

# Shadow Stack

How To Submerge Beneath C++ and Fix Memory Corruptions Immune to Stack Protector

BARTOSZ MOCZULSKI

# Polish Phonology 101

## ~~Shadow Stack~~

How To Submerge Beneath C++ and Fix Memory Corruptions Immune to Stack Protector

~~H~~ ~~HOO~~  
BARTOSZ ~~X~~ MOCZULSKI ~~X~~

**18**

**Sep 2025**

# Shadow Stack

or:

How To Submerge Beneath C++  
and Fix Memory Corruptions  
Immune to Stack Protector



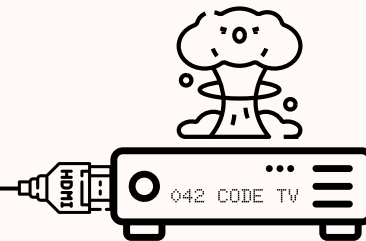
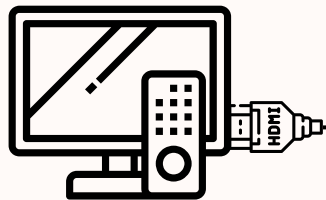
**Bartosz Moczulski**

<https://bartosz.codes/>

**CppCon 2025**

Aurora, Colorado, USA

# I want you



# to watch TV

and streaming services



# 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 (( $\pi$  = 3) == e) { ... }`

## Integration





**I am a Linux detective**

# Your takeaways

## Amusement

- moved to Rust already?
- enjoy peer misery 😈

## Awareness

- stack is tricky
- beware!

## Action

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

# 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>`





# The Encounter



# The Chat





# The Chat



=





# The Chat



+



=





# The Chat



+



=



1/day





# The Chat



+



+



=

=



1/day





# The Chat



+



+



=

=



1/day

+



18



# The Embrace

# GDB coredump analysis

SIGSEGV in Thread "video\_stream"

```
(gdb) bt
#0 pthread_mutex_unlock (m=0x00ffffcc03)
#1 do_stuff (s=0xffffffffcc00)
...
#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  
with you right now?

# ARM 64 calling convention

32 fully interchangeable registers\*

\* except not quite

Source: [https://en.wikipedia.org/wiki/Calling\\_convention#ARM\\_\(A64\)](https://en.wikipedia.org/wiki/Calling_convention#ARM_(A64))

# ARM 64 calling convention

## subroutines

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

## stack

**x31 (SP)**  
stack pointer

Source: [https://en.wikipedia.org/wiki/Calling\\_convention#ARM\\_\(A64\)](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

# ARM 64 calling convention

## MUST preserve

x19-x28

subroutine MUST preserve  
them (callee-saved)

## subroutines

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

## stack

x31 (SP)  
stack pointer

Source: [https://en.wikipedia.org/wiki/Calling\\_convention#ARM\\_\(A64\)](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.

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)

# ARM 64 calling convention

## args + ret(s)

**x0-x7**

input arguments  
returned value(s)

## MUST preserve

**x19-x28**

subroutine MUST preserve  
them (callee-saved)

## subroutines

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

## stack

**x31 (SP)**  
stack pointer

Source: [https://en.wikipedia.org/wiki/Calling\\_convention#ARM\\_\(A64\)](https://en.wikipedia.org/wiki/Calling_convention#ARM_(A64))

# 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.

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 0xfffffccc08

Here x20 is **broken** 0x00ffffcc03  
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 == 0xffffffffcc08
```

