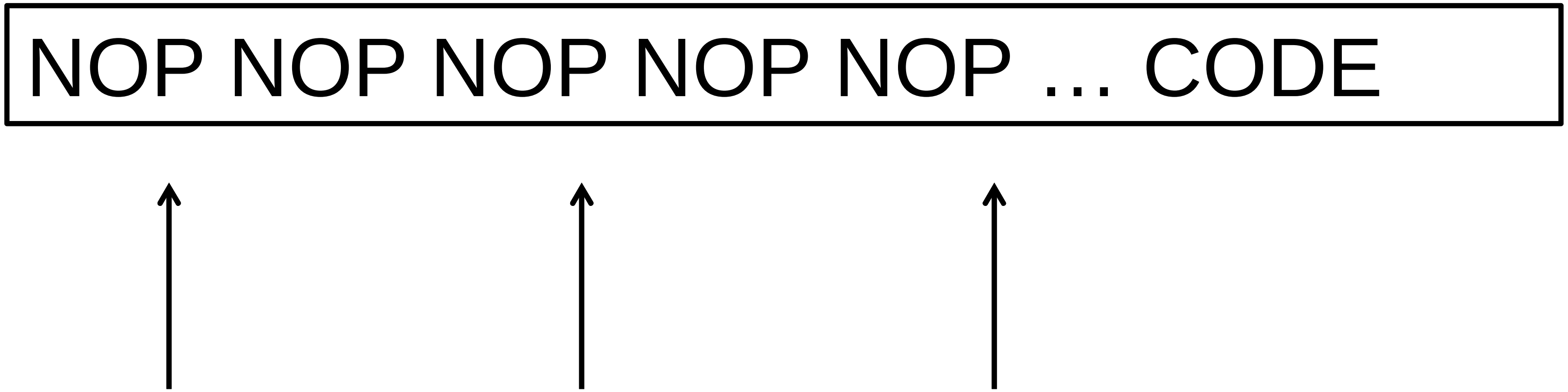
# Other ASLR exploits:

- Partial RIP overwrite
- little endianness: 0x11223344

| | | | | | | |
|---|---|---|---|---|---|---|
| buf | 44 | 33 | 22 | 11 | → | func1 |
| buf | 52 | 33 | 22 | 11 | → | func2 |

# Other ASLR exploits:

- NOP sleds
  - As often used with JavaScript
  - Heap spray a few megabytes…



# Anti-ASLR:

- Find static locations (like PLT)
- Mis-use existing pointers
- Spray & Pray

All three exploit mitigations can be defeated by black magic

Easily

Is there a solution?

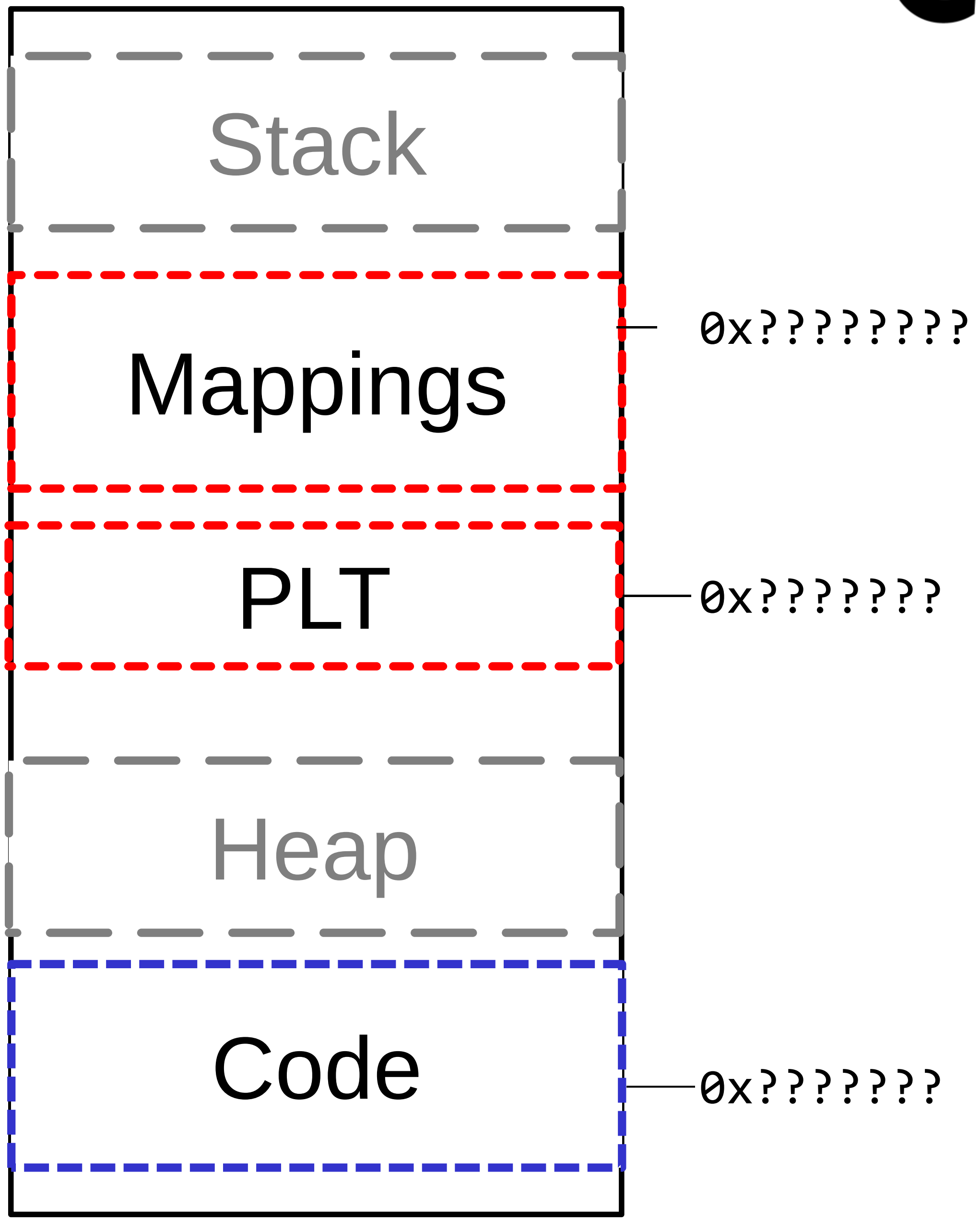
# The solution

The solution to all problems… PIE

- Fix:
  - Compile as PIE
  - PIE: Position Independent Executable
  - Will randomize Code and PLT, too

