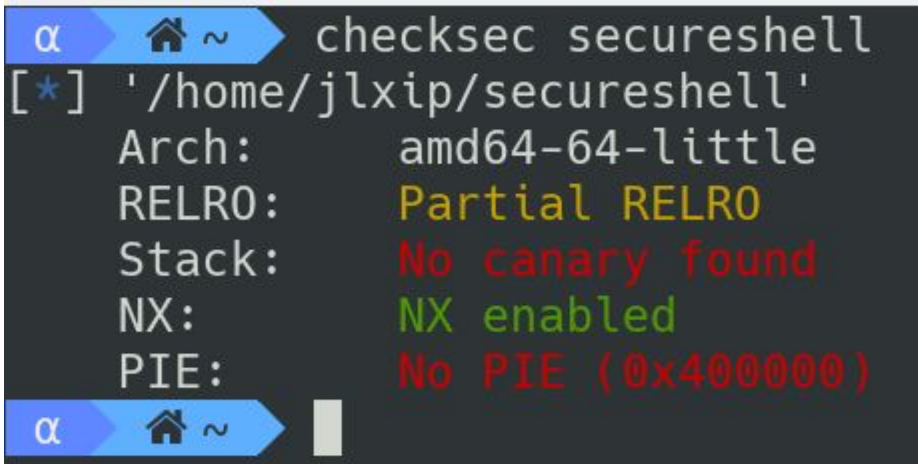
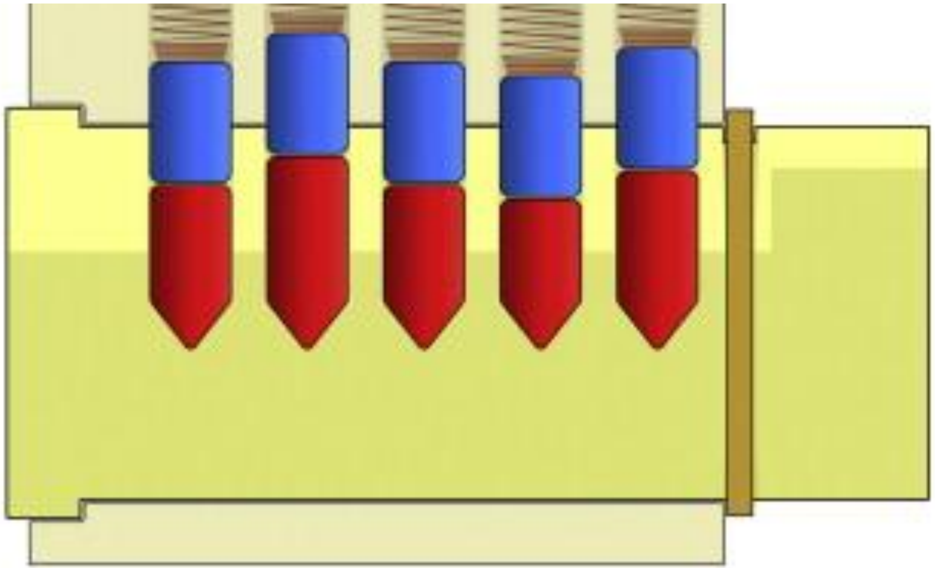
# Bypassing security mitigations

CS412 - Software security



```
α  ~  checksec secureshell
[*] '/home/jlxip/secureshell'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
α  ~
```