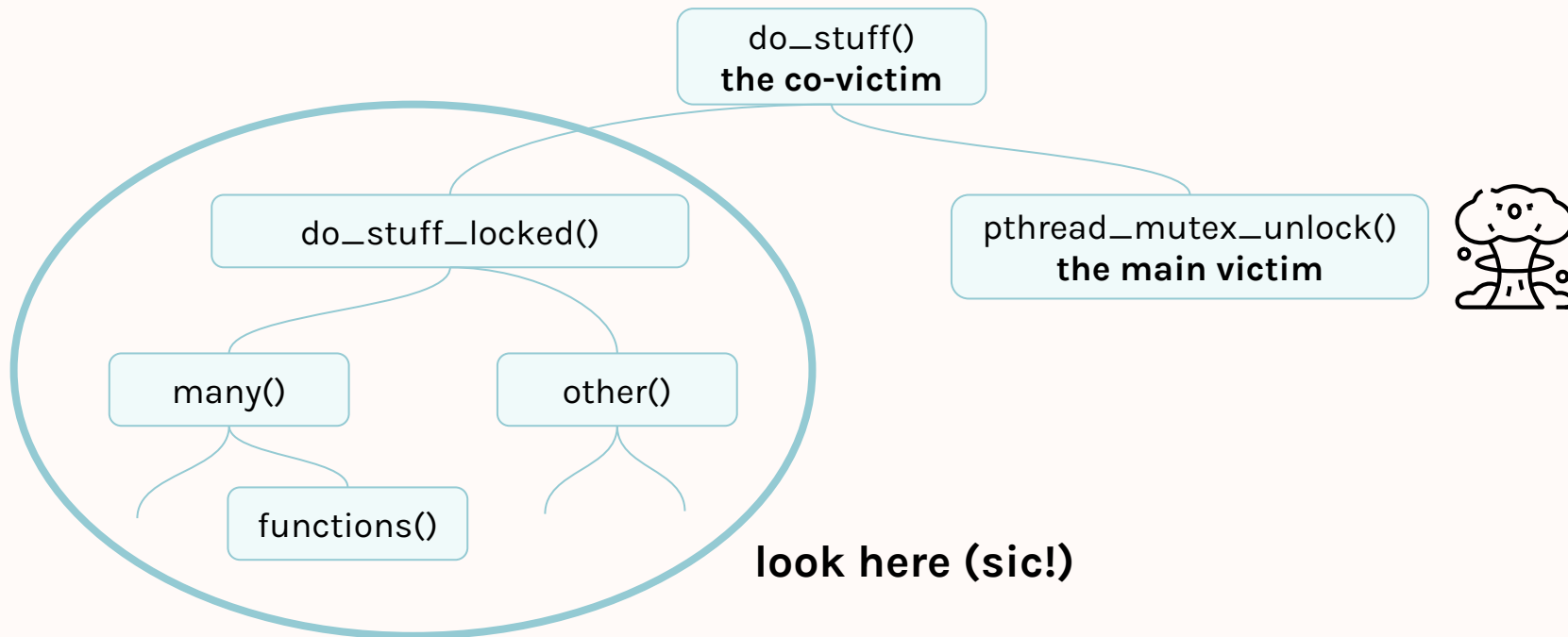
... do the actual stuff (with **do\_stuff**'s x19 and x20 temporarily on stack!)

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

```
ldp    x29, x30, [sp], #32  
ret
```

End of assembler dump.

# The problem





# 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



free stack space

...

`do_stuff_locked()`

`do_stuff()`



# How come?

Call trace:

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

## Stack memory map



free stack space

`some()`

...

`do_stuff_locked()`

`do_stuff()`



# How come?

Call trace:

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

## Stack memory map



free stack space





# How come?

Call trace:

- do\_stuff()
- do\_stuff\_locked()
- some()
- other()
- buggy\_function()

## Stack memory map



free stack space

buggy\_function()

other()

some()

...

do\_stuff\_locked()

do\_stuff()

# 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

free stack space

buggy\_function()

other()

some()

...

do\_stuff\_locked()



do\_stuff()

# With stack-protector?

Is `-fstack-protector` a silver bullet?