- Note:
  - Shared libraries are PIC
    - (Position Independent Code)
  - Because they don't know where they are being loaded

# Exploiting: ASLR for code: PIE



# Ok ok, everything is now ASLR'd and secure
# Can I get my pizza now?


Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch


[the cake is a lie]

ASLR assumes attacker can't get information

What if they can?

Meet: Memory Leak

| char **buf1**[16] | *ptr | SFP | EIP |
| --- | --- | --- | --- |

send(socket, buf1, sizeof(int) * 16, NULL);

Oups, attacker got 64 bytes back

- Pointer to stack, code, heap
- Can deduce base address

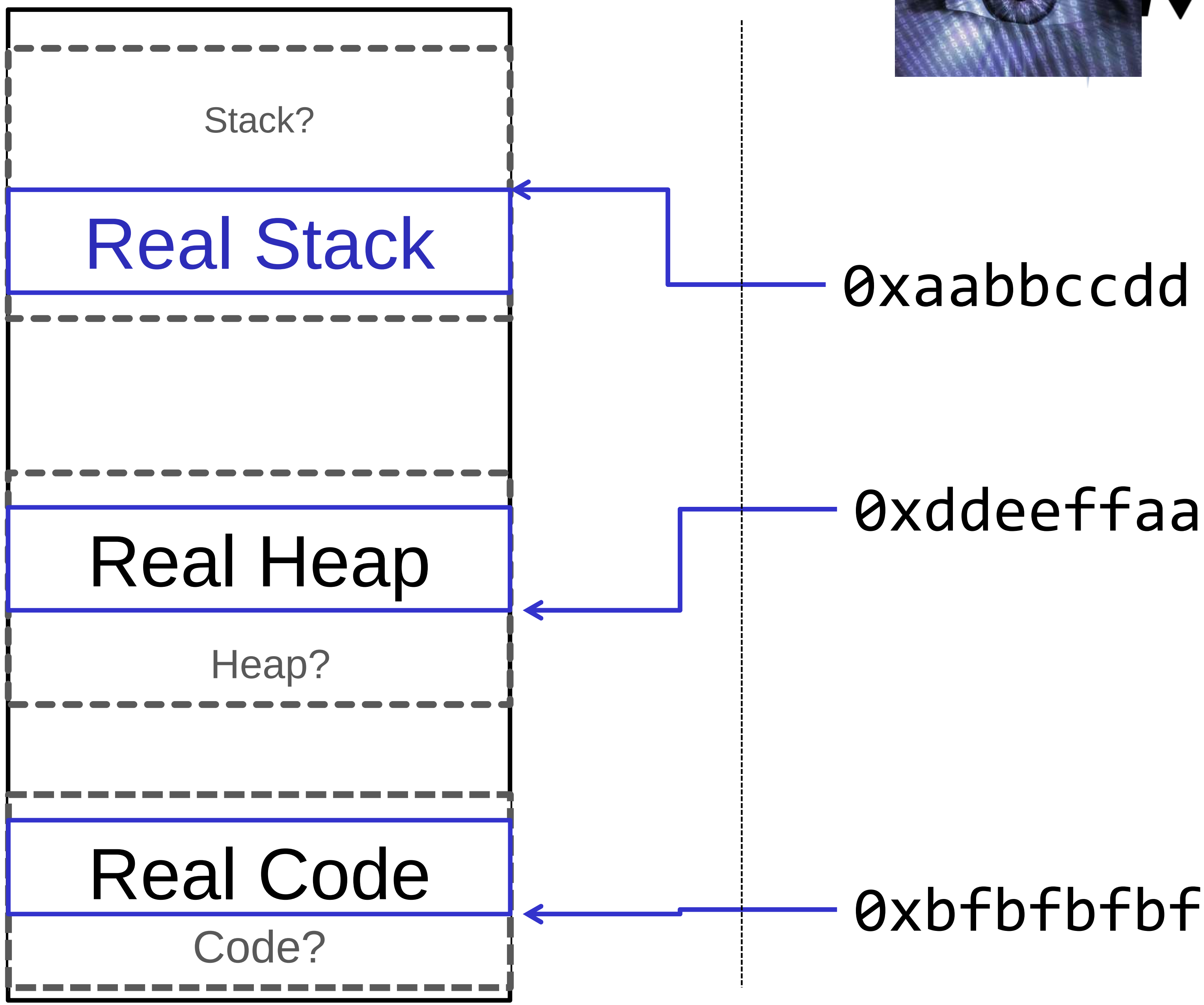
# ASLR vs Memory Leak

| | | | |
|---|---|---|---|
| char **buf1**[16] | *ptr | SFP | EIP |

send(socket, buf1, sizeof(int) * 16, NULL);

| | | | |
|---|---|---|---|
| char **buf1**[16] | *ptr | SFP | EIP |

# Exploiting: ASLR for code: PIE



# TL;DR

Enable ALL the mitigations (DEP, ASLR w/PIE, Stack Protector)