NX/DEP

ASLR

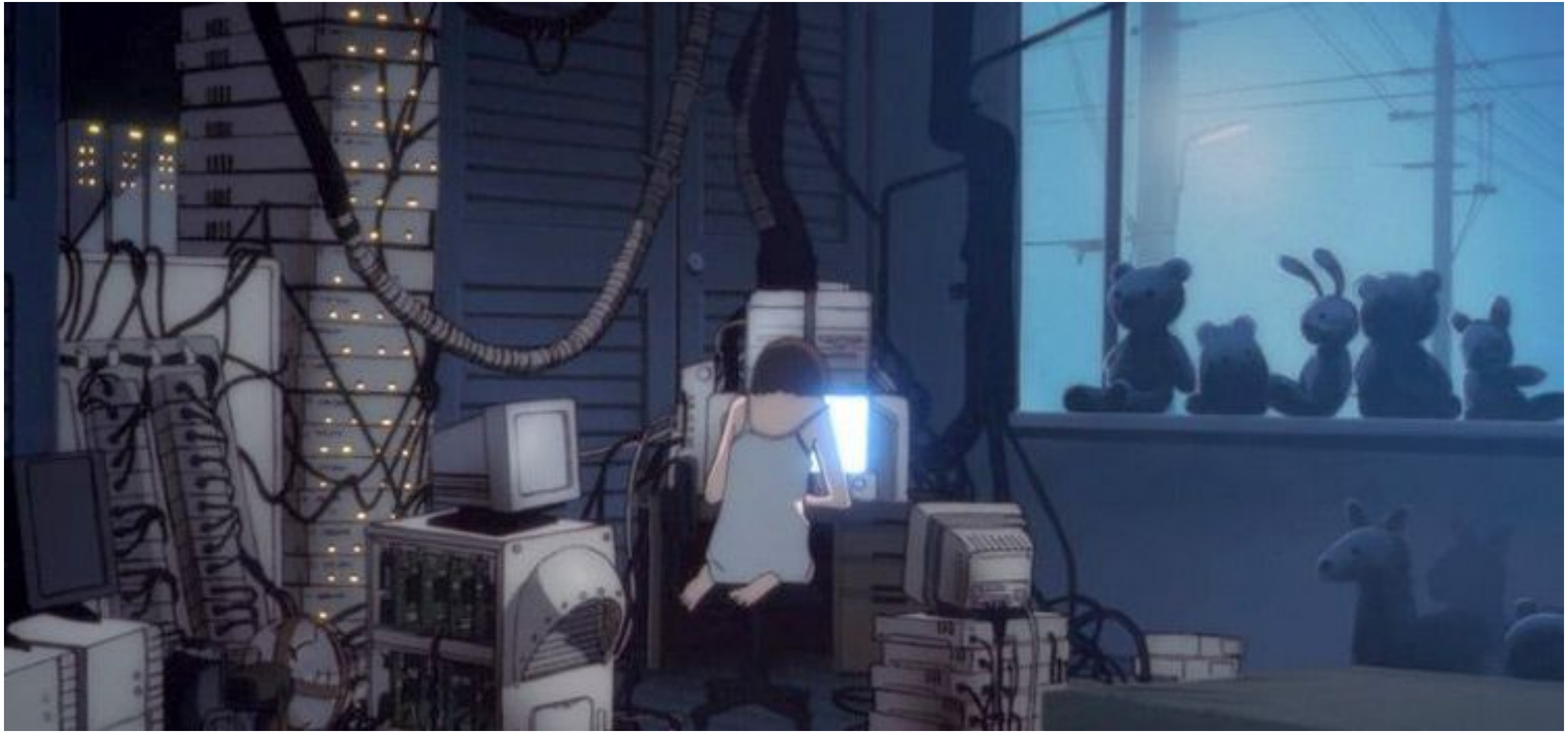
Stack canaries

RELRO

ASAN

Today we will see how to bypass them!

