



Shadow Stack

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

BARTOSZ MOCZULSKI



20
25



September 13 - 19



Polish Phonology 101

~~Shadow Stack~~

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

H HOO
BARTOSZ MOCZNULSKI



20
25



September 13 - 19

18

Sep 2025



Shadow Stack

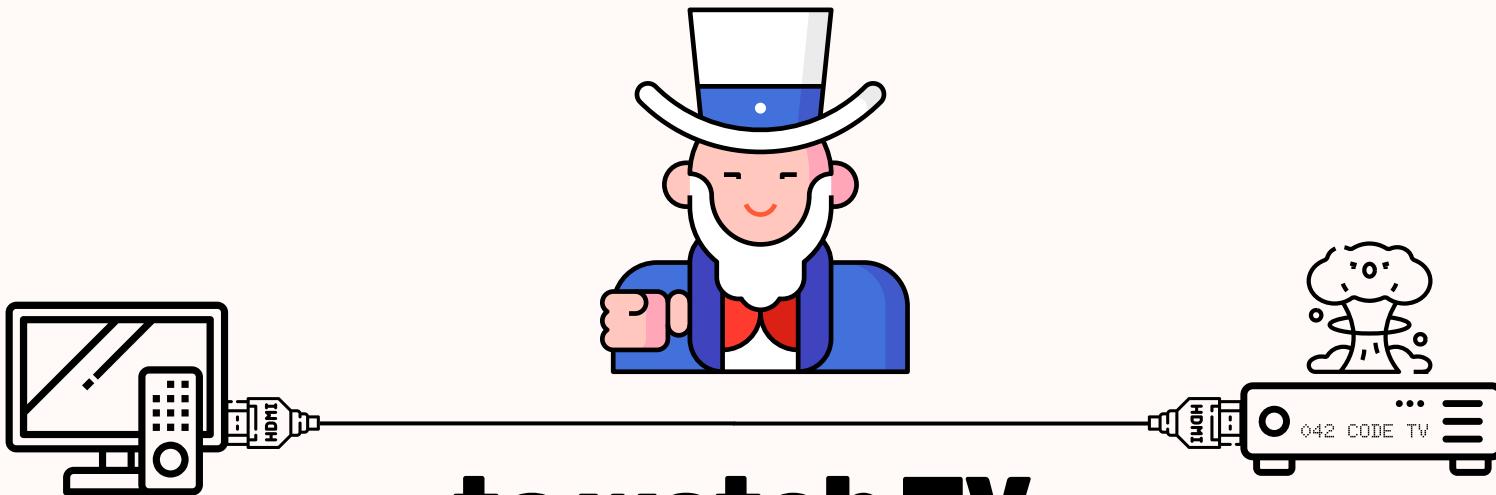
or:

**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 ((π = 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



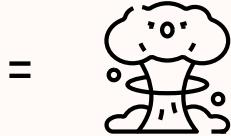
+



+



=



1/day

The Chat



+



+



=



=

1/day



The Embrace

GDB coredump analysis

SIGSEGV in Thread "video_stream"

```
(gdb) bt
#0 pthread_mutex_unlock (m=0x0fffffcc03)
#1 do_stuff (s=0xfffffffcc00)
...
#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

ARM 64 calling convention

subroutines

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

stack

x31 (SP)
stack pointer

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

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

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 0xffffffffc08

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

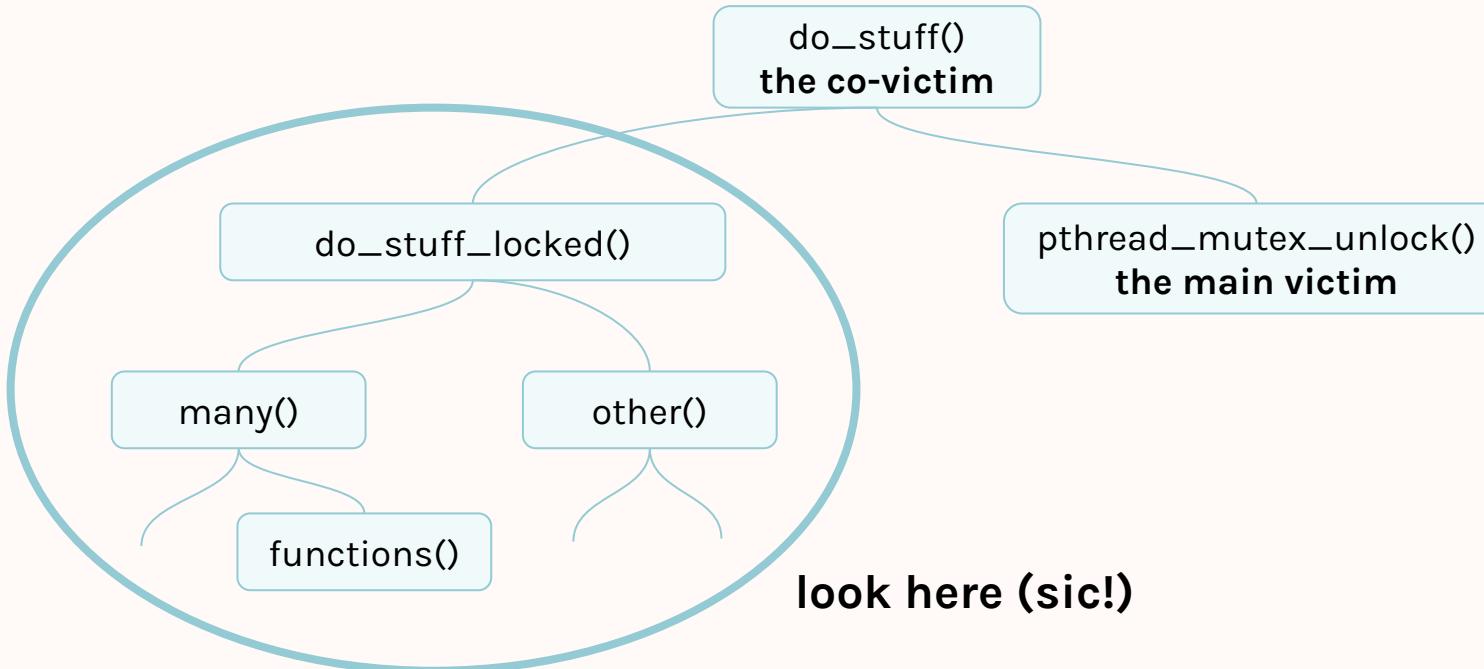
```
(gdb) disassemble do_stuff_locked
Dump of assembler code for function do_stuff_locked:
  stp  x29, x30, [sp, #-32]!
  mov  x29, sp