Defeat ALL the mitigations:

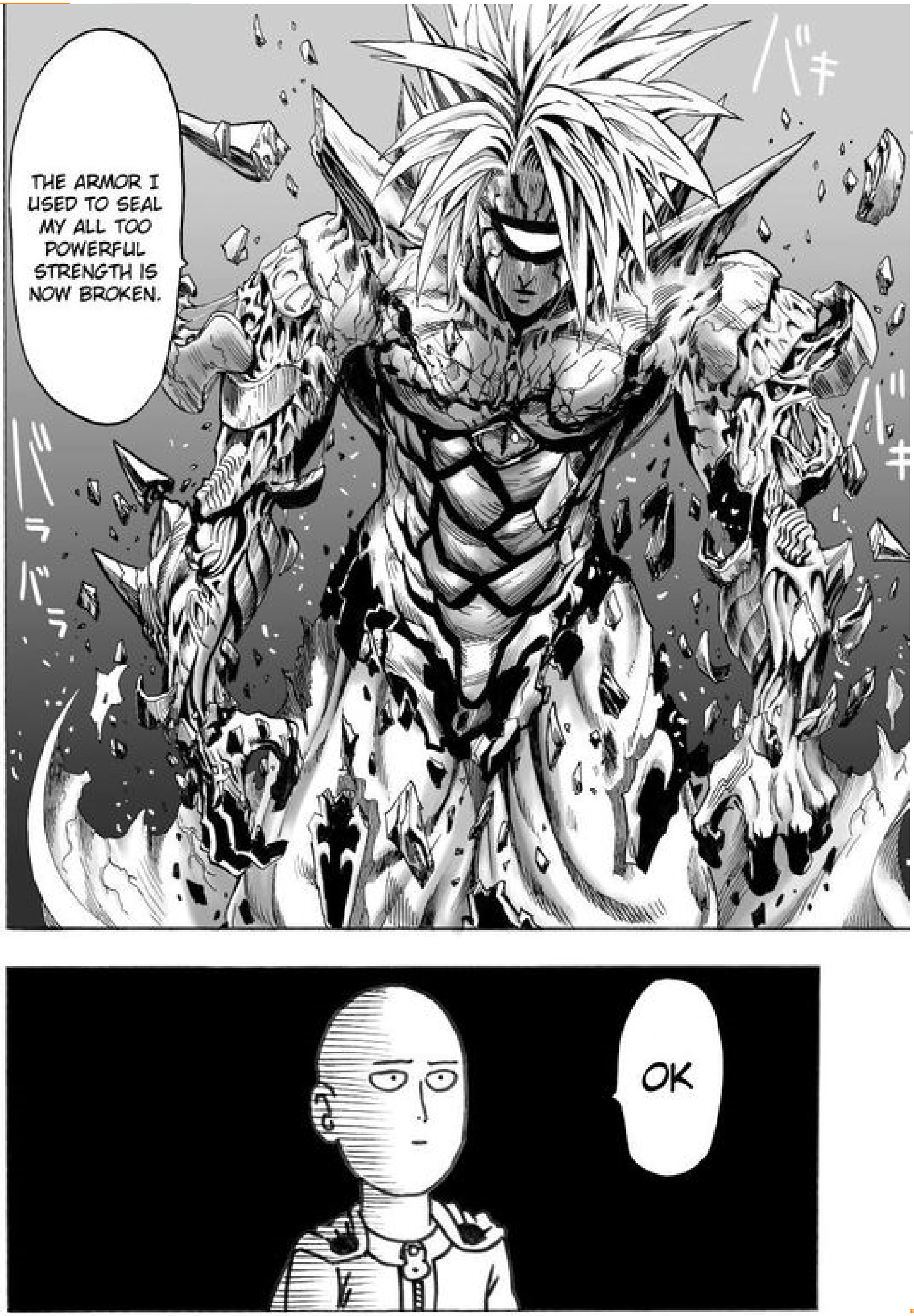
- ROP shellcode as stager to defeat DEP
- Information leak to defeat ASLR
- Non sIP-based stack-overflow vulnerability

# Comparison

| | Stack Canary |
|---|---|
| Stack Overflow | |
| Inter-Chunk Heap Overflow | |
| Arbitrary Write | |
| Use After Free | |
| Type confusion | |

# Comparison

| | DEP | ASLR | DEP + ASLR | PIE + DEP + ASLR |
|---|---|---|---|---|
| Shellcode | | | | |
| Shellcode + Heap Spray | | | | |
| Shellcode + Info Leak | | | | |
| | | | | |
| Ret2libc | | | | |
| Ret2plt | | | | |
| Ret2plt + Infoleak | | | | |
| | | | | |
| ROP | | | | |
| ROP + Info Leak | | | | |




There are multiple Exploit Mitigations

All fail with the right vulnerability

But: They make exploit development harder

Somewhat

As shown in CTF’s at hacker cons, vulns can be identified and a reliable exploit developed in a few hours

- While being drunk
- Int3pids, dragon sector, shellphish, etc.
- Tomorrow, 17:30, Insomnihack Geneve

# Linux Hardening

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

# Exploit Mitigations

Enable DEP:

- Default since like forever
- (for old cpus: kernel.exec-shield = 1)
- To disable for a binary: gcc -z noexecstack

Enable ASLR:

- Default since like forever
- /proc/sys/kernel/randomize_va_space = 2

Enable Stack protector:

- -fstack-protector (Default)
- -fstack-protector-all (ALL Functions)
- -fstack-protector-strong (Better)

