25
Nov 2024

Shadow Stack

(or how to)

fix memory corruptions
immune to stack-protector

Bartosz Moczulski
https://bartosz.codes/

conference
code::dive

Agenda, or bugfixing 101

Encounter

Embrace

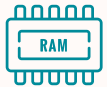
Eliminate

How many

of you had to deal with
memory corruption errors?

So have I

Memory



heap
stack

MT

deadlocks
race conditions

Logic

resource leakage
if ((π = 3) == e) { … }

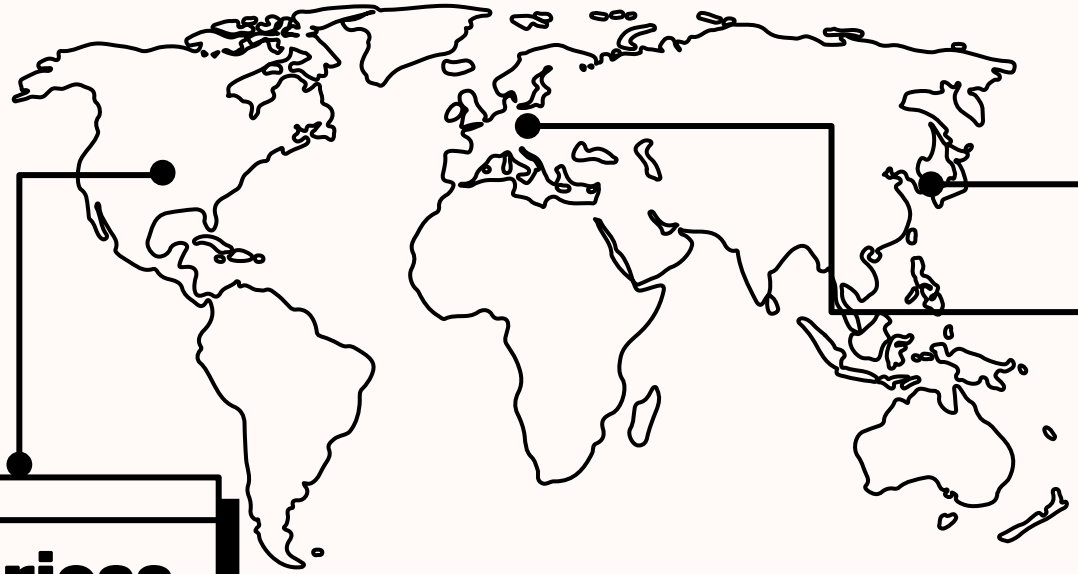
Integration

20+

years

Detectivism

I was (involved) here



Asia

South Korea

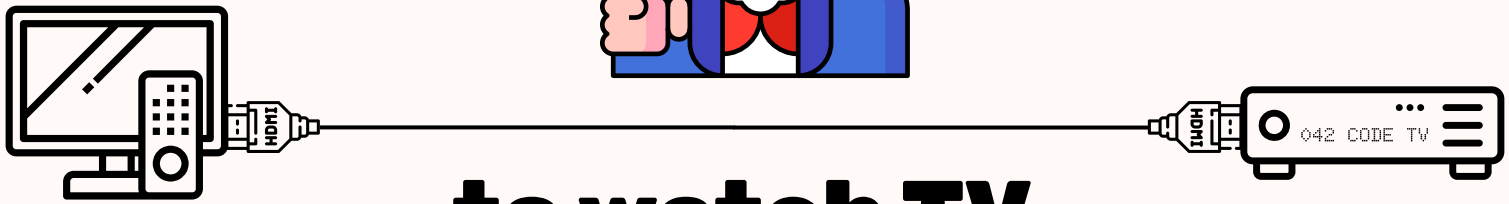
Europe

Poland, Czechia, Germany, Austria, Netherlands, Belgium, Sweden, Norway, Italy, UK, Spain, ...

Americas

USA, (Mexico, Chile)

I want you



to watch TV

and streaming services

Which stack?

This one

- Memory for local variables (and more)
- ISA-supported
- ABI, calling convention
- C, C++, other languages

Not that one

- std::stack<T>

Your takeaways

Amusement

- moved to Rust already?
- enjoy peer misery 😈

Awareness

- stack is tricky
- beware!

Apprehension

- see the evidence
- we can fix it!
- Shadow Stack can help

The Encounter

The Chat

The Chat

=

The Chat

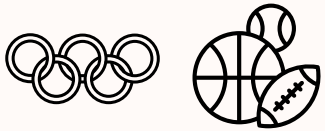
The Chat

+

=

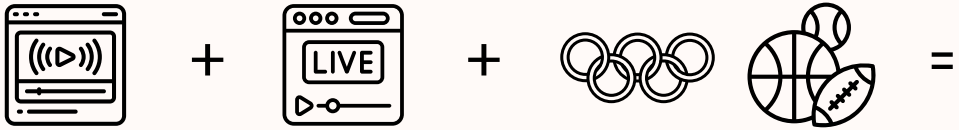
1/day

The Chat

 +  +  =

=  1/day

The Chat



=  1/day + 

16

The Embrace

GDB coredump analysis

```
SIGSEGV in Thread "video_stream"

(gdb) bt
#0 pthread_mutex_unlock (m=0x00ffffcc03)
#1 do_stuff (s=0xffffffcc00)
…
#42 start_thread (arg=...)
```

Anything wrong here?

The C/C++ code

The type

```
struct S {
    void *p;
    pthread_mutex_t mutex;
    // more fields ...
};
```

The function

```
void do_stuff(S *s) {
    pthread_mutex_lock(&s->mutex);
    do_stuff_locked(s);
    pthread_mutex_unlock(&s->mutex);
}
```

Function disassembly

```
(gdb) disassemble do_stuff
Dump of assembler code for function do_stuff:
    stp   x29, x30, [sp, #-32]!
    mov   x29, sp
    stp   x19, x20, [sp, #16]
    add   x20, x0, #8
    mov   x19, x0
    mov   x0, x20
    bl    0xfffff7f1ce20 <pthread_mutex_lock@plt>
    mov   x0, x19
    bl    0xfffff7f1cef0 <do_stuff_locked@plt>
    mov   x0, x20
    bl    0xfffff7f1ce80 <pthread_mutex_unlock@plt>
    ldp   x19, x20, [sp, #16]
    ldp   x29, x30, [sp], #32
    ret
End of assembler dump.
```

ARM matters

How many

of you have a cellphone
in your pocket right now?

ARM 64 calling convention

args + ret(s)

x0-x7
input arguments
returned value(s)

MAY change

x9-x15
subroutine may modify them (caller-saved)

MUST preserve

x19-x28
subroutine MUST preserve them (callee-saved)

subroutines

x30 (LR) - link register
x29 (FP) - frame pointer

stack

x31 (SP)
stack pointer

other

x8, x16, x17, x18
irrelevant for this talk

Source: https://en.wikipedia.org/wiki/Calling_convention#ARM_(A64)

Function disassembly

```
(gdb) disassemble do_stuff
Dump of assembler code for function do_stuff:
    stp   x29, x30, [sp, #-32]!
    mov   x29, sp
    stp   x19, x20, [sp, #16]
    add   x20, x0, #8
    mov   x19, x0
    mov   x0, x20
    bl    0xfffff7f1ce20 <pthread_mutex_lock@plt>
    mov   x0, x19
    bl    0xfffff7f1cef0 <do_stuff_locked@plt>
    mov   x0, x20
    bl    0xfffff7f1ce80 <pthread_mutex_unlock@plt>
    ldp   x19, x20, [sp, #16]
    ldp   x29, x30, [sp], #32
    ret
End of assembler dump.
```

Subroutine boilerplate:

- save LR & FP on stack
- reserve necessary stack space

- revert the above and return

Function disassembly

```
(gdb) disassemble do_stuff
Dump of assembler code for function do_stuff:
    stp   x29, x30, [sp, #-32]!
    mov   x29, sp
    stp   x19, x20, [sp, #16]
    add   x20, x0, #8
    mov   x19, x0
    mov   x0, x20
    bl    0xfffff7f1ce20 <pthread_mutex_lock@plt>
    mov   x0, x19
    bl    0xfffff7f1cef0 <do_stuff_locked@plt>
    mov   x0, x20
    bl    0xfffff7f1ce80 <pthread_mutex_unlock@plt>
    ldp   x19, x20, [sp, #16]
    ldp   x29, x30, [sp], #32
    ret
End of assembler dump.
```

This function uses x19 and x20:

- save x19, x20 on stack

recall: MUST preserve, callee-saved

- restore them before returning (i.e. fulfill "MUST preserve" contract)

Function disassembly

```
(gdb) disassemble do_stuff
Dump of assembler code for function do_stuff:
   stp   x29, x30, [sp, #-32]!
   mov   x29, sp
   stp   x19, x20, [sp, #16]
   add   x20, x0, #8
   mov   x19, x0
   mov   x0, x20
   bl    0xfffff7f1ce20 <pthread_mutex_lock@plt>
   mov   x0, x19
   bl    0xfffff7f1cef0 <do_stuff_locked@plt>
   mov   x0, x20
   bl    0xfffff7f1ce80 <pthread_mutex_unlock@plt>
   ldp   x19, x20, [sp, #16]
   ldp   x29, x30, [sp], #32
   ret
End of assembler dump.
```

Calculate helper variables:

- x20 = address of the mutex (offset 8 into the structure)
- x19 = 1st arg (i.e. address of the structure)

recall: subroutines MUST preserve them

Function disassembly

```
(gdb) disassemble do_stuff
Dump of assembler code for function do_stuff:
    stp   x29, x30, [sp, #-32]!
    mov   x29, sp
    stp   x19, x20, [sp, #16]
    add   x20, x0, #8
    mov   x19, x0
    mov   x0, x20
    bl    0xfffff7f1ce20 <pthread_mutex_lock@plt>
    mov   x0, x19
    bl    0xfffff7f1cef0 <do_stuff_locked@plt>
    mov   x0, x20
    bl    0xfffff7f1ce80 <pthread_mutex_unlock@plt>
    ldp   x19, x20, [sp, #16]
    ldp   x29, x30, [sp], #32
    ret
End of assembler dump.
```

Lock the mutex:

- move 1st arg to x0
  (x20 holds mutex address)
- call pthread_mutex_lock()

Function disassembly

```
(gdb) disassemble do_stuff
Dump of assembler code for function do_stuff:
    stp   x29, x30, [sp, #-32]!
    mov   x29, sp
    stp   x19, x20, [sp, #16]
    add   x20, x0, #8
    mov   x19, x0
    mov   x0, x20
    bl    0xfffff7f1ce20 <pthread_mutex_lock@plt>
    mov   x0, x19
    bl    0xfffff7f1cef0 <do_stuff_locked@plt>
    mov   x0, x20
    bl    0xfffff7f1ce80 <pthread_mutex_unlock@plt>
    ldp   x19, x20, [sp, #16]
    ldp   x29, x30, [sp], #32
    ret
End of assembler dump.
```