Spoiler alert: **NO!**

# With stack-protector?

Stack protector:

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

**Stack**

buggy\_function()



do\_stuff\_locked()



do\_stuff()



# With stack-protector?

Overwriting current canary:

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

**Stack**

buggy\_function()



do\_stuff\_locked()



do\_stuff()



# With stack-protector?

Overwriting parent canary:

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



**Stack**

buggy\_function()



do\_stuff\_locked()

!!

do\_stuff()



# With stack-protector?

Overwriting other areas:

- not detected at all (sic!)



**Stack**

buggy\_function()



do\_stuff\_locke



do\_stuff()



# Is stack-protector any good?

What's the point of it then?

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

**Stack**

buggy\_function



!!



o\_stuff()



52



# The Elimination

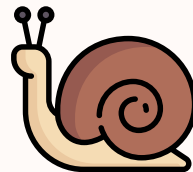


# Do you know

any tools that could help  
identify the corruption place?

# valgrind?

NOPE  
way too slow



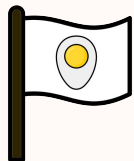
# **Behold!**

# **Shadow Stack**

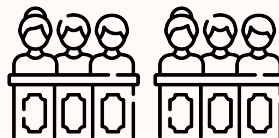
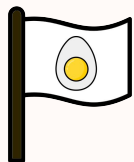
# Name inspiration

Shadow cabinet (politics) and shadow DOM (web)

Big-endians\*



Little-endians opposition

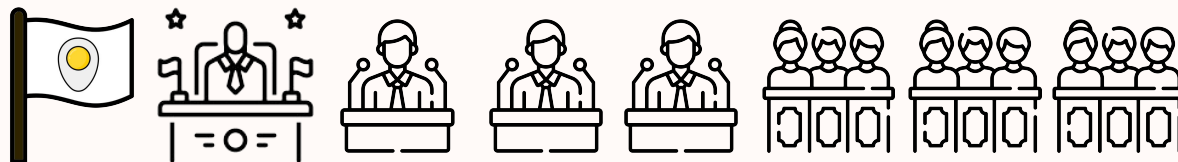


\* reference to *Gulliver's Travels* by Jonathan Swift in byte-order names is no coincidence

# Name inspiration

Shadow cabinet (politics) and shadow DOM (web)

Big-endians\* government (ruling cabinet)



Little-endians opposition

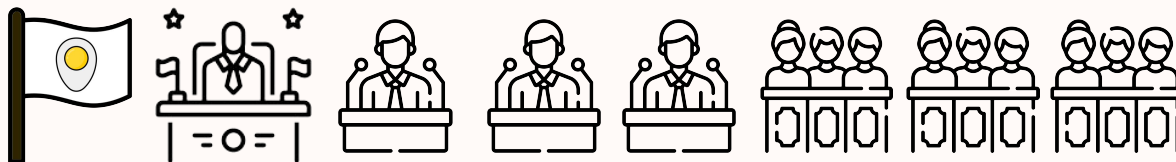


\* reference to *Gulliver's Travels* by Jonathan Swift in byte-order names is no coincidence

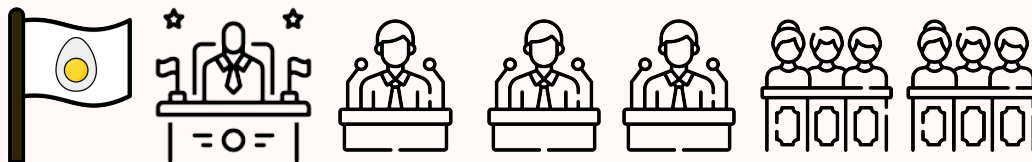
# Name inspiration

Shadow cabinet (politics) and shadow DOM (web)

Big-endians\* government (ruling cabinet)



Little-endians opposition (shadow cabinet)

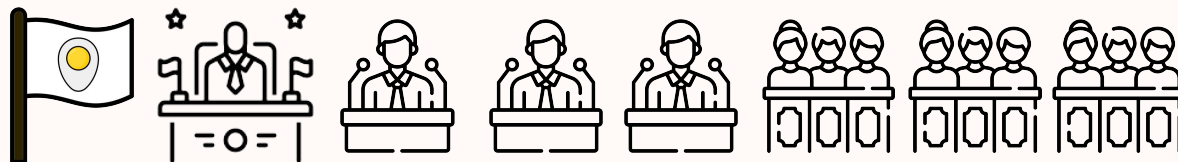


\* reference to *Gulliver's Travels* by Jonathan Swift in byte-order names is no coincidence