# Anti-Exploitation - Hardening

## More Compiler options:

- -D_FORTIFY_SOURCE=2
  - FORTIFY_SOURCE provides (lightweight) buffer overflow checks for the following functions:
    - memcpy, mempcpy, memmove, memset, strcpy, stpcpy, strncpy, strcat, strncat, sprintf, vsprintf, snprintf, vsnprintf, gets.
  - Compile time warnings
  - Default in Ubuntu
- Formatstring
  - Default in Ubuntu
  - -Wformat -Wformat-security
- Full Static Relocation:
  - Default in Ubuntu
  - -Wl,-z-,relro -Wl,-z,now
- Position independent code
  - NOT Default in Ubuntu (*performance penalty*)
  - -pie –fPIE

# Ubuntu Packages Compiled as PIE

| Source package | 8.04 LTS | 9.04 | 9.10 | 10.04 LTS | 10.10 | 11.04 | 11.10 |
|---|---|---|---|---|---|---|---|
| openssh (native) | yes | yes | yes | yes | yes | yes | yes |
| apache2 | -- | yes | yes | yes | yes | yes | yes |
| bind9 | -- | yes | yes | yes | yes | yes | yes |
| openldap | -- | yes | yes | yes | yes | yes | yes |
| postfix | -- | yes | yes | yes | yes | yes | yes |
| cups | -- | yes | yes | yes | yes | yes | yes |
| postgresql-8.3 | -- | yes | yes | yes | yes | yes | yes |
| samba (native) | -- | yes | yes | yes | yes | yes | yes |
| dovecot | -- | yes | yes | yes | yes | yes | yes |
| dhcp3 | -- | yes | yes | yes | yes | yes | yes |
| ntp | -- | -- | yes | yes | yes | yes | yes |
| amavisd-new | -- | -- | yes | yes | yes | yes | yes |
| squid | -- | -- | yes | yes | yes | yes | yes |
| cyrus-sasl2 | -- | -- | yes | yes | yes | yes | yes |
| exim4 | -- | -- | yes | yes | yes | yes | yes |
| nagios3 | -- | -- | yes | yes | yes | yes | yes |
| nagios-plugins | -- | -- | yes | yes | yes | yes | yes |
| xinetd | -- | -- | yes | yes | yes | yes | yes |
| ipsec-tools | -- | -- | yes | yes | yes | yes | yes |
| mysql-dfsg-5.1 | -- | -- | yes | yes | yes | yes | yes |
| evince | -- | -- | -- | yes | yes | yes | yes |
| firefox | -- | -- | -- | yes | yes | yes | yes |
| gnome-control-center | -- | -- | -- | -- | -- | yes | yes |
| tiff | -- | -- | -- | -- | -- | yes | yes |
| totem | -- | -- | -- | -- | -- | yes | yes |
| qemu-kvm | -- | -- | -- | -- | -- | -- | yes |
| pidgin | -- | -- | -- | -- | -- | -- | yes |

# Check: Checksec

```
              init  1235 Full RELRO      Canary found     NX enabled  PIE enabled
       dbus-launch  1436 Partial RELRO   Canary found     NX enabled  No PIE
       dbus-daemon  1453 Partial RELRO   Canary found     NX enabled  No PIE
       dbus-daemon  1454 Partial RELRO   Canary found     NX enabled  No PIE
   upstart-event-b  1465 Full RELRO      No canary found  NX enabled  PIE enabled
   window-stack-br  1471 Partial RELRO   No canary found  NX enabled  No PIE
   upstart-dbus-br  1486 Full RELRO      No canary found  NX enabled  PIE enabled
   upstart-dbus-br  1488 Full RELRO      No canary found  NX enabled  PIE enabled
   upstart-file-br  1497 Full RELRO      Canary found     NX enabled  PIE enabled
       ibus-daemon  1503 Partial RELRO   Canary found     NX enabled  No PIE
   unity-settings-  1517 Partial RELRO   No canary found  NX enabled  No PIE
        bamfdaemon  1519 Partial RELRO   Canary found     NX enabled  No PIE
   at-spi-bus-laun  1523 Full RELRO      Canary found     NX enabled  PIE enabled
     gnome-session  1524 Partial RELRO   Canary found     NX enabled  No PIE
       dbus-daemon  1529 Partial RELRO   Canary found     NX enabled  No PIE
             gvfsd  1533 Partial RELRO   No canary found  NX enabled  No PIE
        ibus-dconf  1538 Partial RELRO   No canary found  NX enabled  No PIE
      ibus-ui-gtk3  1539 Partial RELRO   No canary found  NX enabled  No PIE
          ibus-x11  1542 Partial RELRO   Canary found     NX enabled  No PIE
        gvfsd-fuse  1545 Partial RELRO   No canary found  NX enabled  No PIE
   at-spi2-registr  1555 Full RELRO      Canary found     NX enabled  PIE enabled
        pulseaudio  1645 Full RELRO      Canary found     NX enabled  No PIE
   ibus-engine-sim  1692 Partial RELRO   No canary found  NX enabled  No PIE
          metacity  1775 Partial RELRO   Canary found     NX enabled  No PIE
     dconf-service  1781 Partial RELRO   Canary found     NX enabled  No PIE
       gnome-panel  1819 Partial RELRO   Canary found     NX enabled  No PIE
   indicator-appli  1835 Partial RELRO   No canary found  NX enabled  No PIE
   unity-fallback-  1836 Partial RELRO   No canary found  NX enabled  No PIE
   indicator-bluet  1837 Partial RELRO   No canary found  NX enabled  No PIE
          vmtoolsd  1839 Partial RELRO   Canary found     NX enabled  No PIE
   polkit-gnome-au  1841 Partial RELRO   No canary found  NX enabled  No PIE
          nautilus  1848 Partial RELRO   Canary found     NX enabled  No PIE
         nm-applet  1852 Partial RELRO   Canary found     NX enabled  No PIE
           initctl  1853 Full RELRO      No canary found  NX enabled  PIE enabled
   indicator-messa  1858 Partial RELRO   No canary found  NX enabled  No PIE
   indicator-power  1863 Partial RELRO   No canary found  NX enabled  No PIE
```