Likewise:

- move 1st arg to x0
  (x19 holds structure address)
- call do_stuff_locked()

Function disassembly

```
(gdb) disassemble do_stuff
Dump of assembler code for function do_stuff:
   stp   x29, x30, [sp, #-32]!
   mov   x29, sp
   stp   x19, x20, [sp, #16]
   add   x20, x0, #8
   mov   x19, x0
   mov   x0, x20
   bl    0xfffff7f1ce20 <pthread_mutex_lock@plt>
   mov   x0, x19
   bl    0xfffff7f1cef0 <do_stuff_locked@plt>
   mov   x0, x20
   bl    0xfffff7f1ce80 <pthread_mutex_unlock@plt>
   ldp   x19, x20, [sp, #16]
   ldp   x29, x30, [sp], #32
   ret
End of assembler dump.
```

Unlock the mutex:

- move 1st arg to x0
  (x20 holds mutex address)
- call pthread_mutex_unlock()
- **BOOM!**

Function disassembly

```
(gdb) disassemble do_stuff
Dump of assembler code for function do_stuff:
    stp   x29, x30, [sp, #-32]!
    mov   x29, sp
    stp   x19, x20, [sp, #16]
    add   x20, x0, #8
    mov   x19, x0
    mov   x0, x20
    bl    0xfffff7f1ce20 <pthread_mutex_lock@plt>
    mov   x0, x19
    bl    0xfffff7f1cef0 <do_stuff_locked@plt>
    mov   x0, x20
    bl    0xfffff7f1ce80 <pthread_mutex_unlock@plt>
    ldp   x19, x20, [sp, #16]
    ldp   x29, x30, [sp], #32
    ret
End of assembler dump.
```

Here x20 was correct 0x**ffffffcc08**

Here x20 is **broken** 0x**00**ffffcc**03**
Although ABI mandates that
it MUST not change.

Function disassembly

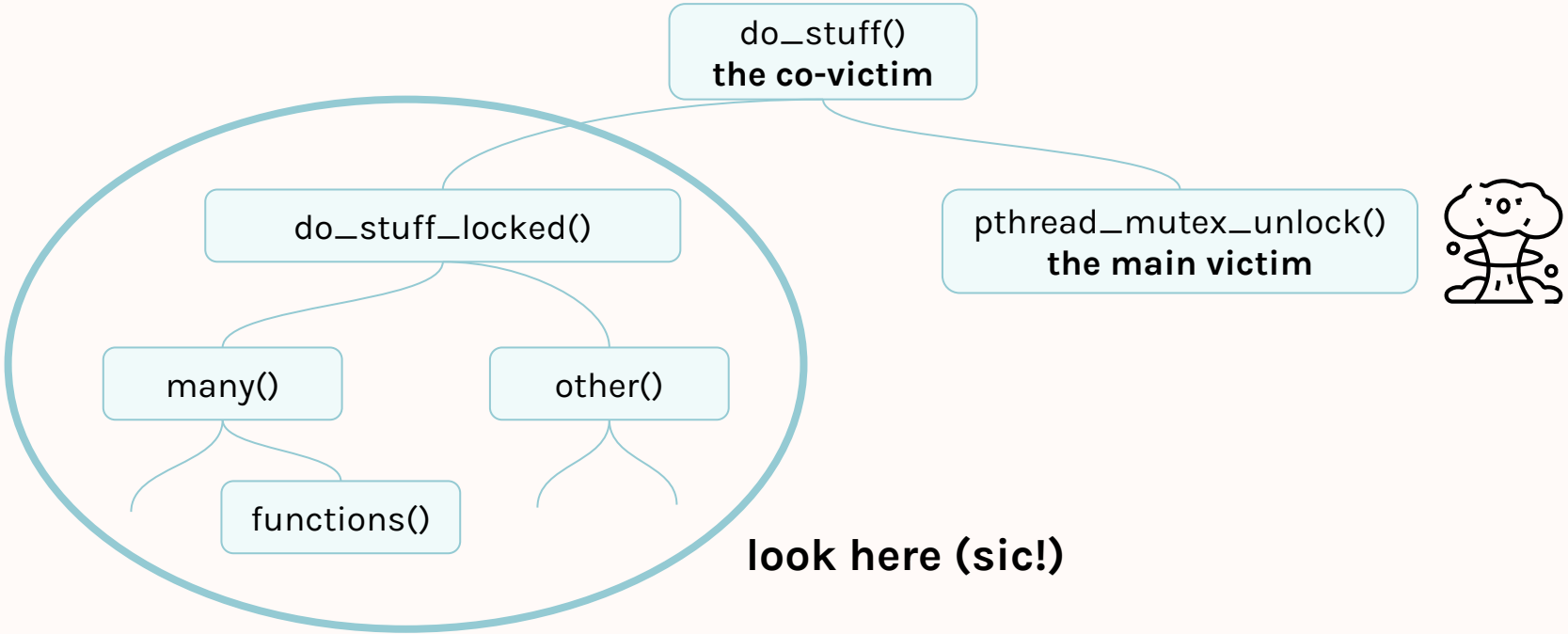
```
(gdb) disassemble do_stuff
Dump of assembler code for function do_stuff:
   stp   x29, x30, [sp, #-32]!
   mov   x29, sp
   stp   x19, x20, [sp, #16]
   add   x20, x0, #8
   mov   x19, x0
   mov   x0, x20
   bl    0xfffff7f1ce20 <pthread_mutex_lock@plt>
   mov   x0, x19
   bl    0xfffff7f1cef0 <do_stuff_locked@plt>
   mov   x0, x20
   bl    0xfffff7f1ce80 <pthread_mutex_unlock@plt>
   ldp   x19, x20, [sp, #16]
   ldp   x29, x30, [sp], #32
   ret
End of assembler dump.
```

The **suspect** is do_stuff_locked() and all the function it calls (**recursively!**).

(sub) Function disassembly

```
(gdb) disassemble do_stuff_locked
Dump of assembler code for function do_stuff_locked:
    stp   x29, x30, [sp, #-32]!
    mov   x29, sp

    stp   x19, x20, [sp, #16]        // push x20 == 0xffffffcc08

    ... do the actual stuff (with do_stuff's x19 and x20 temporarily on stack!)

    ldp   x19, x20, [sp, #16]        // pop  x20 == 0x00ffffcc03

    ldp   x29, x30, [sp], #32
    ret
End of assembler dump.
```

The problem



look here (sic!)

How come?

Don’t suspect:

- Compiler (function calls, etc.)
- other threads
- cosmic rays...

Do suspect:

- out-of-bound write
- stack order (down-growth)

Stack memory map

free stack space

How come?

Call trace:

- do_stuff()

Stack memory map

free stack space

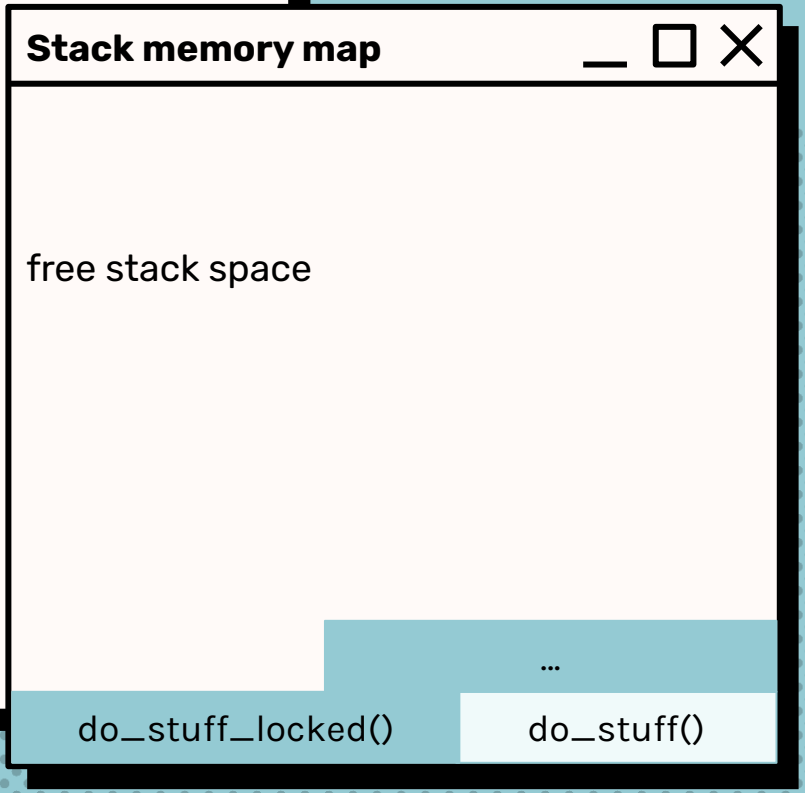
do_stuff()

How come?

Call trace:

- do_stuff()
- do_stuff_locked()

Stack memory map

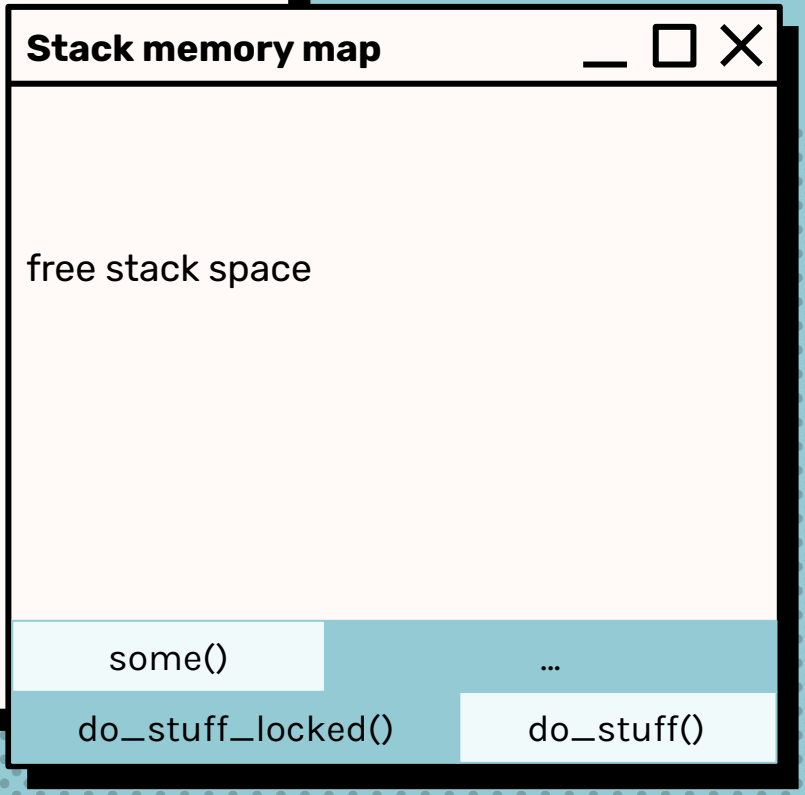


How come?