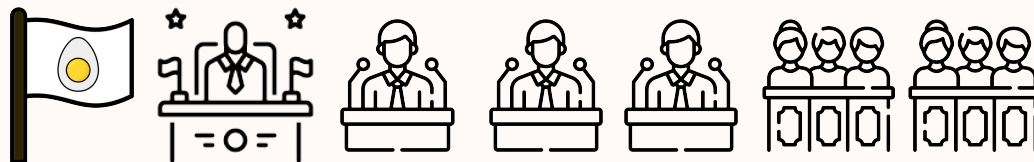
# Name inspiration

Shadow cabinet (politics) and shadow DOM (web)

Big-endians\* government (ruling cabinet)



Little-endians opposition (shadow cabinet)

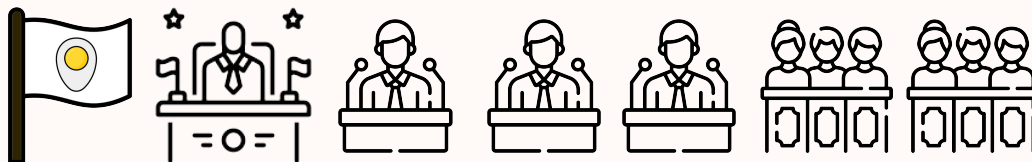


\* reference to *Gulliver's Travels* by Jonathan Swift in byte-order names is no coincidence

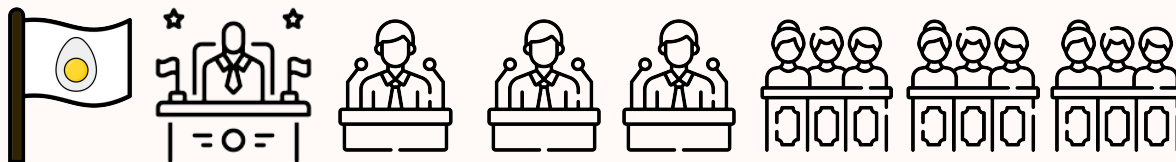
# Name inspiration

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

# 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);  
    // ...  
}
```

# 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)...);
}
```

# 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;
```

# 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);  
    // ...  
}
```