# Stack canary



# Stack canary (history)

THE HISTORY CHANNEL

Canaries: used in coal mines to detect deadly gas leaks (e.g., carbon monoxide)

In case of leak: canaries die quickly, humans have enough time to evacuate

Stack canaries: place before return address on the stack
⇒ stack buffer overflow overwrites canary
⇒ canary integrity checks before return cause early abort

# Stack canary (history)

THE HISTORY CHANNEL

**BYPASS**:
Killing a canary each time was cruel (and expensive), so at some point, miners invented:

***"The canary resuscitator"***

Small sealed chamber hooked up to a tank full of oxygen
⇒ put canary in the chamber as soon as it blacks out
⇒ canary survives (most of the time)

Stack canary bypass: overwrite the canary with the correct value
⇒ pwned executable thinks our canary is "still alive"

# Stack canary (stack cookie, stack guard, …)

Random value, unique per process

Protected functions place it in between local variables and the return address

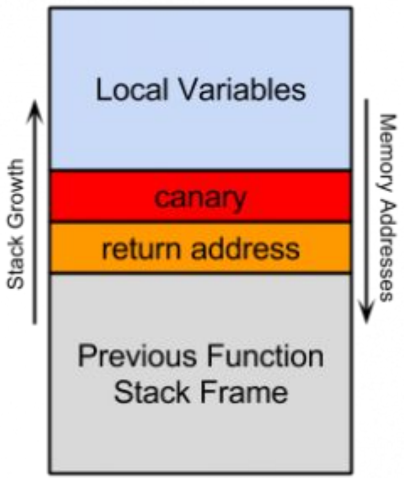
Before every ret instruction, the stack canary is checked to be intact

**BYPASS:**

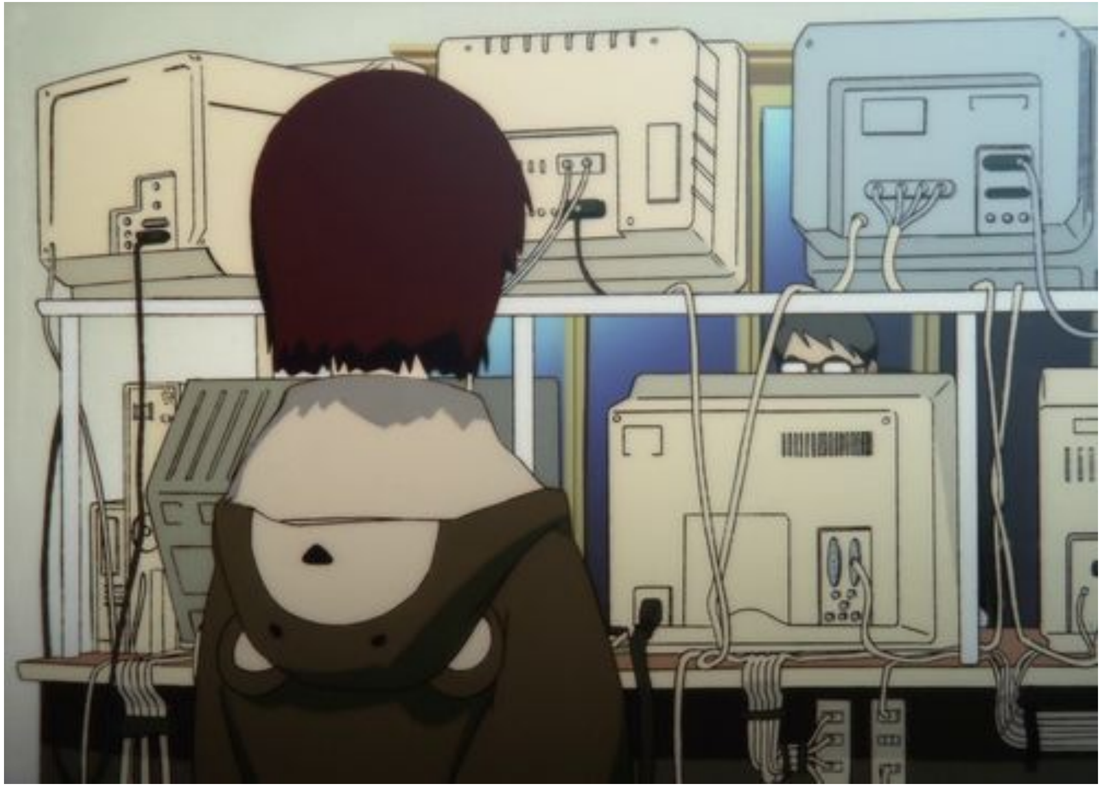
- Leak it with another vulnerability
- Restore the correct value when overwriting stack