Call trace:

- do_stuff()
- do_stuff_locked()
- some()

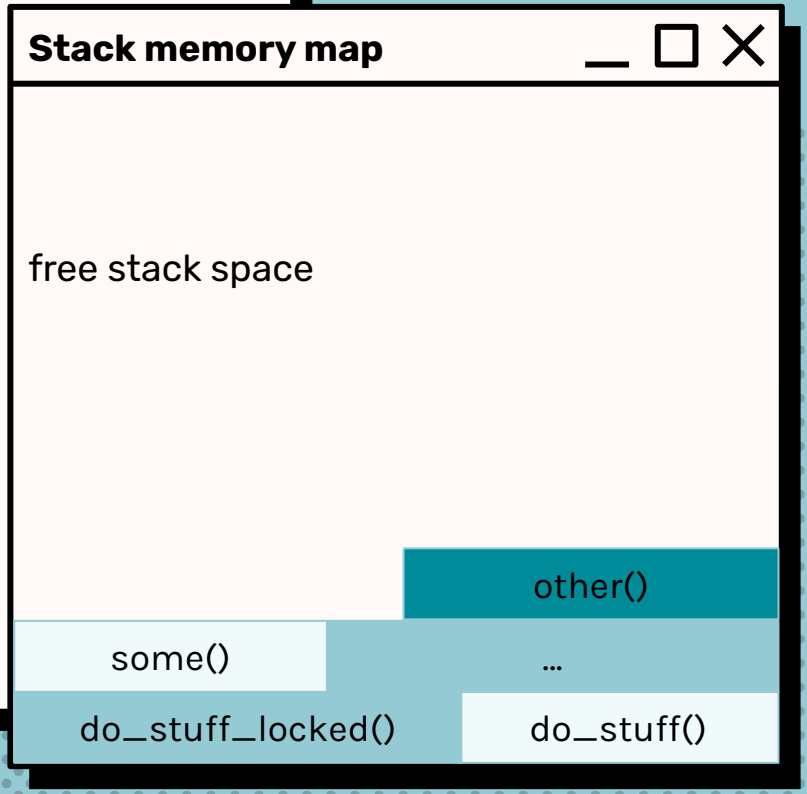
Stack memory map



How come?

Call trace:

- do_stuff()
- do_stuff_locked()
- some()
- other()

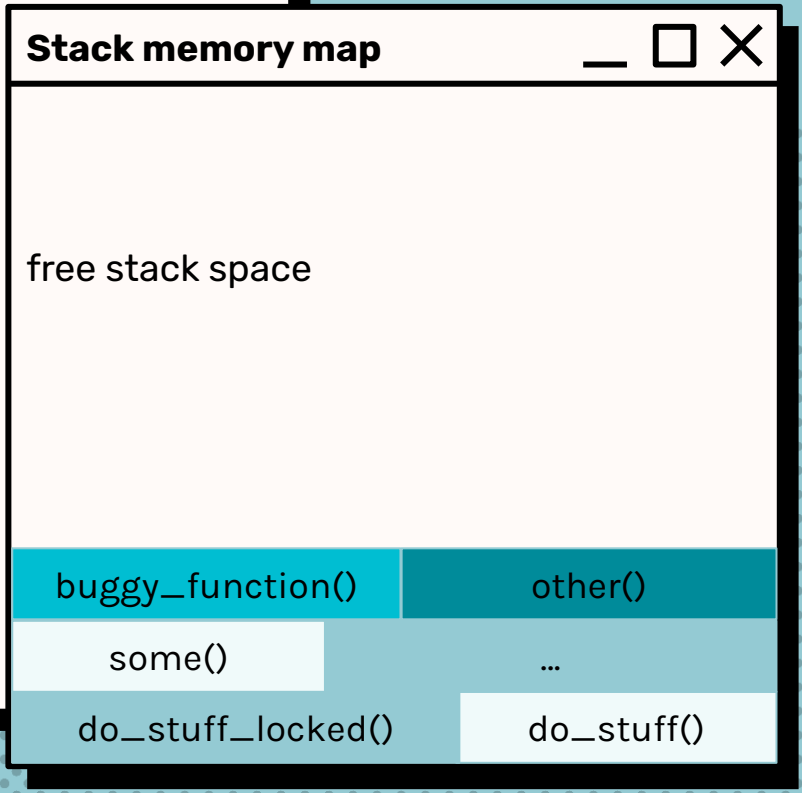


How come?

Call trace:

- do_stuff()
- do_stuff_locked()
- some()
- other()
- buggy_function()

Stack memory map



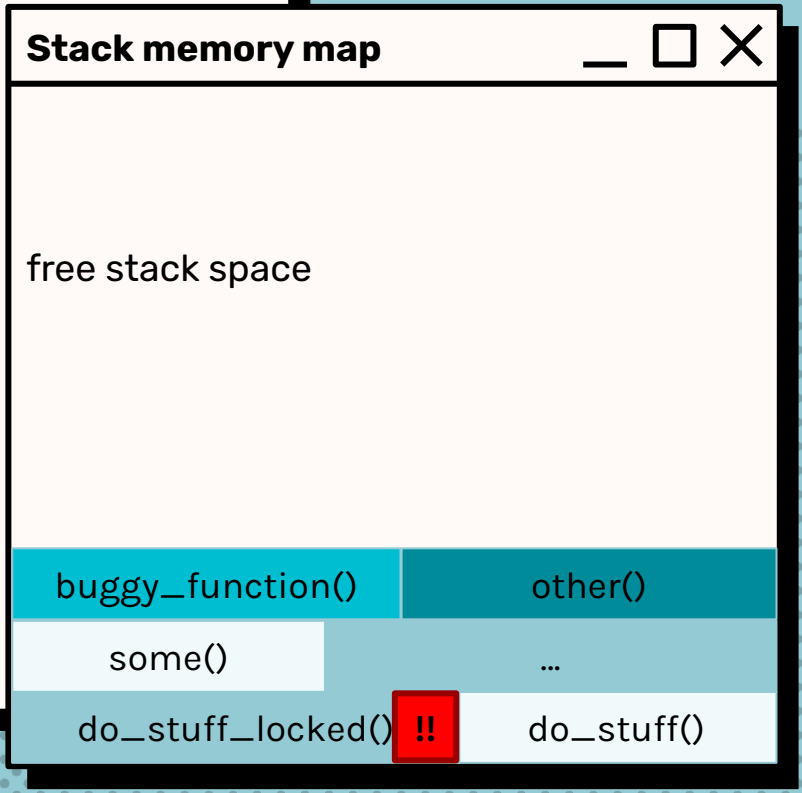
How come?

Call trace:

- do_stuff()
- do_stuff_locked()
- some()
- other()
- buggy_function()

```
void buggy_function() {
    uint8_t bytes[8];
    bytes[666] = 0x03;    // BOOM!
}
```

Stack memory map



With stack-protector?

Is `-fstack-protector` a silver bullet?

Spoiler alert: **NO!**

With stack-protector?

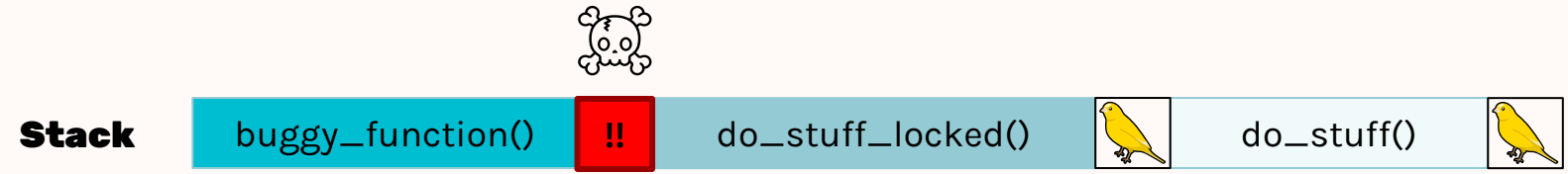
Stack protector:

- cushes constant **canary value** to stack before every function call
- checks **current** canary value before return
- NOTE: not all canaries, not entire stack!

With stack-protector?

Overwriting current canary:

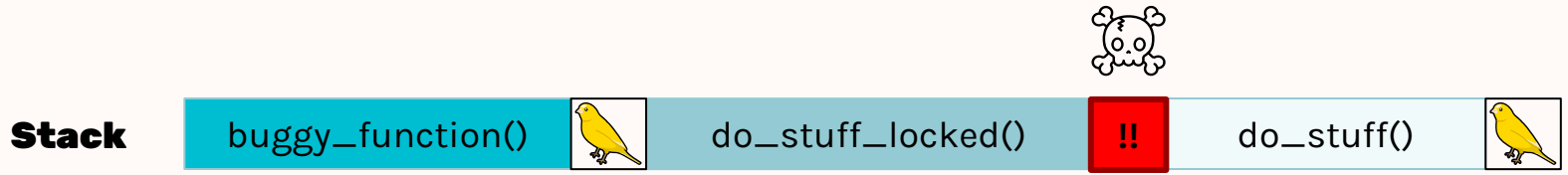
- detected upon return from current function
- i.e. deferred a bit
- many children might be called before that



With stack-protector?

Overwriting parent canary:

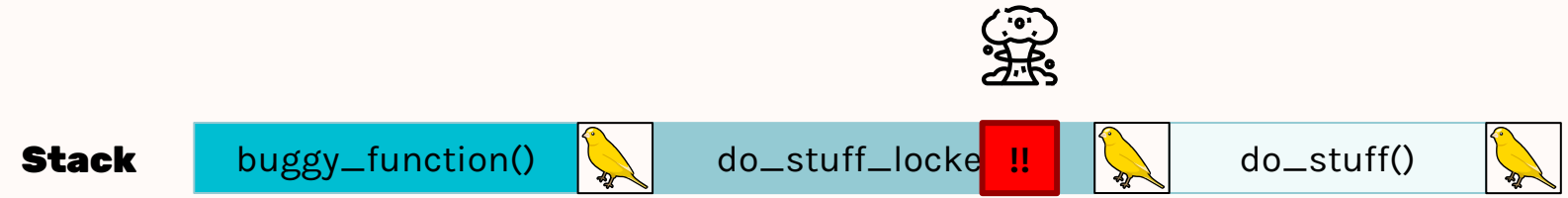
- detected upon return from parent function
- i.e. deferred substantially
- even siblings might be called before that



With stack-protector?

Overwriting other areas:

- not detected at all (sic!)



Is stack-protector any good?

What's the point of it then?

- contiguous buffer overwrites
- `memset(bytes, 0, sizeof(bytes) + 666);`

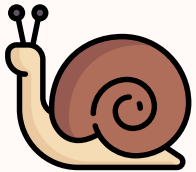
The Elimination

Do you know

any tools that could help
identify the corruption place?

valgrind?