  stp  x19, x20, [sp, #16]          // push x20 == 0xfffffffcc08
  ...
  ... do the actual stuff (with do_stuff's x19 and x20 temporarily on stack!)

  ldp  x19, x20, [sp, #16]          // pop  x20 == 0x00fffffcc03
  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()



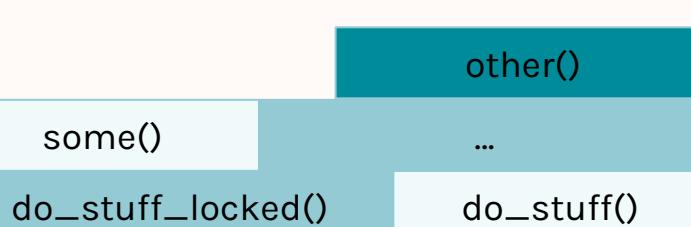
How come?

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



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



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!)



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()



The Elimination



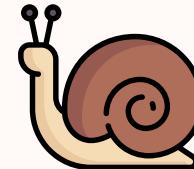
Do you know

any tools that could help
identify the corruption place?



valgrind?

NOPE
way too slow





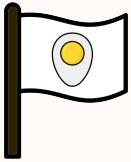
Behold!

Shadow Stack

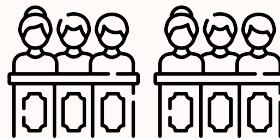
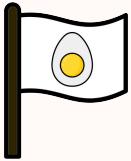
Name inspiration

Shadow cabinet (politics) and shadow DOM (web)

Big-endians*



Little-endians opposition

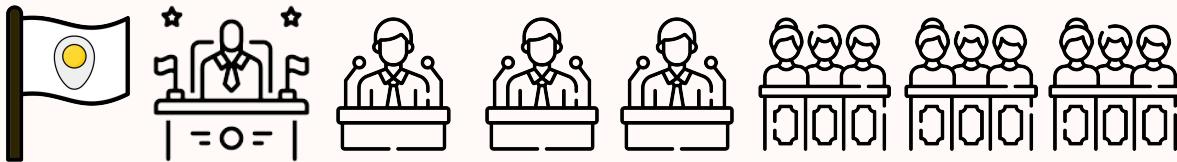


* reference to *Gulliver's Travels* by Jonathan Swift in byte-order names is no coincidence

Name inspiration

Shadow cabinet (politics) and shadow DOM (web)

Big-endians* government (ruling cabinet)



Little-endians opposition

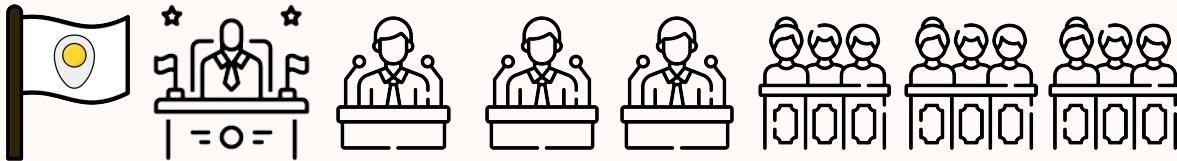


* reference to *Gulliver's Travels* by Jonathan Swift in byte-order names is no coincidence

Name inspiration

Shadow cabinet (politics) and shadow DOM (web)

Big-endians* government (ruling cabinet)



Little-endians opposition (shadow cabinet)

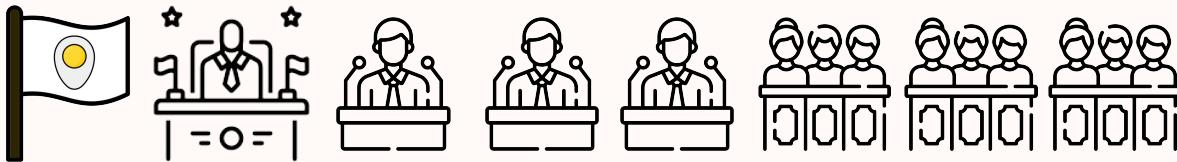


* reference to *Gulliver's Travels* by Jonathan Swift in byte-order names is no coincidence