# Check: Paxtest

```
Executable anonymous mapping             : Killed
Executable bss                           : Killed
Executable data                          : Killed
Executable heap                          : Killed
Executable stack                         : Killed
Executable shared library bss            : Killed
Executable shared library data           : Killed
Executable anonymous mapping (mprotect)  : Vulnerable
Executable bss (mprotect)                : Vulnerable
Executable data (mprotect)               : Vulnerable
Executable heap (mprotect)               : Vulnerable
Executable stack (mprotect)              : Vulnerable
Executable shared library bss (mprotect) : Vulnerable
Executable shared library data (mprotect): Vulnerable
Writable text segments                   : Vulnerable
```

# Check: Paxtest

```
Anonymous mapping randomization test     : 28 quality bits (guessed)
Heap randomization test (ET_EXEC)        : 13 quality bits (guessed)
Heap randomization test (PIE)            : 28 quality bits (guessed)
Main executable randomization (ET_EXEC)  : 28 quality bits (guessed)
Main executable randomization (PIE)      : 28 quality bits (guessed)
Shared library randomization test        : 28 quality bits (guessed)
VDSO randomization test                  : 11 quality bits (guessed)
Stack randomization test (SEGMEXEC)      : 28 quality bits (guessed)
Stack randomization test (PAGEEXEC)      : 28 quality bits (guessed)
Arg/env randomization test (SEGMEXEC)    : 20 quality bits (guessed)
Arg/env randomization test (PAGEEXEC)    : 20 quality bits (guessed)
Randomization under memory exhaustion @~0: 28 bits (guessed)
Randomization under memory exhaustion @0 : 28 bits (guessed)
Return to function (strcpy)              : return addr has NULL byte
Return to function (memcpy)              : Vulnerable
Return to function (strcpy, PIE)         : return addr has NULL byte
Return to function (memcpy, PIE)         : Vulnerable
```

# Advanced Linux hardening

The non-standard stuff

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

# Grsecurity

Uses PaX

- Kernel patch
- Improved DEP and ASLR
- For userspace
- And kernelspace protection (e.g. SMAP emulation)
- Better randomness, more randomness

Also provides:

- Chroot hardening
- Hide /proc stuff
- Ptrace restrictions
- Kernel module loading restrictions
- RBAC (Role Based Access Control)

# Ok I now know everything, pizza?


AG
CH-8645 Jona
Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

# Container

## Linux Container


Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

# Linux Container

## Relevant?

- TEH CLOUD

## Container: All container share the same kernel

- LXC
- Docker
- FreeBSD Jails (since March 2000)
- Solaris Zones
- Obsolete: Vserver, openvz

## Virtualization: Each guest has his very own kernel

- Vmware, virtualbox, kvm, …
- Not covered here

## RBAC's

- SELinux (redhat), Apparmour (Suse), …
- Not convered here

- Chroot is not a container
  - Path restriction only
  - But: Can access other processes, the kernel, IPC, etc.

```
compass@ubuntu:~$ sudo chroot /var/chroot
root@ubuntu:/# cd root/
root@ubuntu:/root# ./w00t -0 --dir /nonexisting
clssic
[+] creating /nonexisting directory
[+] chrooting to /nonexisting
[+] change working directory to real root
[+] chrooting to real root
root@ubuntu:/# ls /
bin    cdrom  etc    initrd.img  lib64        media
boot   dev    home   lib         lost+found   mnt
root@ubuntu:/#
```

# LXC/Docker: Use namespaces for containerization

- Restrict view/access of certain processes

Linux provides the following namespaces:

| Namespace | Constant | Isolates |
|---|---|---|
| IPC | CLONE_NEWIPC | System V IPC, POSIX message queues |
| Network | CLONE_NEWNET | Network devices, stacks, ports, etc. |
| Mount | CLONE_NEWNS | Mount points |
| PID | CLONE_NEWPID | Process IDs |
| User | CLONE_NEWUSER | User and group IDs |
| UTS | CLONE_NEWUTS | Hostname and NIS domain name |

## Lxc container cannot:

- Interact with host processes
- Access root file system
- Access special devices (block, network, …)
- Mount filesystems
- Execute special ioctl's