NOPE
way too slow



Brainstormin
g

Adam Szaj

pronounce: /shy/
AS + C++ = ❤️
AS + metaprogramming = ❤️
AS + challenges = ❤️
"impossible" = 404

Brainstormin
g

GCC/Clang fork or plugin?

- custom stack protector
- could call my own handler at every function call and return

Adam Szaj

pronounce: /shy/
AS + C++ = ❤️
AS + metaprogramming = ❤️
AS + challenges = ❤️
"impossible" = 404

Brainstormin
g

GCC/Clang fork or plugin?

- custom stack protector
- could call my own handler at every function call and return

Hold my 🍺

Adam Szaj

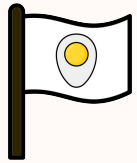
pronounce: /shy/
AS + C++ = ❤️
AS + metaprogramming = ❤️
AS + challenges = ❤️
"impossible" = 404

Behold!
Shadow Stack

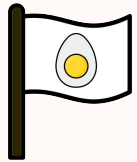
Name inspiration

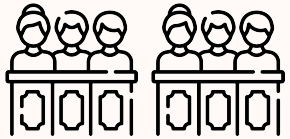
Shadow cabinet (politics) and shadow DOM (web)

Big-endians*



Little-endians opposition



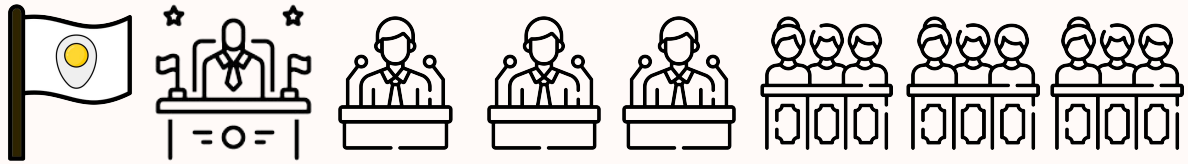


* reference to *Gulliver's Travels* by Jonathan Swift in byte-order names is no coincidence

Name inspiration

Shadow cabinet (politics) and shadow DOM (web)

Big-endians* government (ruling cabinet)



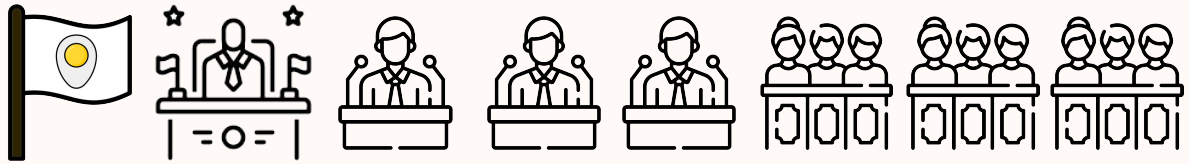
Little-endians opposition

* reference to *Gulliver's Travels* by Jonathan Swift in byte-order names is no coincidence

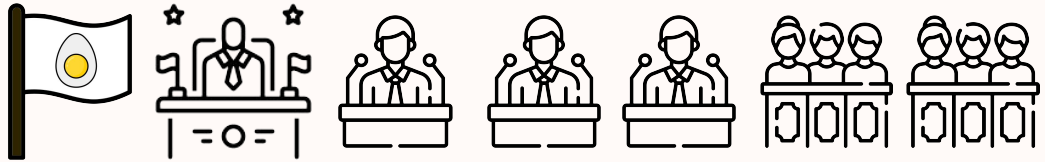
Name inspiration

Shadow cabinet (politics) and shadow DOM (web)

Big-endians* government (ruling cabinet)



Little-endians opposition (shadow cabinet)

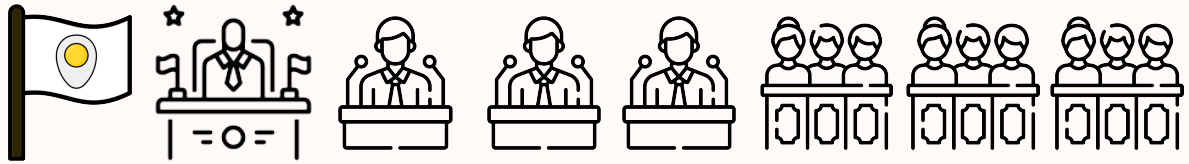


* reference to *Gulliver's Travels* by Jonathan Swift in byte-order names is no coincidence

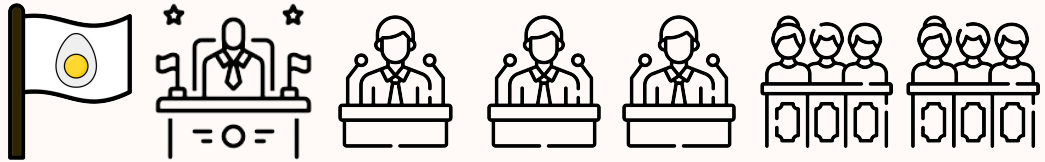
Name inspiration

Shadow cabinet (politics) and shadow DOM (web)

Big-endians* government (ruling cabinet)



Little-endians opposition (shadow cabinet)

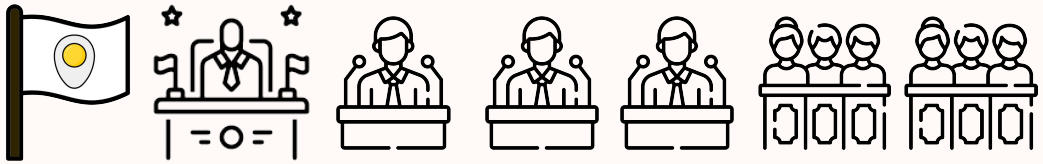


* reference to *Gulliver's Travels* by Jonathan Swift in byte-order names is no coincidence

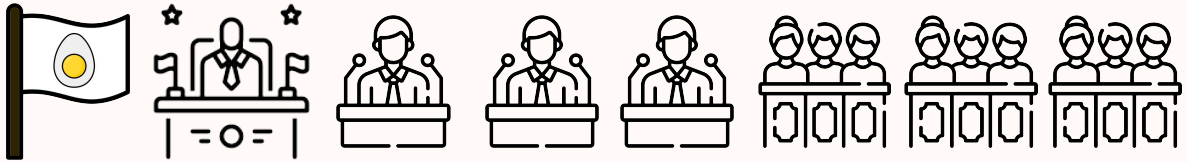
Name inspiration

Shadow cabinet (politics) and shadow DOM (web)

Big-endians* government (shadow cabinet)



Little-endians opposition (ruling cabinet)



* reference to *Gulliver's Travels* by Jonathan Swift in byte-order names is no coincidence

Pseudo-code

Pseudo-code

pre-call

```
// get current stack pointer
void *sp = &sp;

// append to shadow area
memcpy(shadow_end, sp, delta);
shadow_end += delta;
```

Pseudo-code

pre-call

```
// get current stack pointer
void *sp = &sp;

// append to shadow area
memcpy(shadow_end, sp, delta);
shadow_end += delta;
```

post-return

```
// shrink shadow area
shadow_end -= delta;
```

Pseudo-code

pre-call

```
// get current stack pointer
void *sp = &sp;

// append to shadow area
memcpy(shadow_end, sp, delta);
shadow_end += delta;
```

check twice

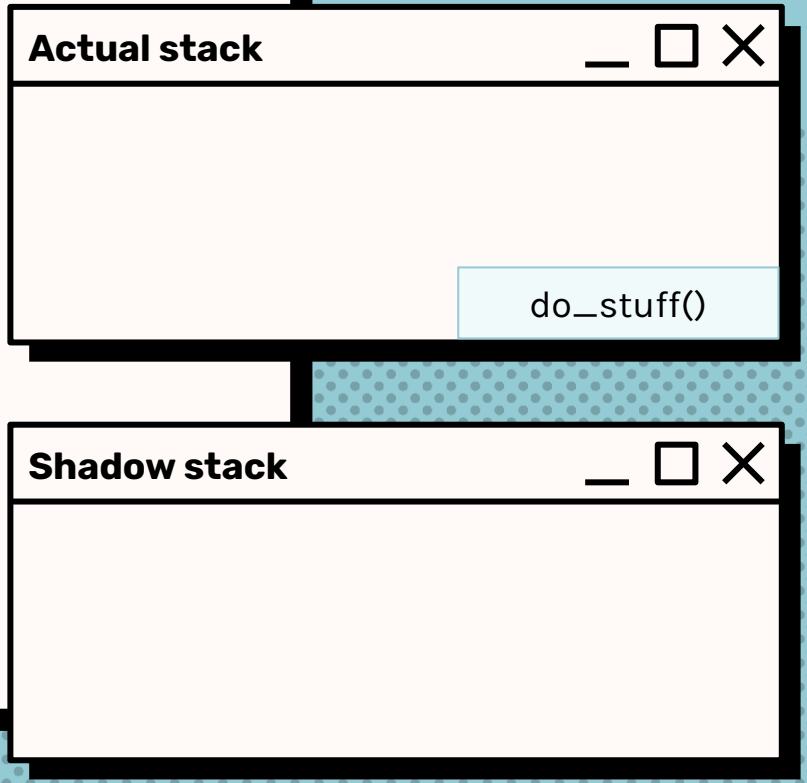
```
// compare entire shadow area
memcmp(shadow, actual, size);

// print backtrace upon error
backtrace(...);
```

post-return

```
// shrink shadow area
shadow_end -= delta;
```

In action

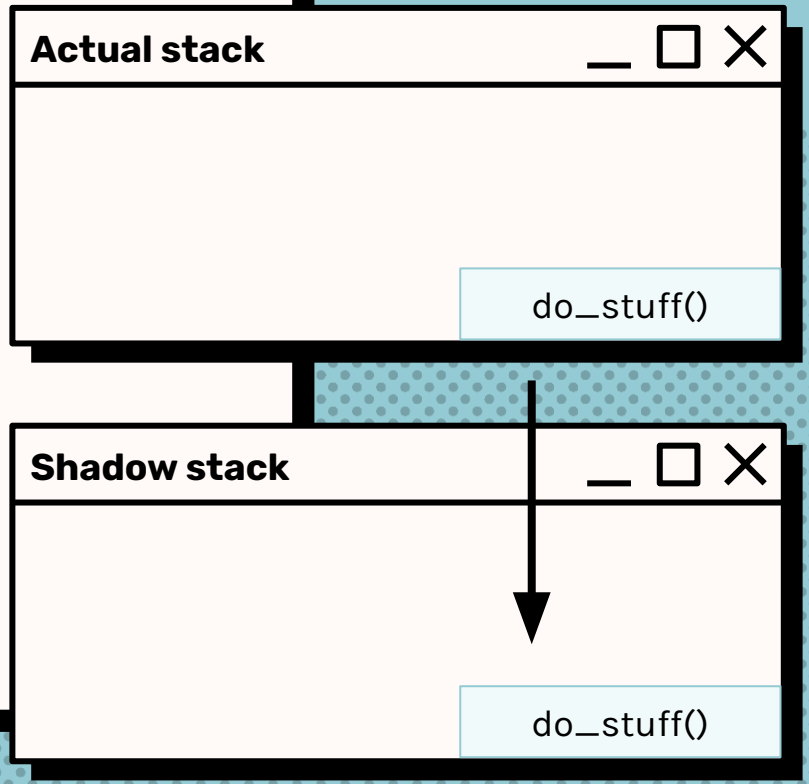
do_stuff()