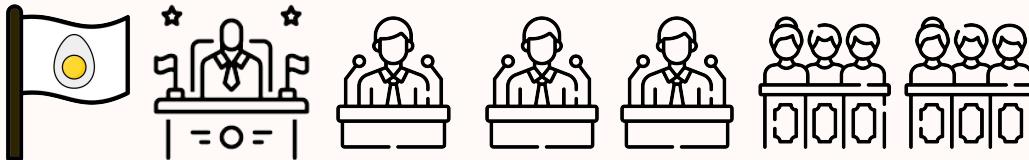
Name inspiration

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

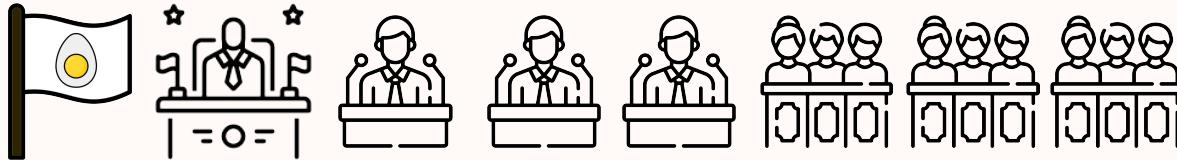
Name inspiration

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);
}
```

In action

do_stuff()

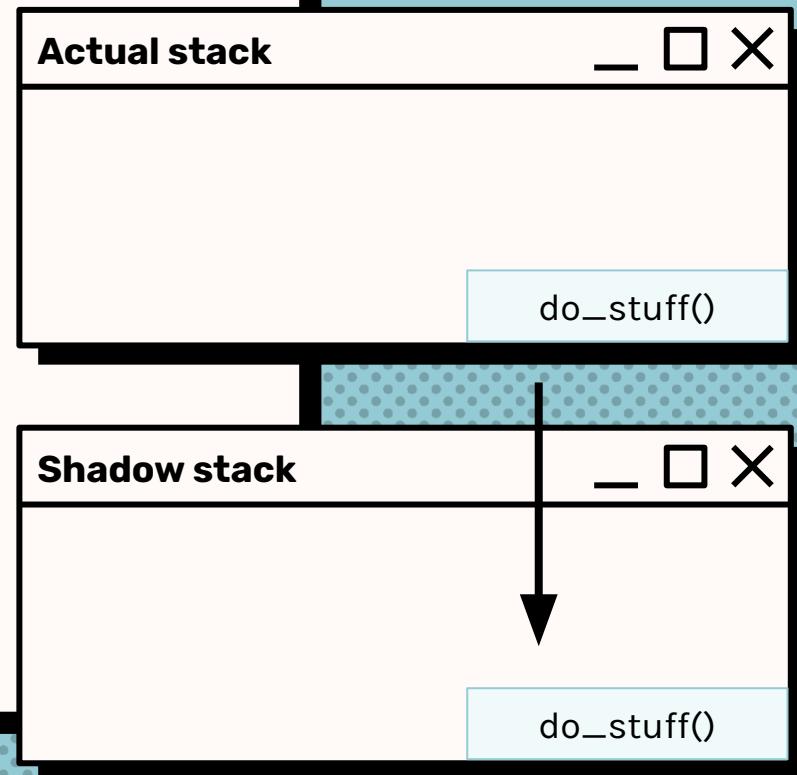
Actual stack

do_stuff()