# 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 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);
}
```

69



# In action

do\_stuff()

Actual stack



do\_stuff()

Shadow stack



# In action

do\_stuff()

↳ PRE-CALL

Actual stack

\_ □ ×

do\_stuff()

Shadow stack

\_ □ ×

do\_stuff()





# In action

do\_stuff()

↳ PRE-CALL

Actual stack



memcmp()

Shadow stack



# In action

do\_stuff()

↳ do\_stuff\_locked()

## Actual stack

do\_stuff\_locked()

do\_stuff()

## Shadow stack

do\_stuff()

# In action

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

## Actual stack

do\_stuff\_locked()

do\_stuff()

## Shadow stack

do\_stuff\_locked()

do\_stuff()



# In action

do\_stuff()

↳ do\_stuff\_locked()

↳ PRE-CALL

## Actual stack

memcmp()

memcmp()

memcmp()

## Shadow stack

# In action

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

## Actual stack

some()

...

do\_stuff\_locked()

do\_stuff()

## Shadow stack

do\_stuff\_locked()

...

do\_stuff()

# In action

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

## Actual stack

some()

...

do\_stuff\_locked()

do\_stuff()

## Shadow stack

some()

...

do\_stuff\_locked()

do\_stuff()

# In action

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

## Actual stack

memcmp()

memcmp()

memcmp()

memcmp()

## Shadow stack

# In action

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

## Actual stack

buggy\_function()

some()

...

do\_stuff\_locked()

do\_stuff()

## Shadow stack

some()

...

do\_stuff\_locked()

do\_stuff()

# In action

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

## Actual stack

		buggy_function()
some()	...	
do_stuff_locked()	!!	do_stuff()

## Shadow stack

some()	...	
do_stuff_locked()		do_stuff()

# In action

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

## Actual stack

some()

...

do\_stuff\_locked()



do\_stuff()

## Shadow stack

some()

...

do\_stuff\_locked()

do\_stuff()

# In action

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

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

## Actual stack

memcmp()

memcmp()

memcmp()

!!

memcmp()

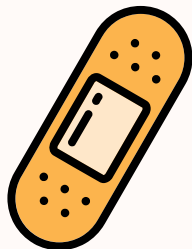
## Shadow stack



**What should  
happen now?**

# To heal ...

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



## Actual stack

some()

...

do\_stuff\_locked()

do\_stuff()

## Shadow stack

some()

...

do\_stuff\_locked()

do\_stuff()

# ... and continue

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

## Actual stack

some()

...

do\_stuff\_locked()

do\_stuff()

## Shadow stack

...

do\_stuff\_locked()

do\_stuff()

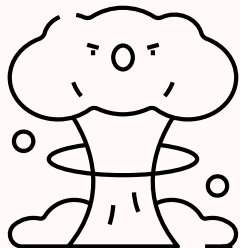
# ... or not to heal?

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()

# DEMO

keep your fingers crossed...

87

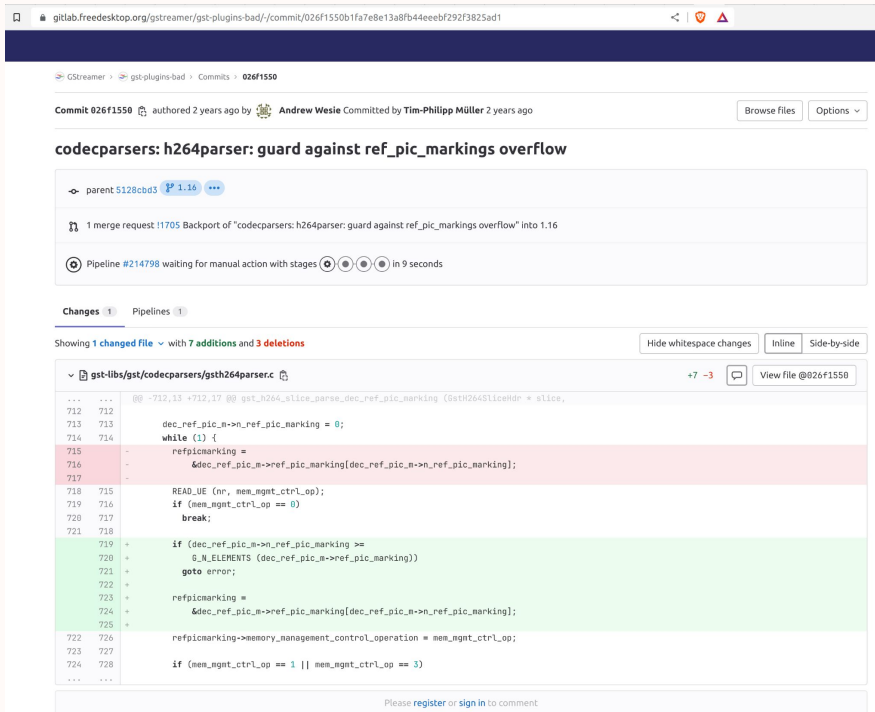