In action

do_stuff()

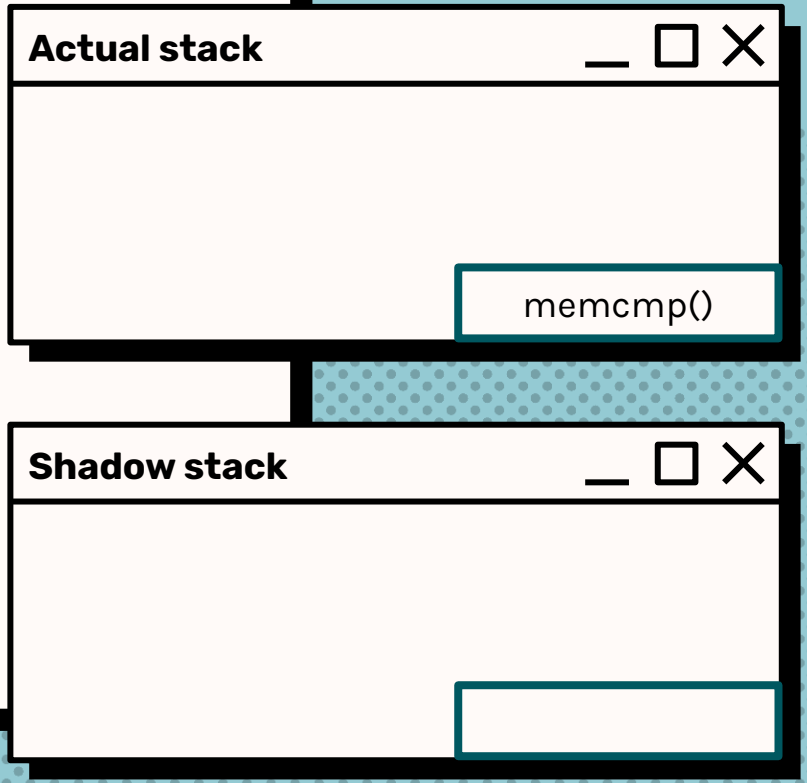
↳ PRE-CALL



In action

do_stuff()

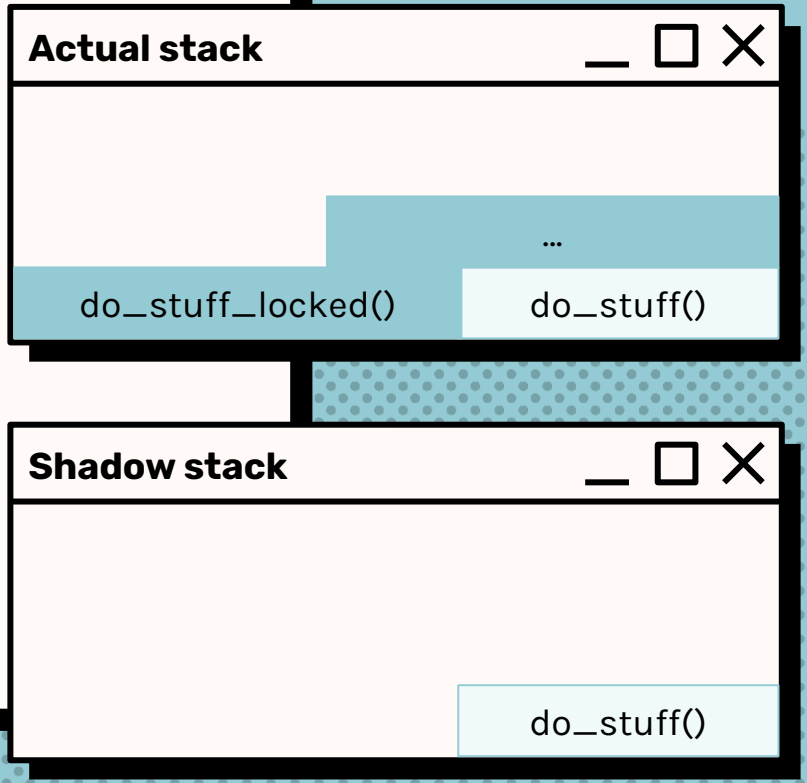
↳ PRE-CALL



In action

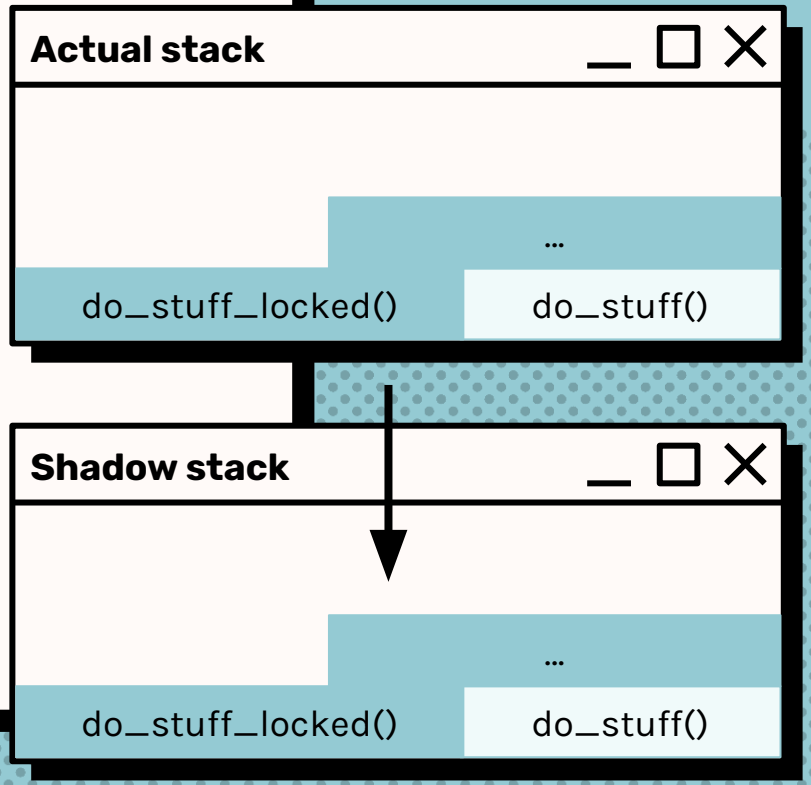
do_stuff()

↳ do_stuff_locked()