Shadow stack

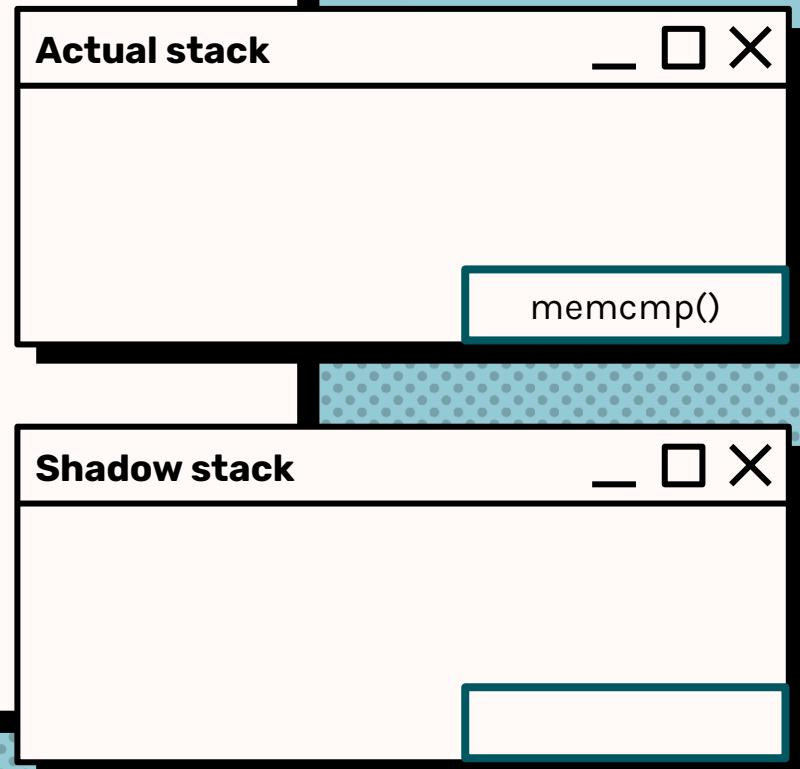
In action

do_stuff()
↳ PRE-CALL



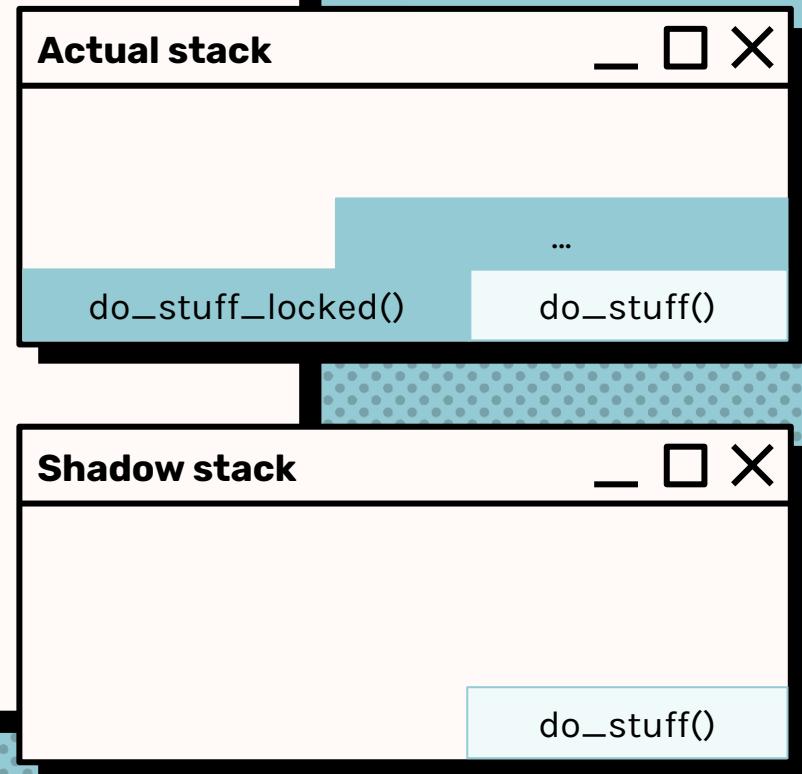
In action

do_stuff()
↳ PRE-CALL



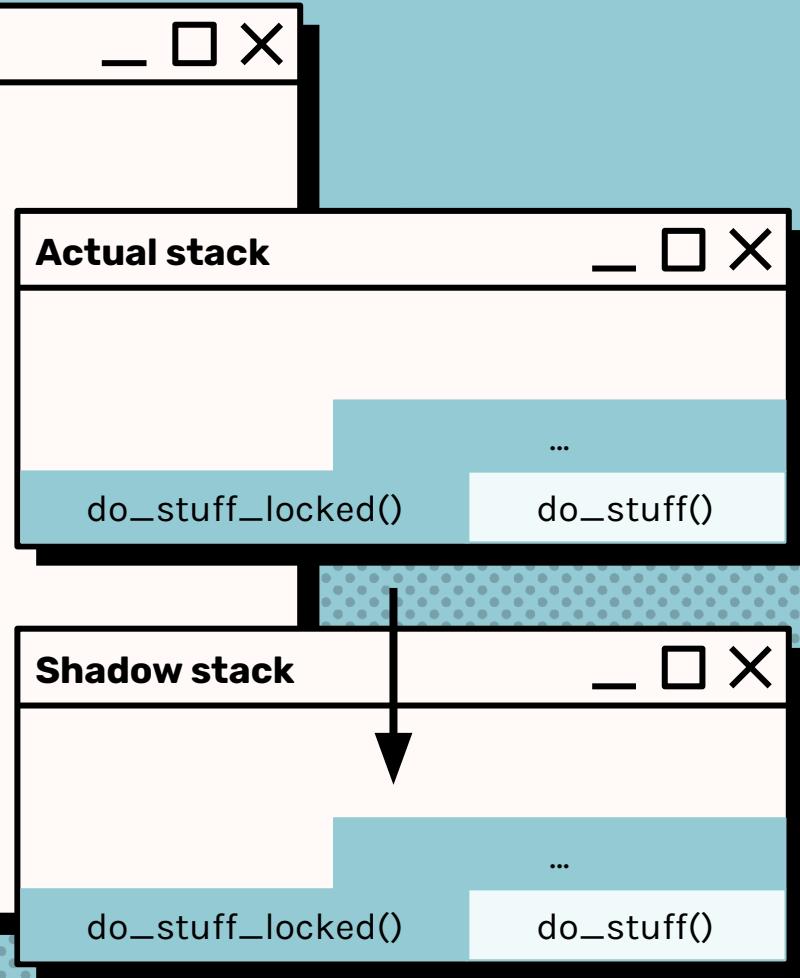
In action

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



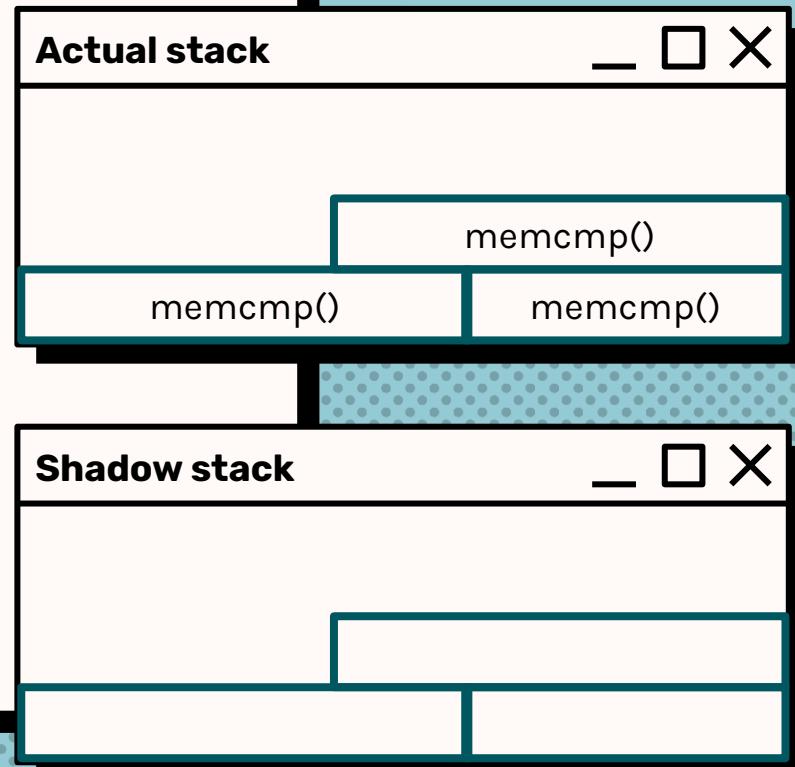