## Lxc container can access:

- /proc: certain files
- /sys: certain files
- Do a lot of other stuff

# LXC container share their Kernel… Wait – what about Kernel security?

Userspace vs. kernelspace

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

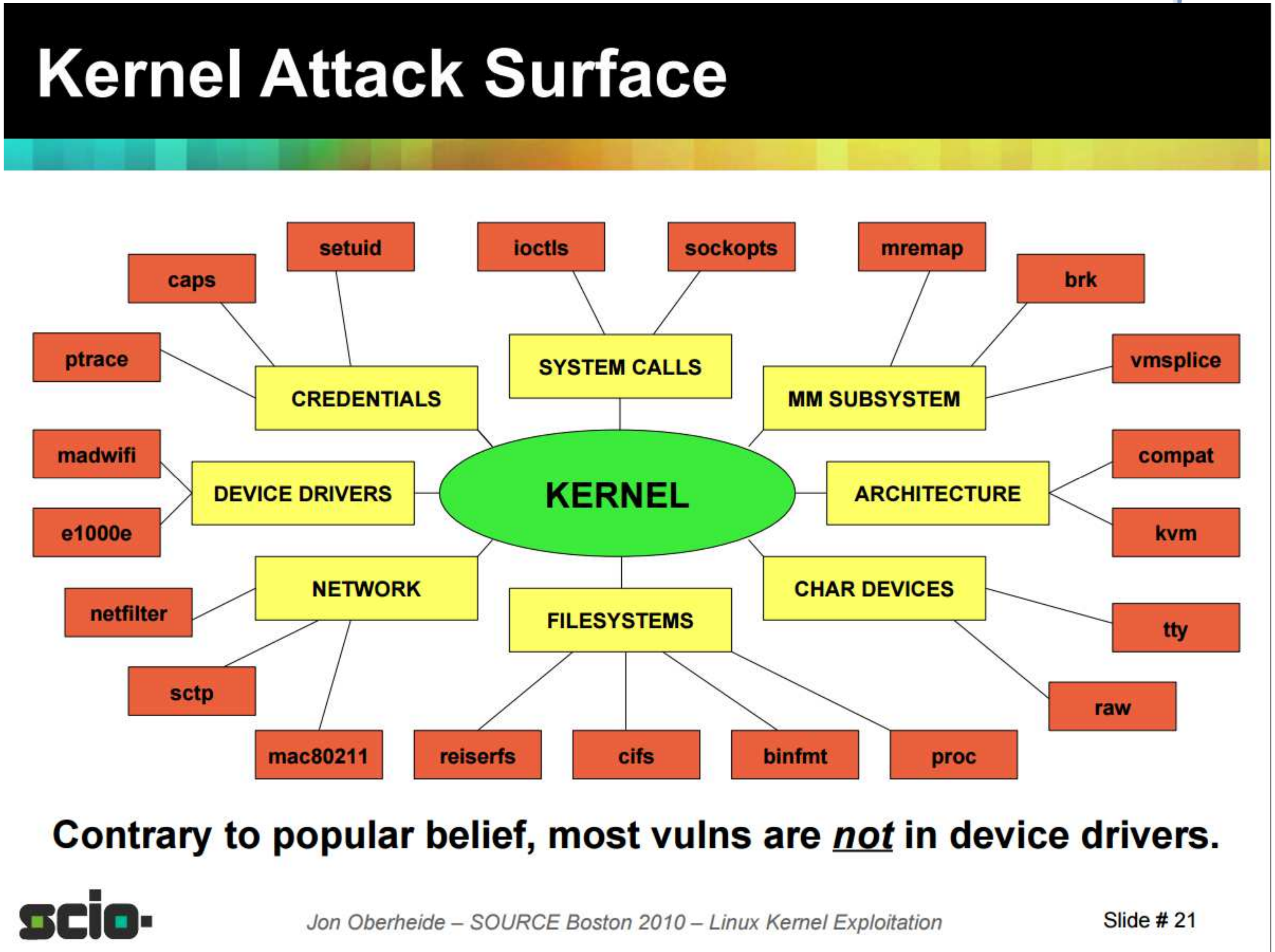
# Linux Kernel

Protecting the kernelspace

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

# Linux Kernel Attack Surface



# Linux Kernel Syscalls

38 sys_rename
39 sys_mkdir
40 sys_rmdir
41 sys_dup
42 sys_pipe
43 sys_times
44 sys_prof [sys_ni_syscall]
45 sys_brk
46 sys_setgid
47 sys_getgid
48 sys_signal
49 sys_geteuid
50 sys_getegid
51 sys_acct
52 sys_umount2 [sys_umount] (2.2+)
53 sys_lock [sys_ni_syscall]
54 sys_ioctl
55 sys_fcntl
56 sys_mpx [sys_ni_syscall]
57 sys_setpgid
58 sys_ulimit [sys_ni_syscall]
59 sys_oldolduname
60 sys_umask
61 sys_chroot
62 sys_ustat
63 sys_dup2
64 sys_getppid
65 sys_getpgrp
66 sys_setsid
67 sys_sigaction
68 sys_sgetmask
69 sys_ssetmask

108 sys_fstat [sys_newfstat]
109 sys_olduname [sys_uname]
110 sys_iopl
111 sys_vhangup
112 sys_idle
113 sys_vm86old
114 sys_wait4
115 sys_swapoff
116 sys_sysinfo
117 sys_ipc
118 sys_fsync
119 sys_sigreturn
120 sys_clone
121 sys_setdomainname
122 sys_uname [sys_newuname]
123 sys_modify_ldt
124 sys_adjtimex
125 sys_mprotect
126 sys_sigprocmask
127 sys_create_module
128 sys_init_module
129 sys_delete_module
130 sys_get_kernel_syms
131 sys_quotactl
132 sys_getpgid
133 sys_fchdir
134 sys_bdflush
135 sys_sysfs
136 sys_personality
137 sys_afs_syscall [sys_ni_syscall]
138 sys_setfsuid
139 sys_setfsgid

178 sys_rt_sigqueueinfo (2.2+)
179 sys_rt_sigsuspend (2.2+)
180 sys_pread (2.2+)
181 sys_pwrite (2.2+)
182 sys_chown (2.2+)
183 sys_getcwd (2.2+)
184 sys_capget (2.2+)
185 sys_capset (2.2+)
186 sys_sigaltstack (2.2+)
187 sys_sendfile (2.2+)
188 sys_getpmsg [sys_ni_syscall]
189 sys_putpmsg [sys_ni_syscall]
190 sys_vfork (2.2+)

# Linux Kernel Protection Mechanisms

So, what about the Linux Kernel?