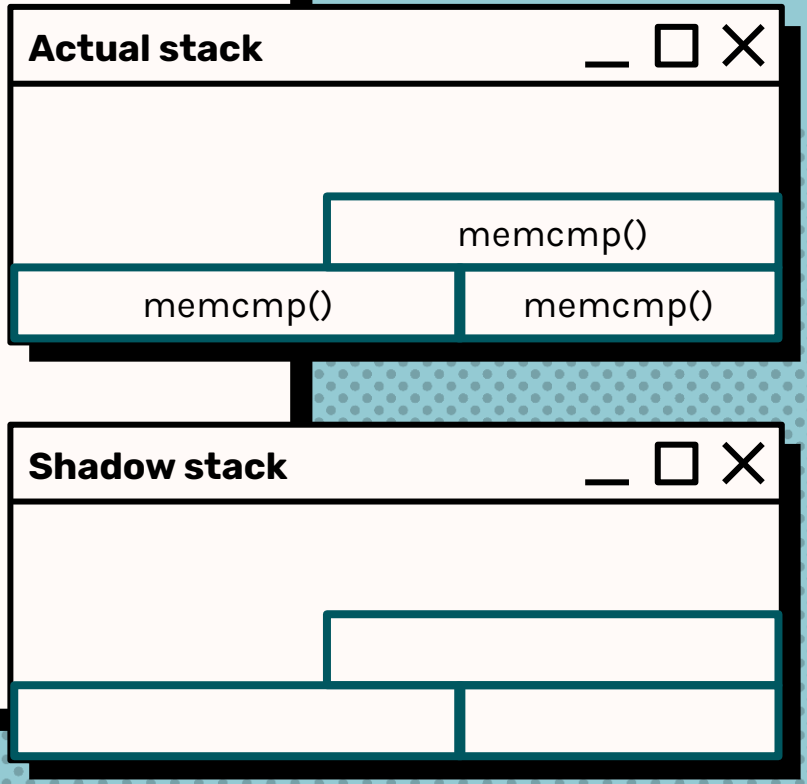
In action

do_stuff()
↳ do_stuff_locked()
↳ PRE-CALL



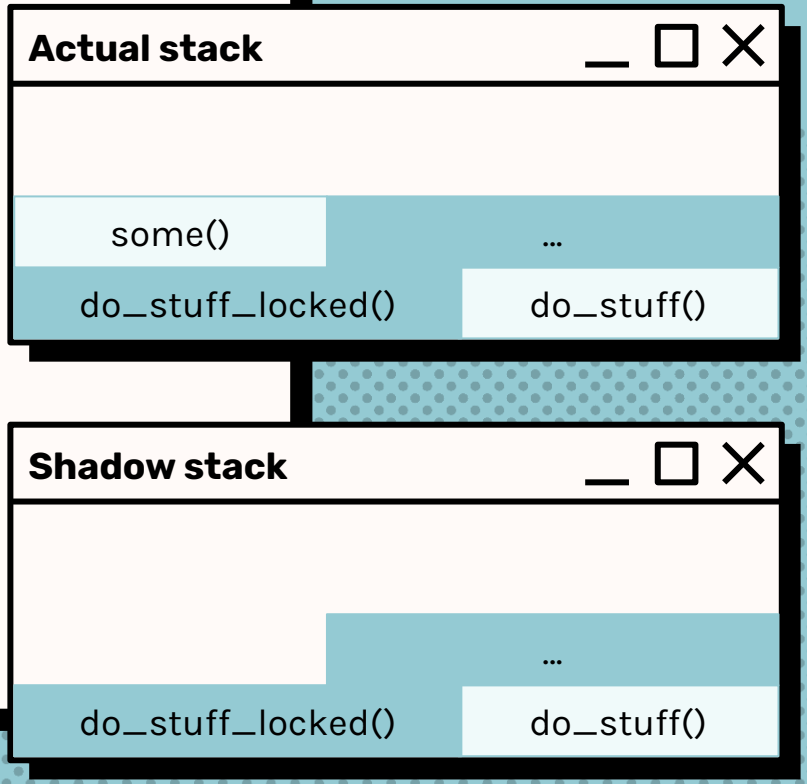
In action

do_stuff()
↳ do_stuff_locked()
↳ PRE-CALL



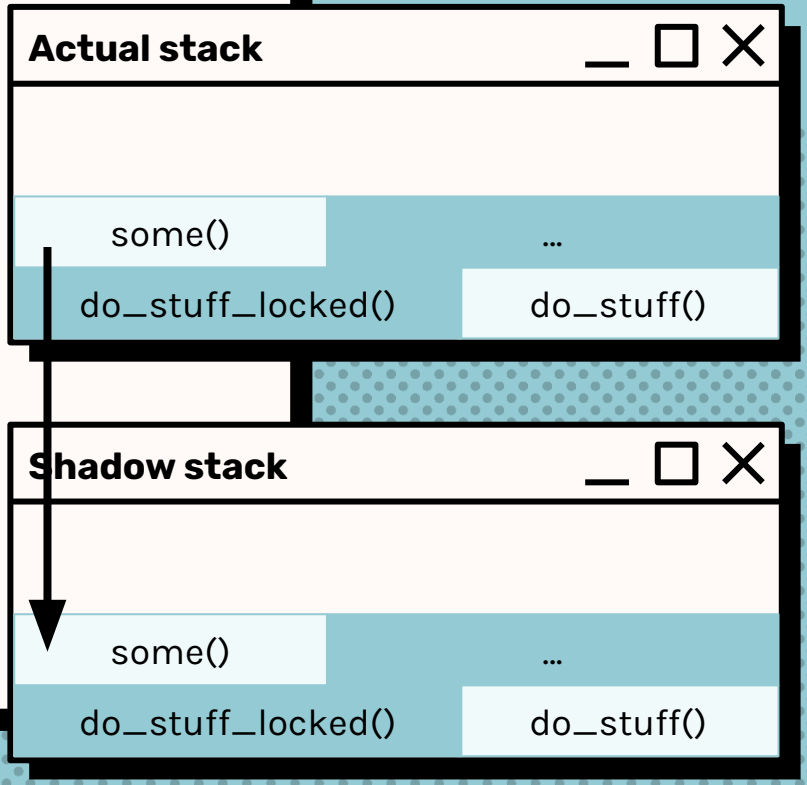
In action

do_stuff()
↳ do_stuff_locked()
↳ some()



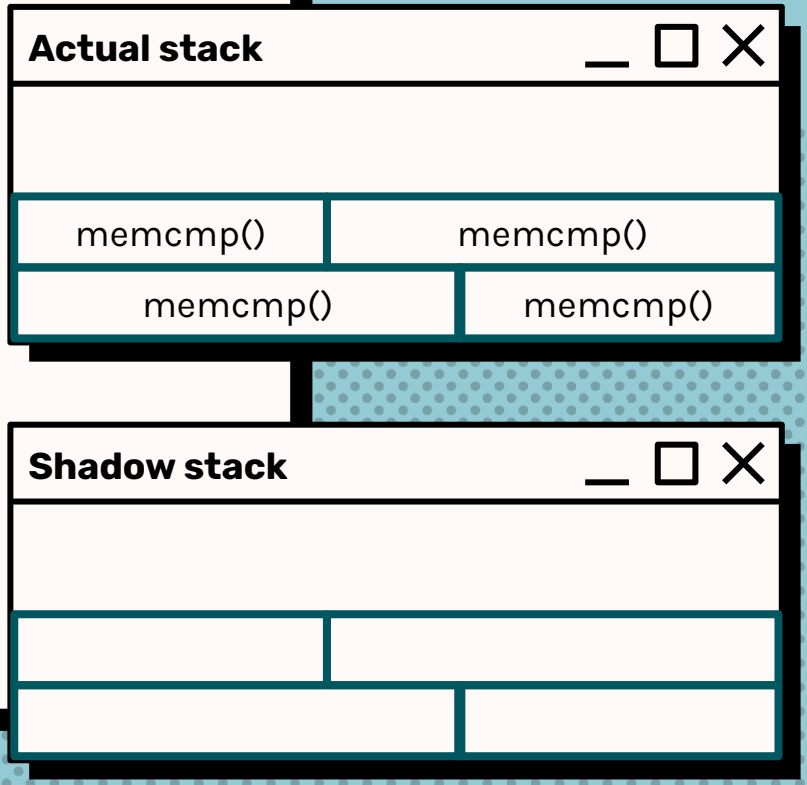
In action

do_stuff()
↳ do_stuff_locked()
↳ some()
↳ PRE-CALL



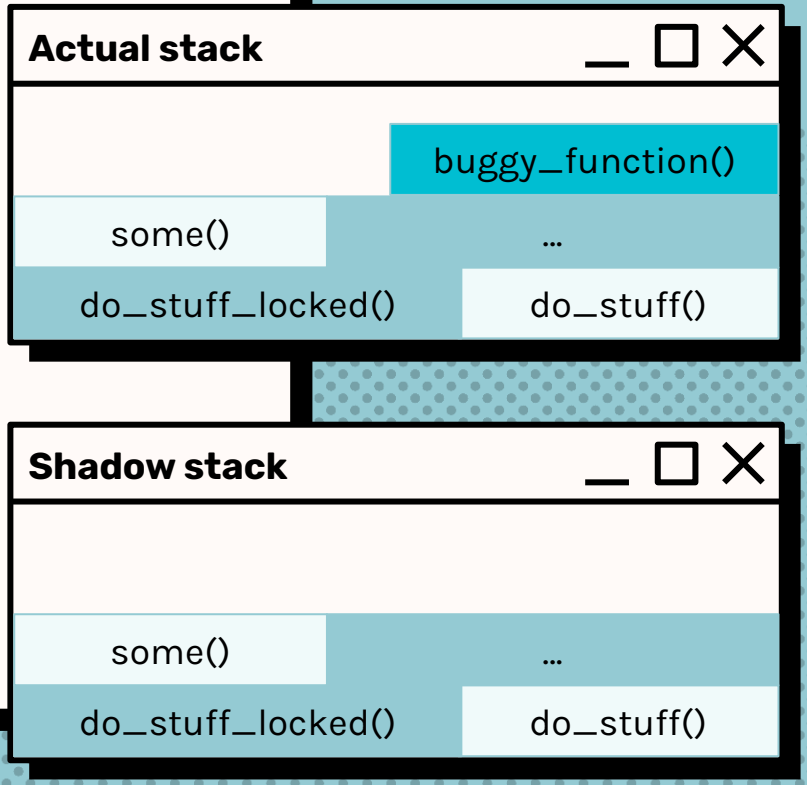
In action

do_stuff()
↳ do_stuff_locked()
↳ some()
↳ PRE-CALL



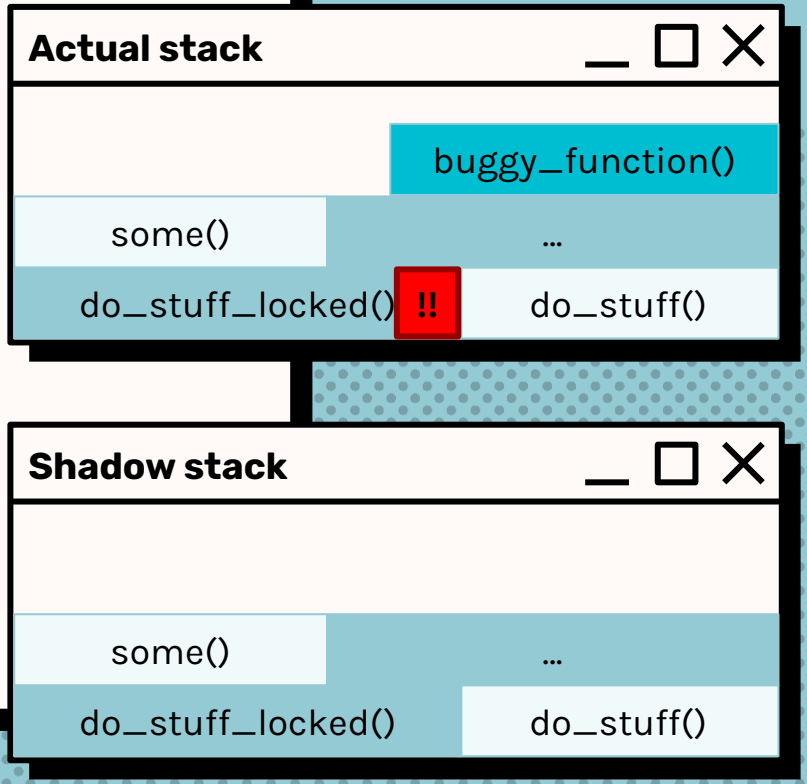
In action

do_stuff()
↳ do_stuff_locked()
↳ some()
↳ buggy_function()



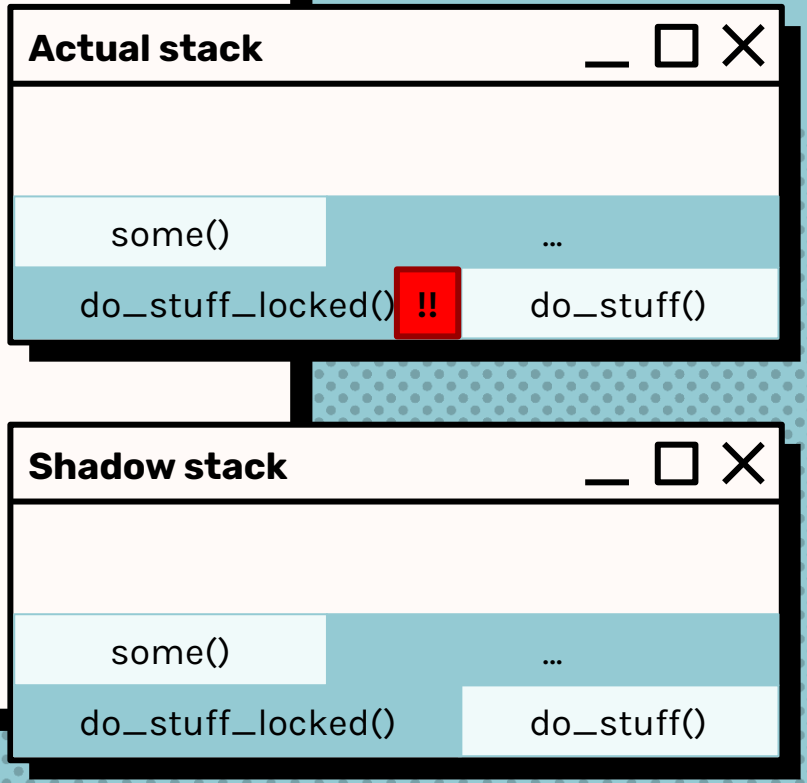
In action

do_stuff()
↳ do_stuff_locked()
↳ some()
↳ buggy_function()