In action

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



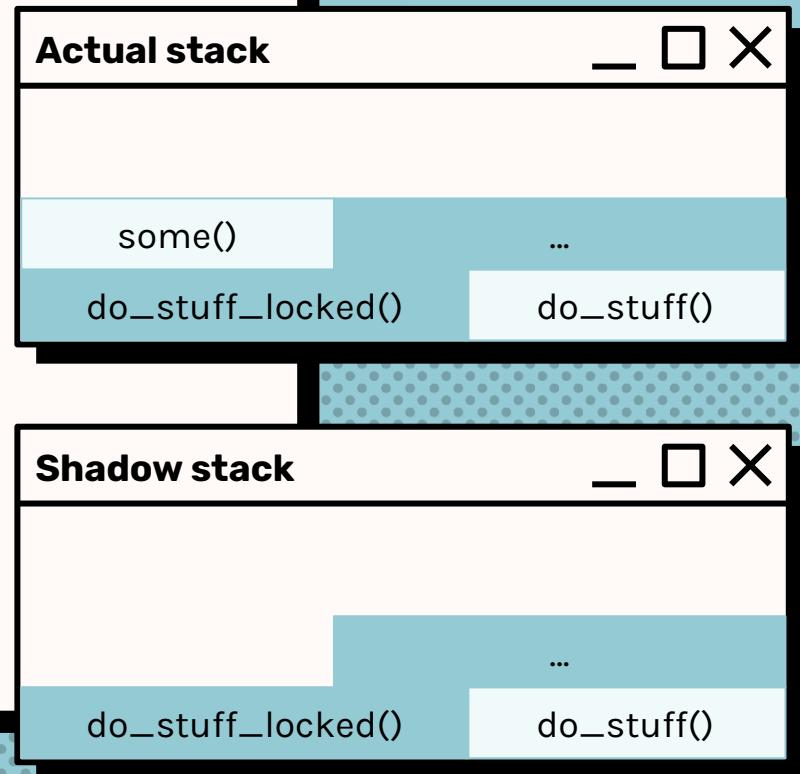
In action

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



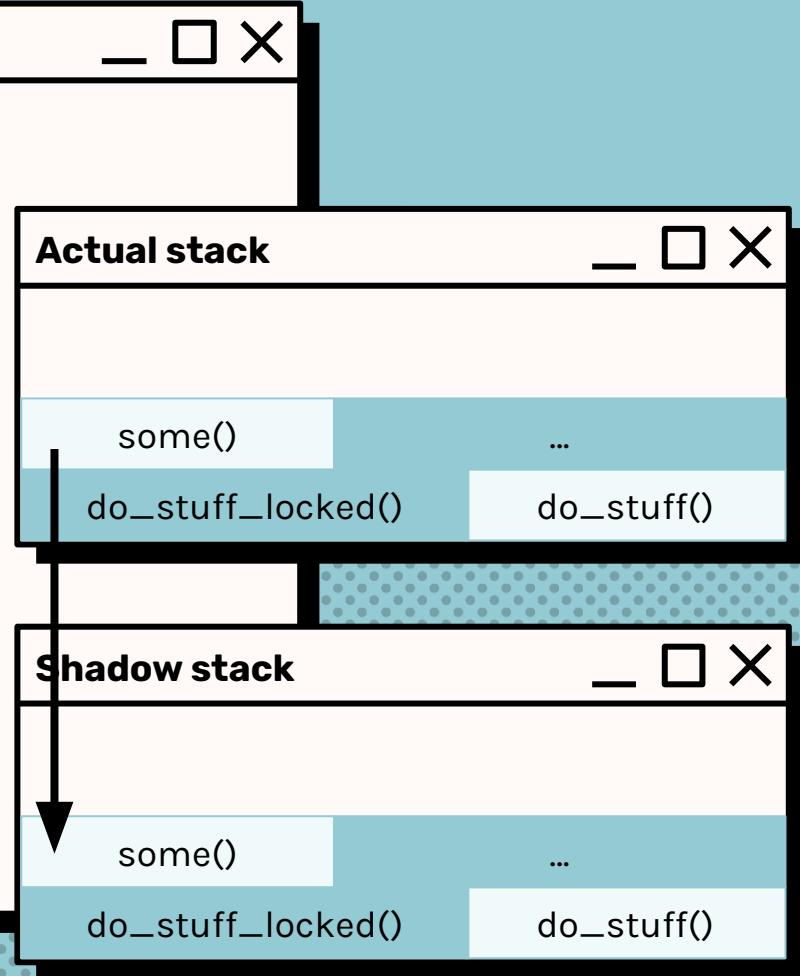
In action

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



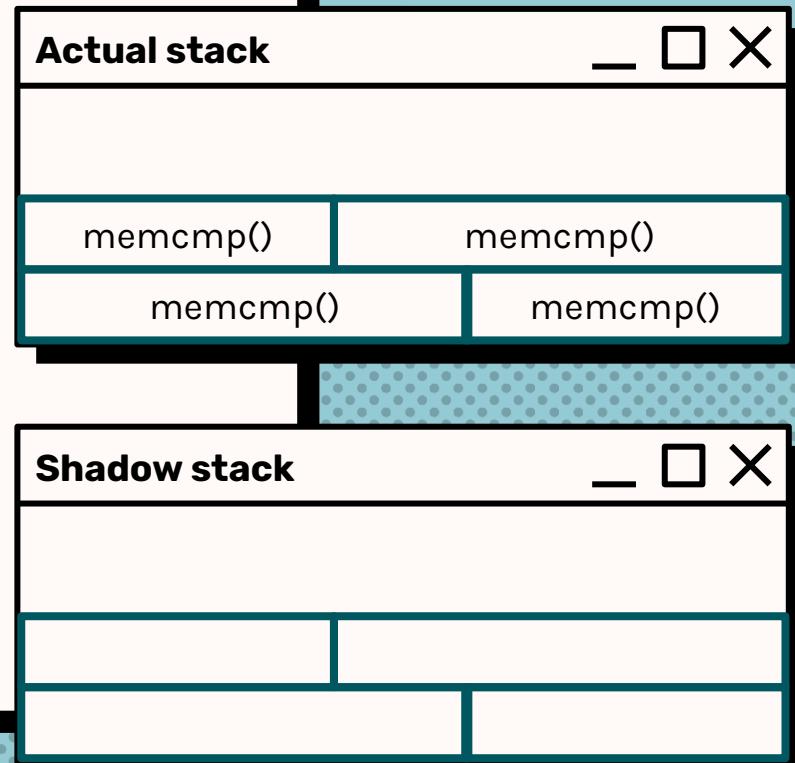