ASLR: No

- kASLR
- Since Kernel 3.14
- Disabled by default in most distributions
- Weaker than userspace (less entropy)
  - But: crash in kernel is very noticable

DEP: Yes

- But some pages are W & X…
  - Because of X86 (BIOS etc. )
  - Therefore, not so useful

Stack Protector: YES

FORTIFY_SOURCE: YES

| | |
|---|---|
| 2012: Ivy Bridge | (e.g. i7 48xx, 49xx) |
| 2013: Haswell | (e.g. i7 47xx) |
| 2014: Broadwell | (e.g. i7 56xx, 55xx, 58xx, 59xx) |
| 2015: Skylake | (e.g. i7 65xx, 66xx, 67xx, 68xx, 69xx) |

## SMEP: Supervisor Mode Execution Protection

- Deny Kernel execution from userspace memory (ret2usr)
- Since Kernel 3.0
- Needs CPU support: Ivy Bridge ++
- Enabled by default in modern distributions
- Workaround: In-kernel ROP
- cat /proc/cpuinfo | grep smep

## SMAP: Supervisor Mode Access Prevention

- Deny Kernel direct access to userspace memory
- Since Kernel 3.7
- Needs CPU support: Broadwell ++
- Enabled by default in modern distributions

# Linus on security

```
From: Linus Torvalds <torvalds@linux-foundation.org>
Newsgroups: fa.linux.kernel
Subject: Re: [stable] Linux 2.6.25.10
Date: Tue, 15 Jul 2008 02:28:23 UTC
Message-ID: <fa.FocnvcnLqG7kPaYdjYdPJfSdhjc@ifi.uio.no>

On Tue, 15 Jul 2008, pageexec@freemail.hu wrote:
>
> so guys (meaning not only Greg but Andrew, Linus, et al.), when will you
> publicly explain why you're covering up security impact of bugs? and even
> more importantly, when will you change your policy or bring your process
> in line with what you declared?

We went through this discussion a couple of weeks ago, and I had
absolutely zero interest in explaining it again.

I personally don't like embargoes. I don't think they work. That means
that I want to fix things asap. But that also means that there is never a
time when you can "let people know", except when it's not an issue any
more, at which point there is no _point_ in letting people know any more.

So I personally consider security bugs to be just "normal bugs". I don't
cover them up, but I also don't have any reason what-so-ever to think it's
a good idea to track them and announce them as something special.

So there is no "policy". Nor is it likely to change.

                        Linus
```

```
From: Linus Torvalds <torvalds@linux-foundation.org>
Newsgroups: fa.linux.kernel
Subject: Re: [stable] Linux 2.6.25.10
Date: Tue, 15 Jul 2008 16:14:00 UTC
Message-ID: <fa.a8PYBfKwFq0Fl6ls/kFjBfKbV44@ifi.uio.no>

On Tue, 15 Jul 2008, Linus Torvalds wrote:
>
> So as far as I'm concerned, "disclosing" is the fixing of the bug. It's
> the "look at the source" approach.

Btw, and you may not like this, since you are so focused on security, one
reason I refuse to bother with the whole security circus is that I think
it glorifies - and thus encourages - the wrong behavior.

It makes "heroes" out of security people, as if the people who don't just
fix normal bugs aren't as important.

In fact, all the boring normal bugs are _way_ more important, just because
there's a lot more of them. I don't think some spectacular security hole
should be glorified or cared about as being any more "special" than a
random spectacular crash due to bad locking.

Security people are often the black-and-white kind of people that I can't
stand. I think the OpenBSD crowd is a bunch of masturbating monkeys, in
that they make such a big deal about concentrating on security to the
point where they pretty much admit that nothing else matters to them.

To me, security is important. But it's no less important than everything
*else* that is also important!

                        Linus
```

http://yarchive.net/comp/linux/security_bugs.html

http://www.washingtonpost.com/sf/business/2015/11/05/net-of-insecurity-the-kernel-of-the-argument/

# Linux Kernel Politics

## Infrastructure people

- Don't know they are there, except when something breaks

## 8 stable kernel trees!

- And Distros have their own stable kernels…

## Actively hide security fixes in commit messages

- And they are honest about this

## Distros in charge of security

- Is this good or not?

## Conclusion:

- Important security fixes are maybe not backported to stable kernels

# Kernel Hardening

## Reduce features!

- Make menuconfig
- Remove:
  - Drivers
    - CVE-2016-2384: arbitrary code execution due to a double-free in the usb-midi linux kernel driver
  - Features
  - Protocols