# Summary

# 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

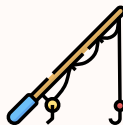


The screenshot shows a GitHub commit page for the repository `gstreamer/gst-plugins-bad`. The commit hash is `026f1550`, authored 2 years ago by `Andrew Wesie` and committed by `Tim-Philipp Müller`. The commit message is `codecparsers: h264parser: guard against ref_pic_markings overflow`. The page shows a merge request from `parent 5128cbd5` to `1.16`. The commit details section shows `1 merge request 1705` and a pipeline `#214798` waiting for manual action. The `Changes` section shows `Showing 1 changed file` with `7 additions` and `3 deletions`. The file `gst-lib/gst/codecparsers/gsth264parser.c` is shown with a diff. The diff highlights a `while` loop that was previously `while (1) {` and is now `while (dec_ref_pic_m->n_ref_pic_marking == 0;)`. The `refpicmarking` variable is updated to `&dec_ref_pic_m->ref_pic_marking[dec_ref_pic_m->n_ref_pic_marking;]`. The `READ_U` function is called with `mem_mngt_ctrl_op` and `if (mem_mngt_ctrl_op == 0) break;` is added. The `if` statement is updated to `if (dec_ref_pic_m->n_ref_pic_marking == 6_N_ELEMENTS (dec_ref_pic_m->ref_pic_marking)) goto error;`. The `refpicmarking` variable is updated to `&dec_ref_pic_m->ref_pic_marking[dec_ref_pic_m->n_ref_pic_marking;]`. The `refpicmarking->memory_management_control_operation` is updated to `mem_mngt_ctrl_op`. The `if` statement is updated to `if (mem_mngt_ctrl_op == 1 || mem_mngt_ctrl_op == 3)`.

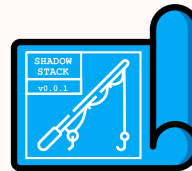
# Shadow Stack maturity



**Fish**

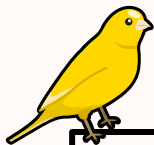


**Rod**



**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

`do_stuff()`

free stack space

# Up-growing?

Call trace:

- `do_stuff()`
- `do_stuff_locked()`

## Stack memory map

`do_stuff()``do_stuff_locked()`

...

free stack space

# Up-growing?

Call trace:

- do\_stuff()
- do\_stuff\_locked()
- some()

## Stack memory map

do\_stuff()

do\_stuff\_locked()

...

some()

free stack space

# Up-growing?

Call trace:

- do\_stuff()
- do\_stuff\_locked()
- some()
- other()

## Stack memory map

do\_stuff()

do\_stuff\_locked()

...

some()

other()

free stack space

# Up-growing?

Call trace:

- do\_stuff()
- do\_stuff\_locked()
- some()
- other()
- buggy\_function()

## Stack memory map

do\_stuff()

do\_stuff\_locked()

...

some()

other()

buggy\_function()

free stack space

# 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

do\_stuff()

do\_stuff\_locked()

...

some()

other()

buggy\_function()



free stack space

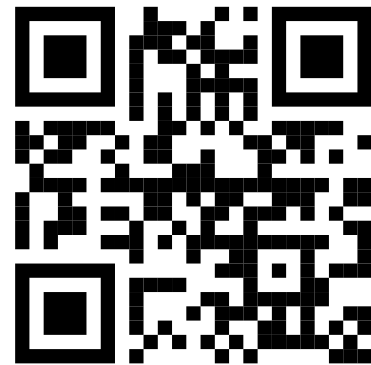


# Can we switch?

98



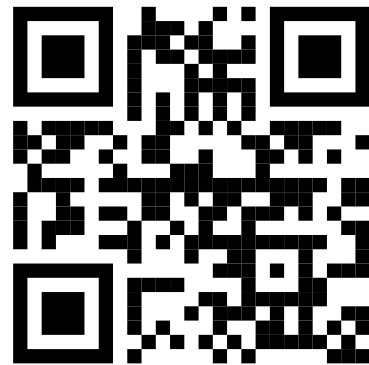
# Q & A



**FEEDBACK**

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

# Thank you



**FEEDBACK**

Bartosz Moczulski  
will return in  
"Once Upon a Thread in the Mutex:  
The Road to `std::synchronized_value<T>` and Why It Matters"