In action

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



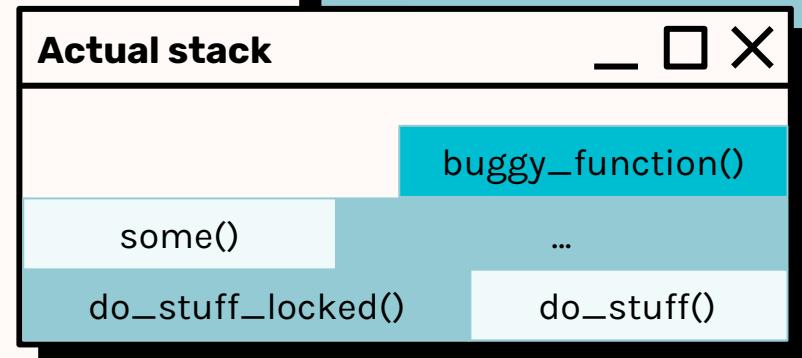
In action

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



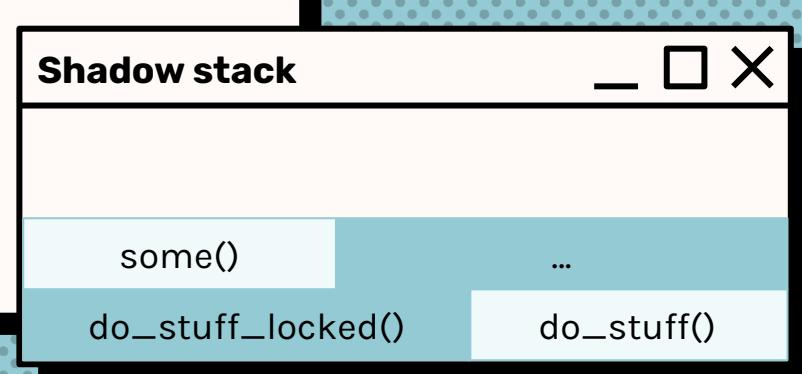
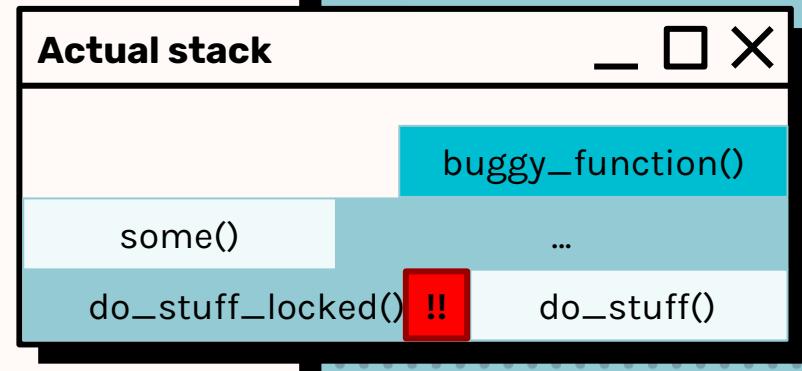
In action

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