## Use grsecurity / PaX

## Use current CPU

## Enable kASLR

# Seccomp-bpf

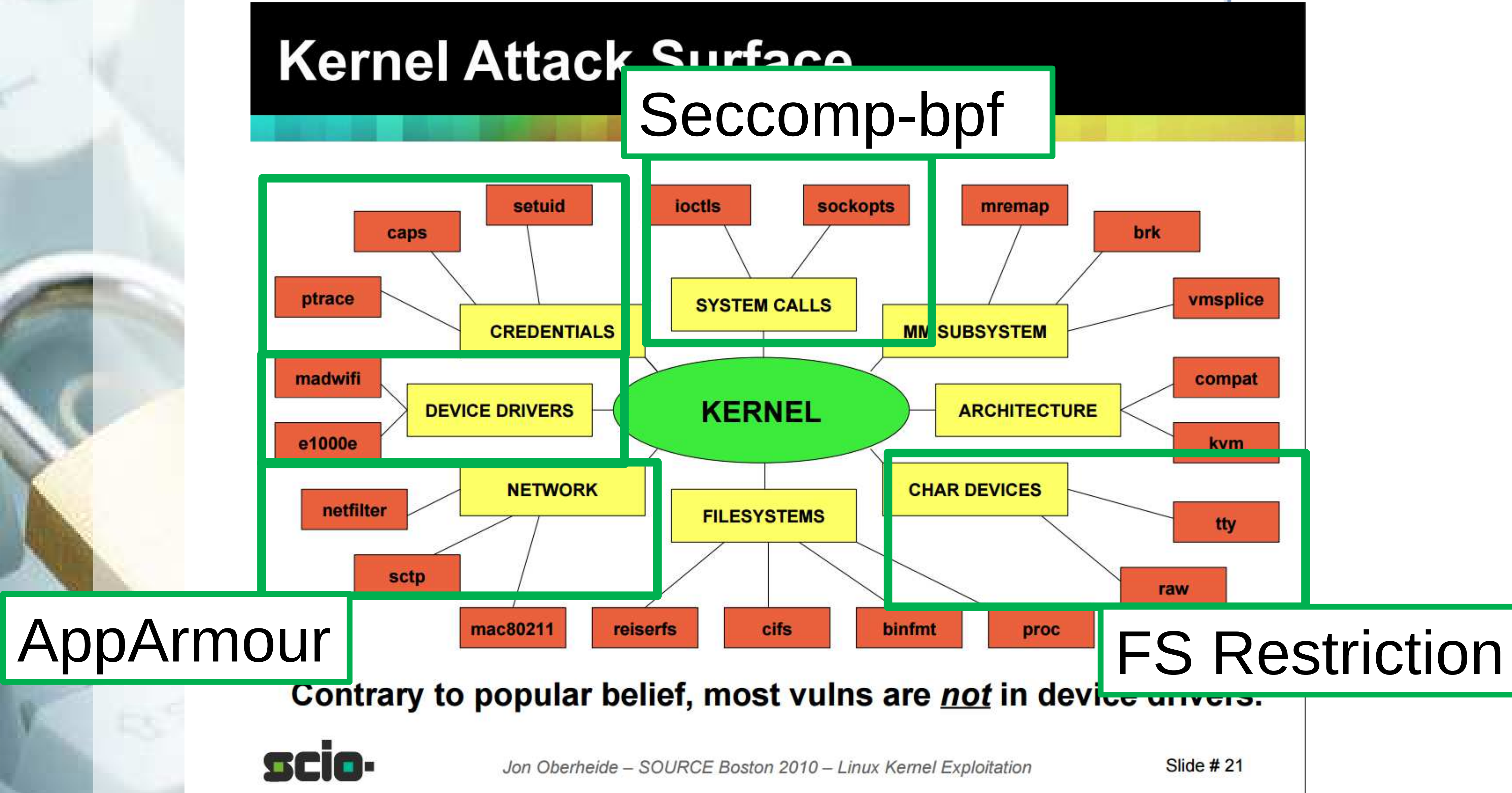
- Seccomp: Since Kernel 2.6.12 (2005)
- Seccomp-bpf: Since Kernel 3.5 (2012)
- Whitelist (blacklist) system calls
  - E.g. exit(), read(), write(), …
- Who cares?
  - Chrome-Flash, Chrome-Renderer, vsftpd, OpenSSH, Firefox, Tor, …

# FS hardening

- /proc
- /sys
- /dev/[zero, null, urandom]
- Nothing else

# AppArmour

- "additional restrictions on mounts, socket, ptrace and file access. Specifically restricting cross-container communication."


Kernel Attack Surface
Seccomp-bpf
setuid
caps
ptrace
CREDENTIALS
ioctls
sockopts
SYSTEM CALLS
mremap
brk
vmsplice
MM SUBSYSTEM
madwifi
e1000e
DEVICE DRIVERS
KERNEL
ARCHITECTURE
compat
kvm
NETWORK
netfilter
sctp
FILESYSTEMS
CHAR DEVICES
tty
raw
mac80211
reiserfs
cifs
binfmt
proc
AppArmour
FS Restriction
Contrary to popular belief, most vulns are not in device drivers.
scio
Jon Oberheide – SOURCE Boston 2010 – Linux Kernel Exploitation
Slide # 21

# Recap! Linux Kernel

Container share same Kernel

Kernel is not very secure

# Conclusion

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

# Recap! Recap!

## Want more security?

- Compile everything as PIE

## Even more?

- Grsecurity Kernel patch

## More Kernel security?

- Strip kernel of features
- Seccomp-bpf

-> Or better: Ask your distribution to do it! <-

The good news:

Most companies get owned by web vuln's anyway

- SQL injection
- Shell upload

Or social engineering…

# Questions?


Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

# When I’m quick:
# More slides (backup slides)

Compass Security Schweiz AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

# Advanced Hardening - CLANG CFI

Clang is a C/C++ frontend for LLVM

Has Control Flow Integrity!

- -fsanitize=cfi
- Mostly helps against type confusion attacks

- -fsanitize=cfi-cast-strict: Enables strict cast checks
- -fsanitize=cfi-derived-cast: Base-to-derived cast to the wrong dynamic type.
- -fsanitize=cfi-unrelated-cast: Cast from void* or another unrelated type to the wrong dynamic type.
- -fsanitize=cfi-nvcall: Non-virtual call via an object whose vptr is of the wrong dynamic type.
- -fsanitize=cfi-vcall: Virtual call via an object whose vptr is of the wrong dynamic type.
- -fsanitize=cfi-icall: Indirect call of a function with wrong dynamic type.

## Blind ROP

- Brute force all ROP gadgets
- Based on replies (Crash, Freeze, No Crash)
- No need to know anything about the process (don't even need binary!)

## Sigreturn oriented programming

- Use signal handler to invoke code
- Does not need as many gadgets as normal ROP (just "syscall; ret")

# Clang Compiler

Frontend for LLVM (compiles C to LLVM IR)

SafeStack

- -fsanitize=safe-stack
- Split stack into safe- (SIP etc.) and unsafe stack

# Clang Compiler: Address Sanitizer

- Detects memory corruption bugs
- Heavy performance penalty
- Do not use in production!
  - http://www.openwall.com/lists/oss-security/2016/02/17/9

```
Date: Wed, 17 Feb 2016 23:19:21 +0100
From: Szabolcs Nagy <nsz@...t70.net>
To: oss-security@...ts.openwall.com
Subject: Address Sanitizer local root

There is an alarming trend that Address Sanitizer and related
compiler instrumentations from compiler-rt are used as a hardening
solution and run in production.
```