**BYPASS:**

- Bruteforce it in case a crash does not kill the program
- Bruteforce byte-by-byte, trying to overwrite only the first byte of the canary, checking for canary murder (e.g., in a forking server)



# NX / W^X / DEP



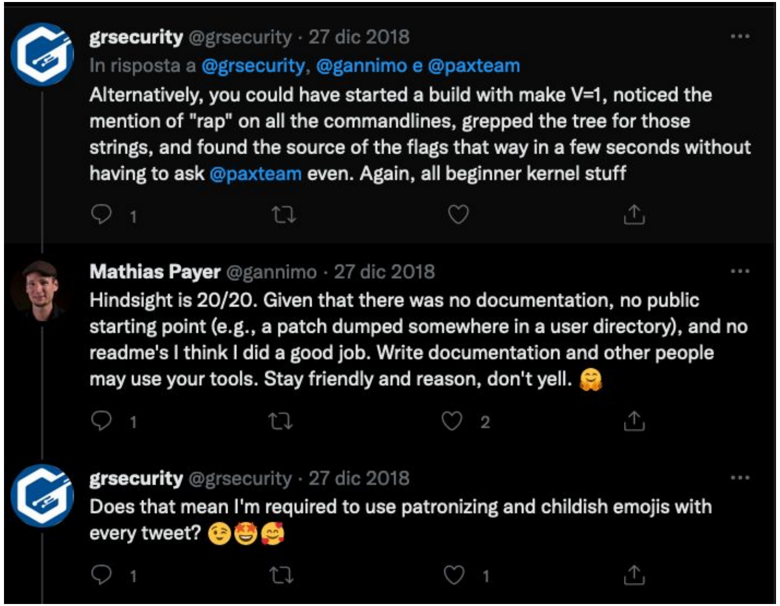
# NX (history)

Originally part of the PaX security patches to the Linux kernel maintained by grsecurity

PaX patches not merged into mainline Linux due to "unnecessary" features "slowing down the kernel"

From 2017 onward: open-source ⇒ closed-source, sold as proprietary blobs

In consequence: constant drama! :)
(and some stuff actually being implemented in the kernel nowadays)



grsecurity @grsecurity · 27 dic 2018
In risposta a @grsecurity, @gannimo e @paxteam
Alternatively, you could have started a build with make V=1, noticed the mention of "rap" on all the commandlines, grepped the tree for those strings, and found the source of the flags that way in a few seconds without having to ask @paxteam even. Again, all beginner kernel stuff

1

Mathias Payer @gannimo · 27 dic 2018
Hindsight is 20/20. Given that there was no documentation, no public starting point (e.g., a patch dumped somewhere in a user directory), and no readme's I think I did a good job. Write documentation and other people may use your tools. Stay friendly and reason, don't yell. 🤗

1 2

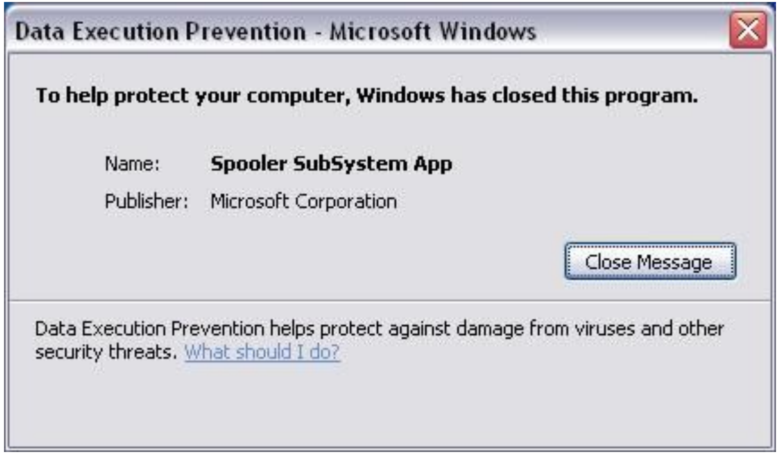
grsecurity @grsecurity · 27 dic 2018
Does that mean I'm required to use patronizing and childish emojis with every tweet? 😉🤩🥰