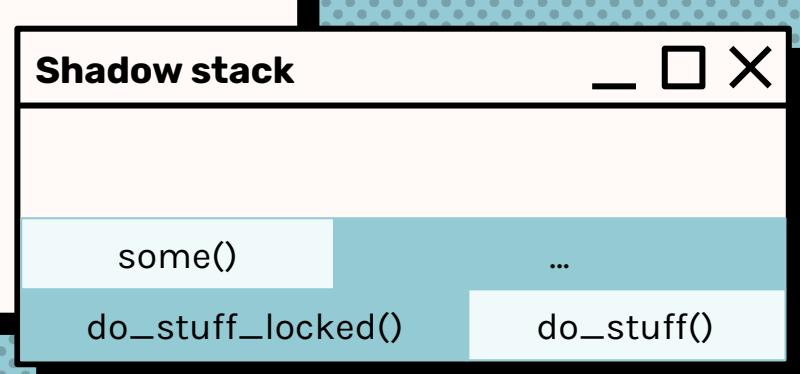
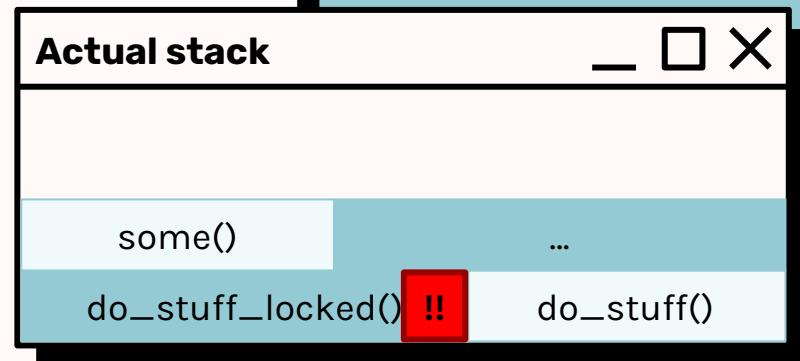
In action

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



In action

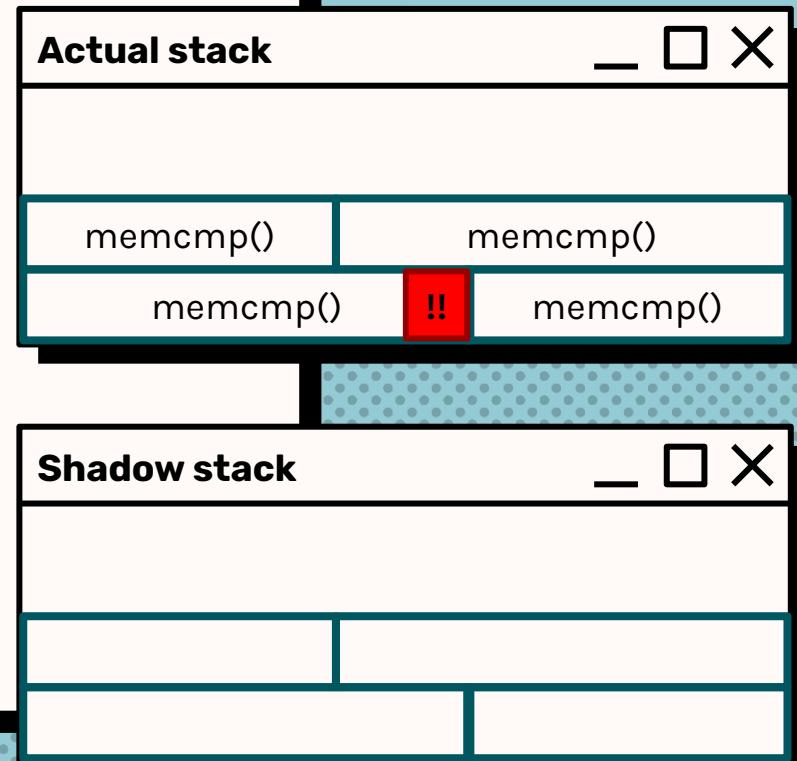
```
do_stuff()  
↳ do_stuff_locked()  
↳ some()  
↳ return
```



In action

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