In action

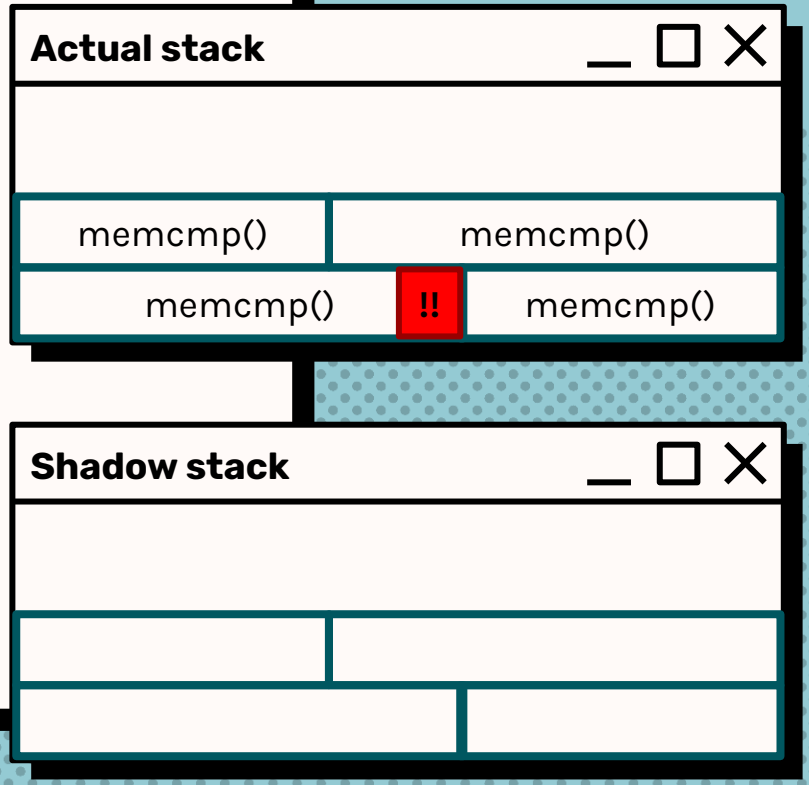
```
do_stuff()
  ↳ do_stuff_locked()
      ↳ some()
          ↳ return
```



In action

do_stuff()
↳ do_stuff_locked()
↳ some()
↳ POST-RETURN

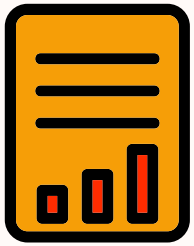
```
memcmp() != 0
```

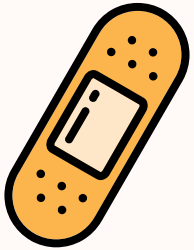


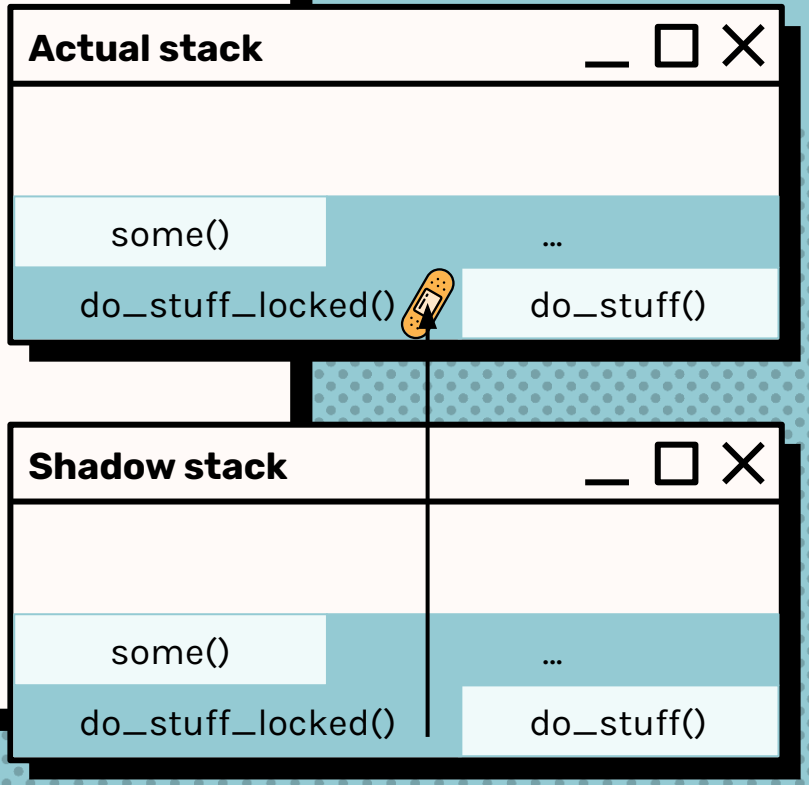
What should happen now?

To heal ...

do_stuff()
↳ do_stuff_locked()
↳ some()
↳ policy = heal

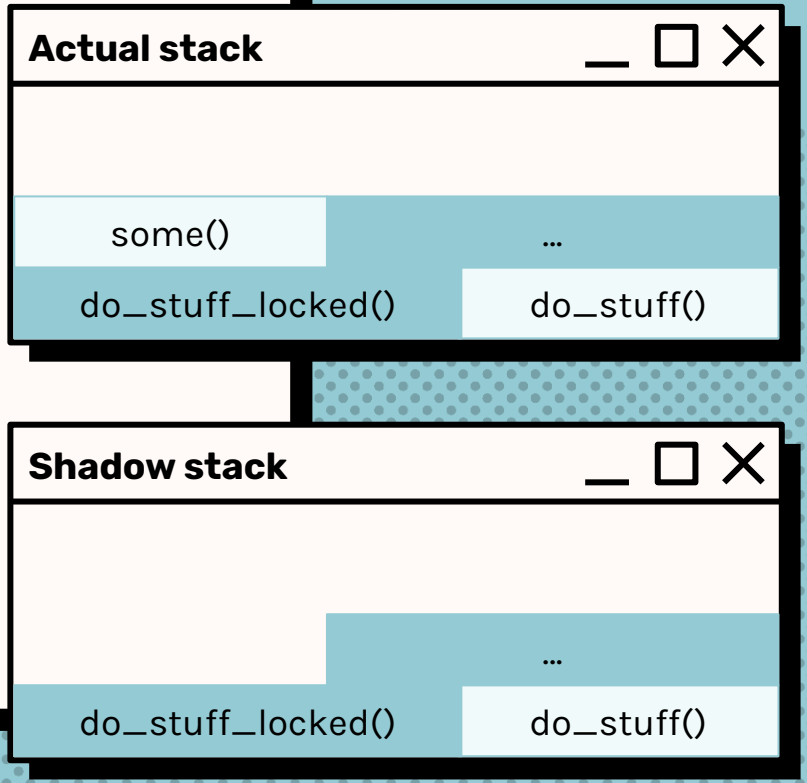






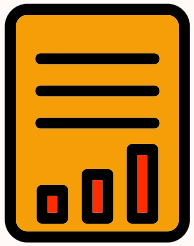
… and continue

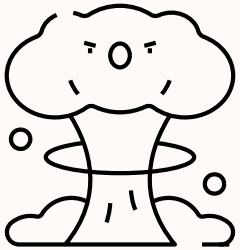
do_stuff()
↳ do_stuff_locked()
↳ some()



… or not to heal?

do_stuff()
↳ do_stuff_locked()
↳ some()
↳ policy = **DIE NOW!**





Actual stack

some()	…	
do_stuff_locked()	!!	do_stuff()

Shadow stack

some()	…	
do_stuff_locked()		do_stuff()

The implementation in C++

RAII + perfect forwarding

```
template <class F, class... Args>
auto invoke(F&& f, Args&&... args)
{
  void *sp = &sp;
  detail::guard c(callee_traits::address(f), &sp); // THE MAGIC
  return std::invoke(std::forward<F>(f), std::forward<Args>(args)...);
}
```

The implementation in C

Macros + compiler extensions

```
#define shst_invoke(f, ...)
   (typeof( f(__VA_ARGS__) ))
   shst_invoke_impl((shst_f)f, ##__VA_ARGS__)
```

Near-perfect forwarding 🙈

```
__builtin_apply_args()
__builtin_apply()
__builtin_return()
```

Usage with C function

Original code

```
void do_stuff(S *s) {
    // ...
    do_stuff_locked(s);
    // ...
}
```

Protected code

```
void do_stuff(S *s) {
    // ...
    shst_invoke(do_stuff_locked, s);
    // ...
}
```

Usage with C++ function

Original code

```
void do_stuff(S *s) {
    // ...
    do_stuff_locked(s);
    // ...
}
```

Protected code

```
void do_stuff(S *s) {
    // ...
    shst::invoke(do_stuff_locked, s);
    // ...
}
```

Usage with C++ method

Original code

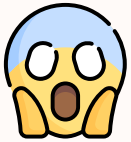
```
void do_stuff(S *s) {
    // ...
    s->brew(0xC0FFEE);
    // ...
}
```

Protected code

```
void do_stuff(S *s) {
    // ...
    shst::invoke(&S::brew, s, 0xC0FFEE);
    // ...
}
```

Usage with LD_PRELOAD

Preload code (feat. shameless ABI abuse!)



```
using shst_f = void* (*)(void* x0, void* x1, … void* x7);

extern "C" void* do_stuff(void* x0, void* x1, … void* x7)
{
  auto real = reinterpret_cast<shst_f>(dlsym(RTLD_NEXT, "do_stuff"));
  return shst::invoke(real, x0, x1, x2, x3, x4, x5, x6, x7);
}
```

Usage with LD_PRELOAD

Preload code with C++ name mangling

```
using shst_f = void* (*)(void* x0, void* x1, … void* x7);

extern "C" void* _Z3fooP1S(void* x0, void* x1, … void* x7)
{
  auto real = reinterpret_cast<shst_f>(dlsym(RTLD_NEXT, "_Z3fooP1S"));
  return shst::invoke(real, x0, x1, x2, x3, x4, x5, x6, x7);
}
```

DEMO

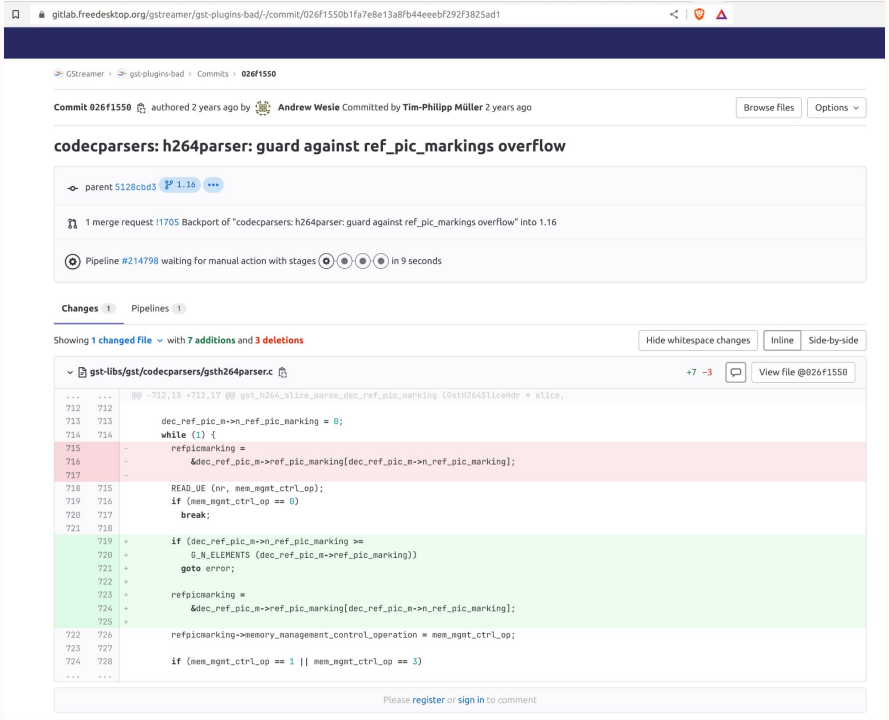
keep your fingers crossed...