1 1

# NX

NX = **n**o e**x**ecute, W^X = **w**rite xor **e**xecute, DEP = **d**ata **e**xecution **p**revention
⇒ no simultaneously writable and executable memory

No more “shellcode on stack → jmp rsp → shell” :(



To help protect your computer, Windows has closed this program.

Name: Spooler SubSystem App
Publisher: Microsoft Corporation

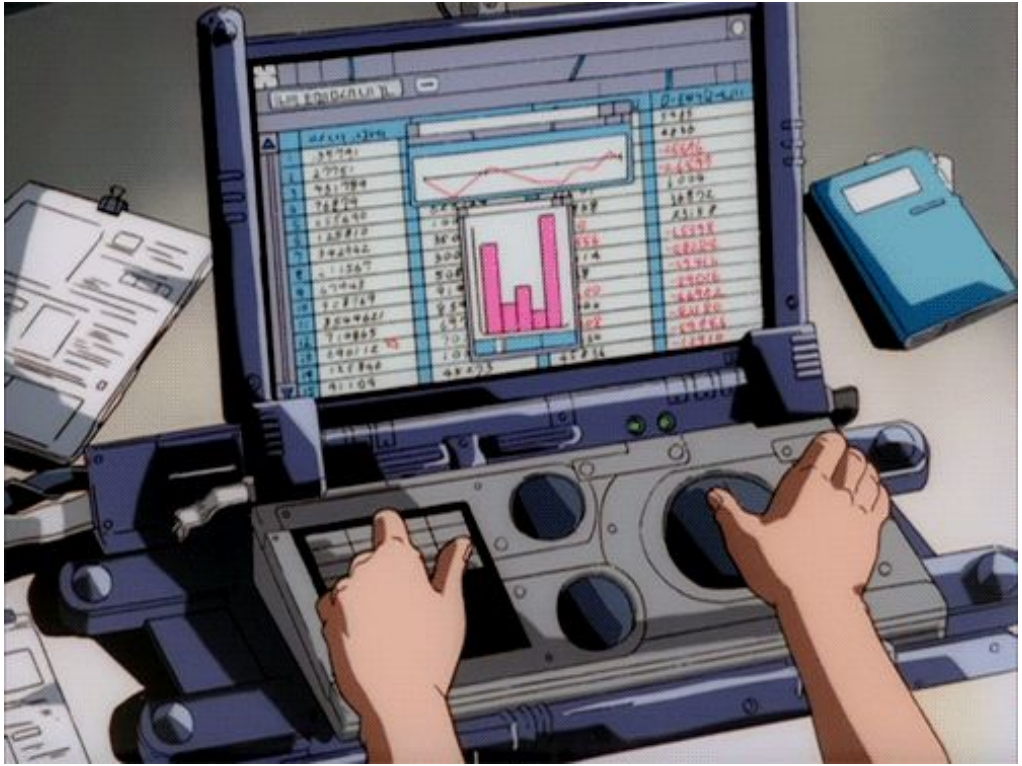
Data Execution Prevention helps protect against damage from viruses and other security threats. What should I do?