Summary

Success!

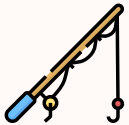
Root cause

- Missing input data sanitation in one GStreamer function
- Array of size 10 written at index 240
- Fixed upstream a few months earlier

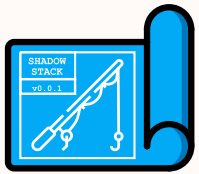


Shadow Stack maturity

Fish



Rod



Blueprint

Main lessons

Use stack-protector consciously

Canaries are your friends
but not almighty saviors.

Update

your dependencies regularly!

Order matters

If you can’t change it
at least watch out!

Up-growing?

Call trace:

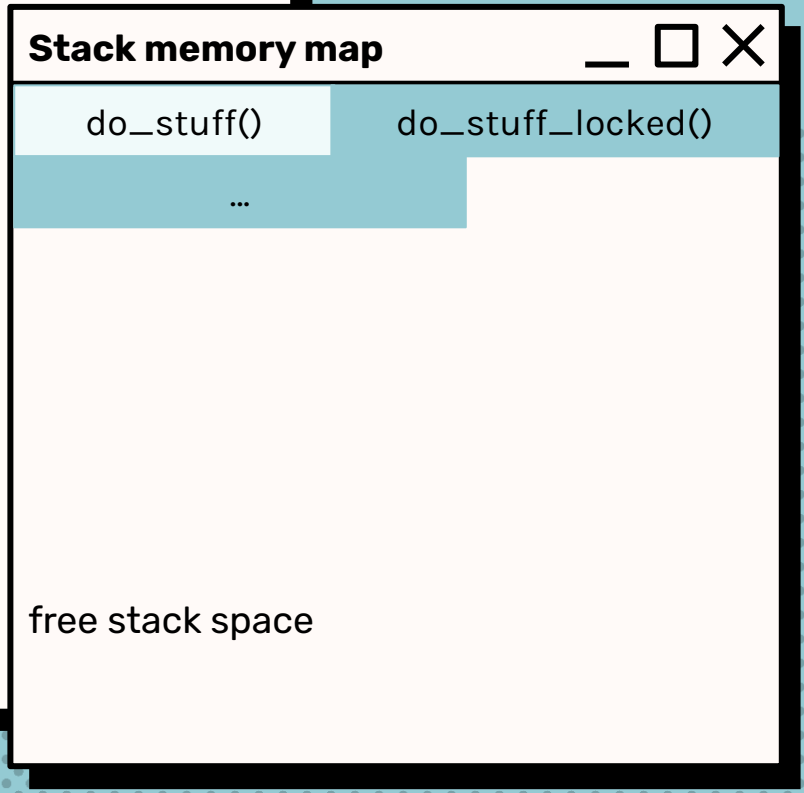
- do_stuff()

Stack memory map

Up-growing?

Call trace:

- do_stuff()
- do_stuff_locked()

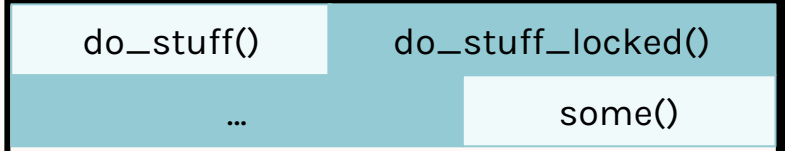


Up-growing?

Call trace:

- do_stuff()
- do_stuff_locked()
- some()

Stack memory map

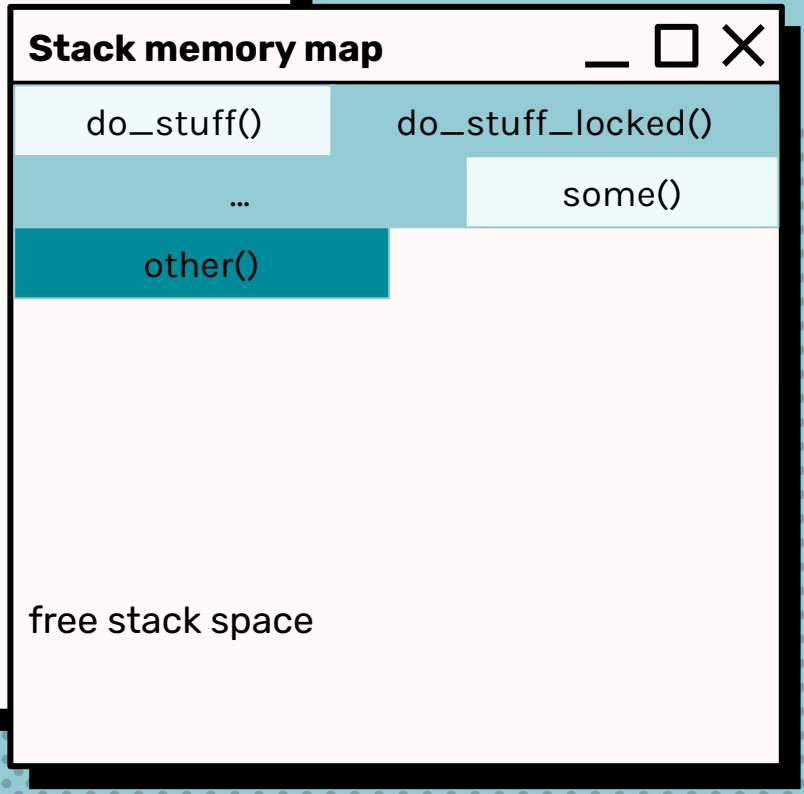


free stack space

Up-growing?

Call trace:

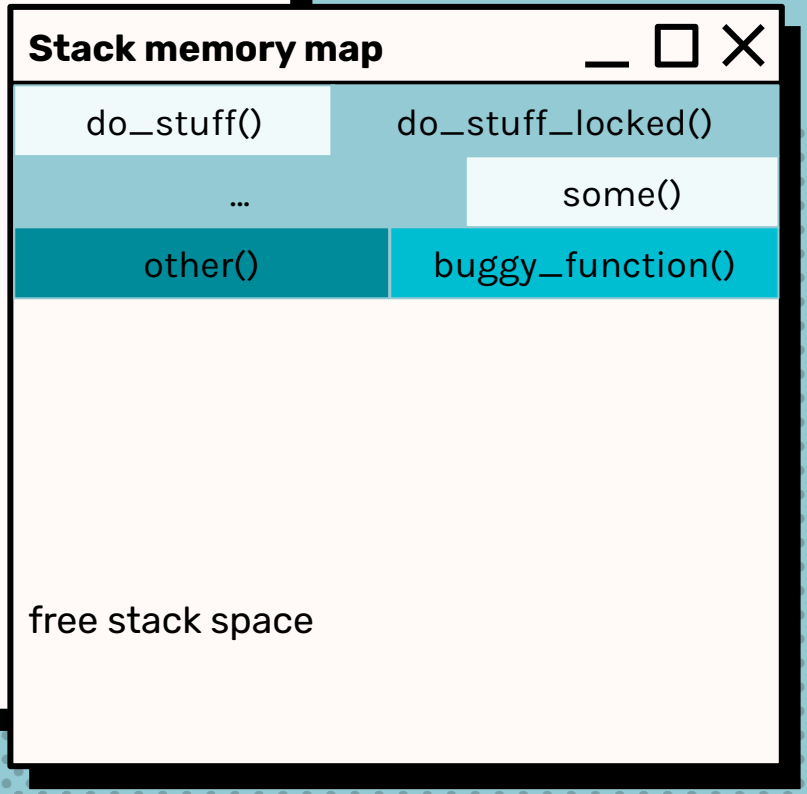
- do_stuff()
- do_stuff_locked()
- some()
- other()



Up-growing?

Call trace:

- do_stuff()
- do_stuff_locked()
- some()
- other()
- buggy_function()



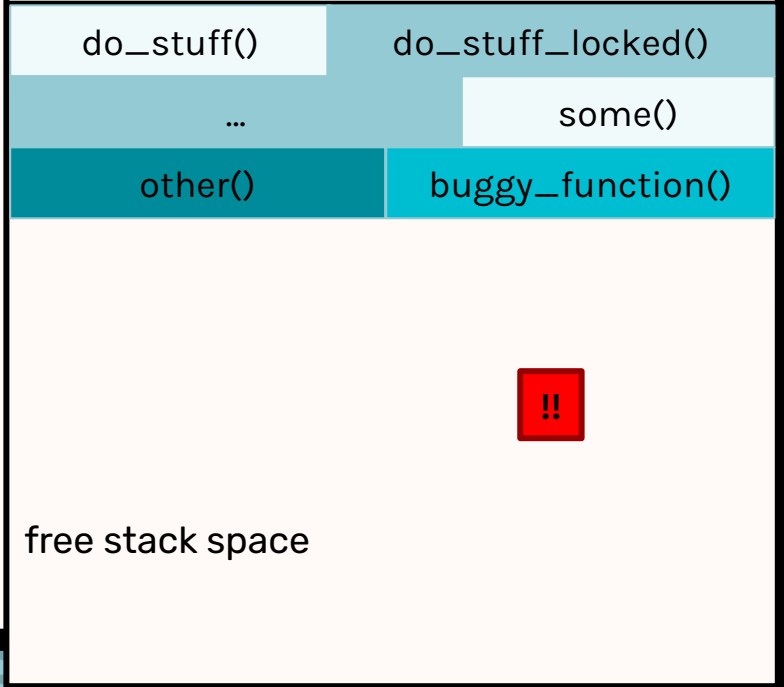
Up-growing?

Call trace:

- do_stuff()
- do_stuff_locked()
- some()
- other()
- buggy_function()

```
void buggy_function() {
    uint8_t bytes[8];
    bytes[666] = 0x03;    // MEH!
}
```

Stack memory map



Can we switch?

Q & A

https://github.com/bmoczulski/shadow-stack/

Thank you

Bartosz Moczulski

will return

(perhaps) in "You only leak twice"