**BYPASS:**
Code Reuse:

- ROP (Return Oriented Programming)
- JOP (Jump Oriented Programming)
- SROP (Sig-return Oriented Programming)
- BOC (Basic-block Oriented Programming)

**BYPASS:**
Abuse the JIT (e.g., RWX pages for javascript execution in browsers)

# ASLR



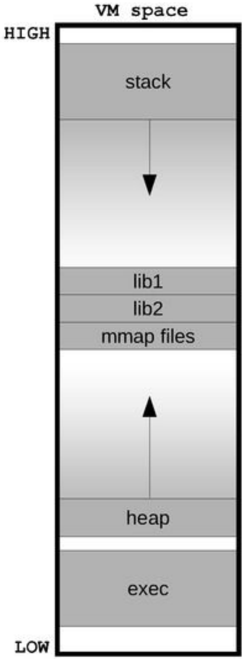
# ASLR (history)

THE HISTORY CHANNEL

Introduced by grsecurity in 2001, officially merged into the Linux kernel in 2005

Temporarily turn it off:

- `sysctl -a “kernel.randomize_va_space = 0”` for the whole system
- `setarch -R <binary> <arguments>` for a single invocation



Kernel ASLR added to Linux in 2014
⇒ kernel addresses randomized in addition to userspace
⇒ only 6 bits of randomness, easily bruteforced

# ASLR

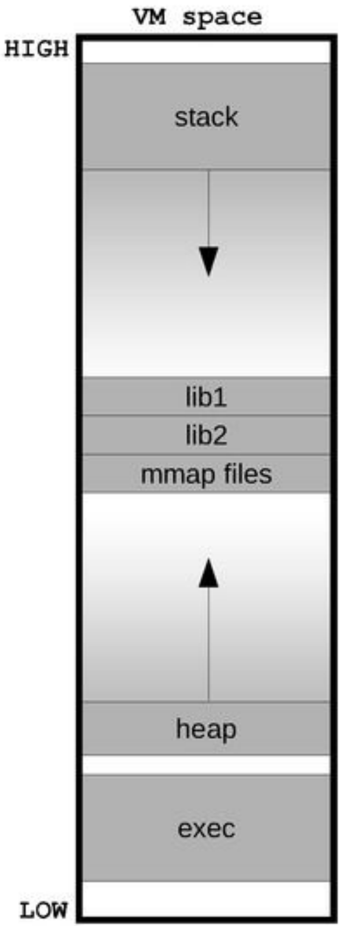
Base address of mapped libraries, stack, heap, ... randomized on each process execution

PIE != ASLR!!!
⇒ binary base address not necessarily randomized

Not included in checksec: OS feature, not encoded in the binary

Makes code reuse (ret2libc, ROP, ...) harder

**BYPASS:**
Leak the ASLR base address with another vulnerability (e.g., libc address leak, stack address leak)



# ASAN

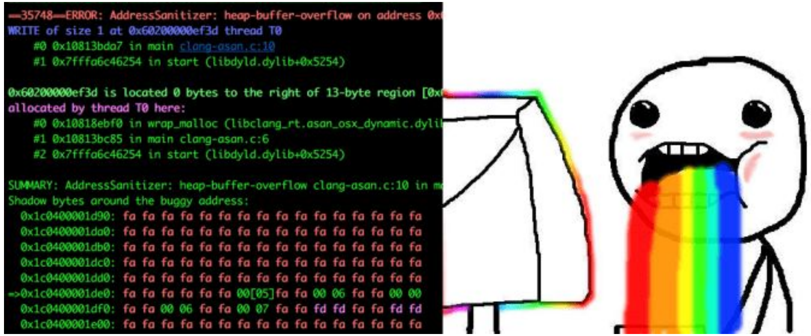
# ASAN (history)

AddressSanitizer (aka ASan) developed by Google as LLVM pass in 2014, later ported to GCC

Can be enabled with "-fsanitize=address" at compile time
⇒ incurs ~2x execution speed slowdown

Additional sanitizers:

- LeakSanitizer (memory leaks)
- ThreadSanitizer (data races and deadlocks)
- MemorySanitizer (uninitialized memory)
- HWASAN, or Hardware-assisted AddressSanitizer
- UBSan, or UndefinedBehaviorSanitizer



```
==35748==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x
WRITE of size 1 at 0x60200000ef3d thread T0
    #0 0x10813bda7 in main clang-asan.c:10
    #1 0x7fffa6c46254 in start (libdyld.dylib+0x5254)

0x60200000ef3d is located 0 bytes to the right of 13-byte region [0x
allocated by thread T0 here:
    #0 0x10818ebf0 in wrap_malloc (libclang_rt.asan_osx_dynamic.dyli
    #1 0x10813bc85 in main clang-asan.c:6
    #2 0x7fffa6c46254 in start (libdyld.dylib+0x5254)

SUMMARY: AddressSanitizer: heap-buffer-overflow clang-asan.c:10 in m
Shadow bytes around the buggy address:
  0x1c0400001d90: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x1c0400001da0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x1c0400001db0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x1c0400001dc0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x1c0400001dd0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x1c0400001de0: fa fa fa fa fa fa 00[05]fa fa 00 06 fa fa 00 00
  0x1c0400001df0: fa fa 00 06 fa fa 00 07 fa fa fd fd fa fa fd fd
  0x1c0400001e00: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
```

# ASAN

Keeps track of access permissions in "shadow memory"
⇒ 1 byte of shadow memory for 8 bytes of used memory

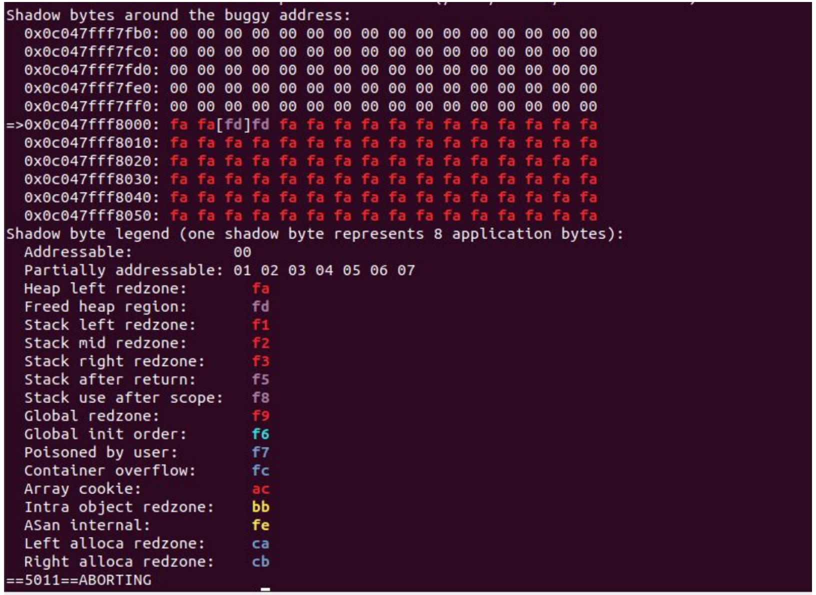
Bits in shadow memory encode whether bytes in program memory should be accessible

Code is instrumented to add "red zones" (non-accessible zones) around stack buffers

malloc and free are instrumented to add red zones around heap allocations

Every memory read or write first checks shadow memory to see whether access is allowed

**BYPASS:**
Controlled writes: do not overflow linearly but "skip" red zones



```
Shadow bytes around the buggy address:
  0x0c047fff7fb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7fc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0c047fff8000: fa fa[fd]fd fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8010: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8020: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8030: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8040: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8050: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
==5011==ABORTING
```

https://github.com/google/sanitizers/wiki/AddressSanitizerAlgorithm

# Final Word on the CTF

Deadline extended by a week to have one more exercise session for questions

Heap playground and heap meanu were borked $\Rightarrow$ fixed now :)

If you encounter bugs/problems: let us know!!!