memcmp() != 0

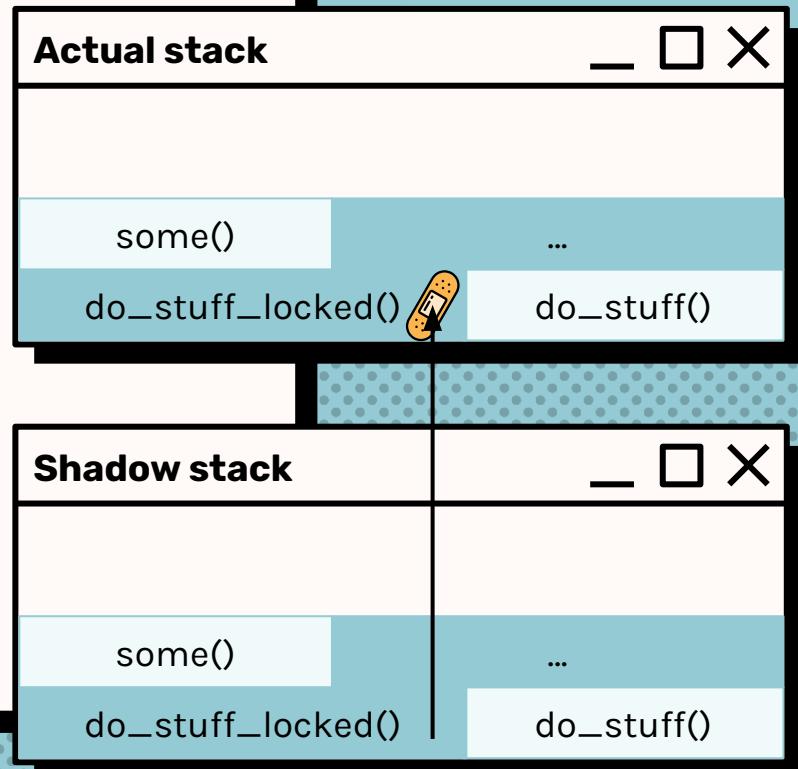
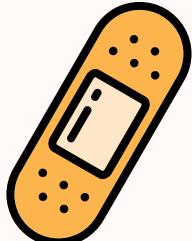
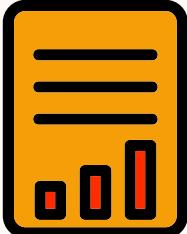




What should happen now?

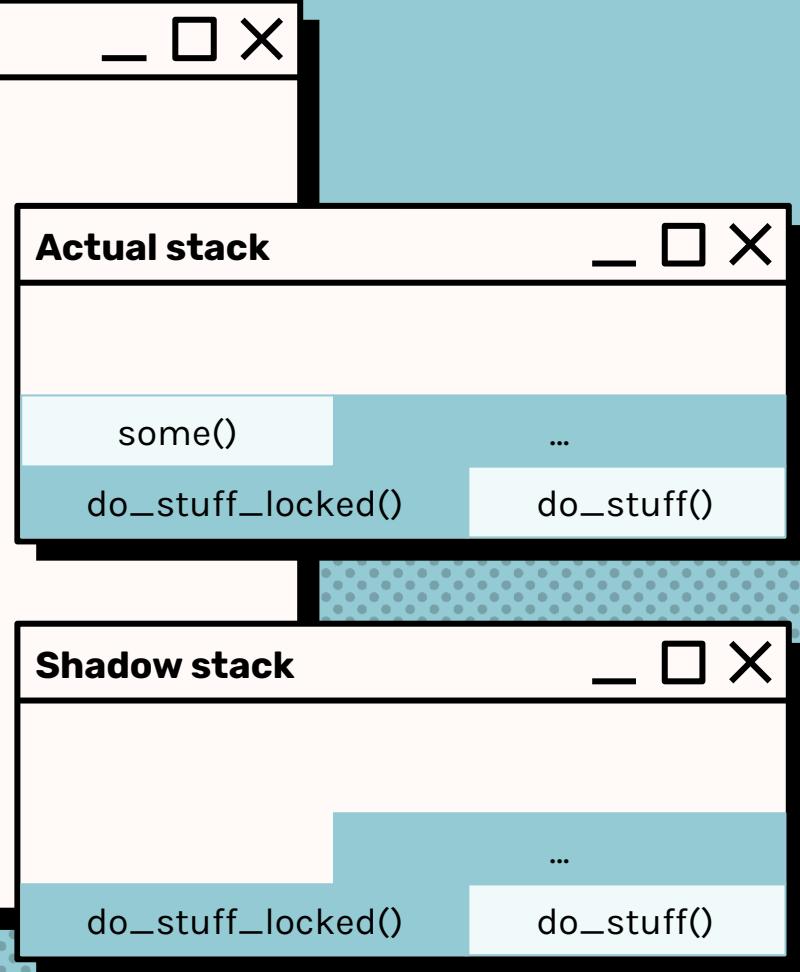
To heal ...

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



... and continue

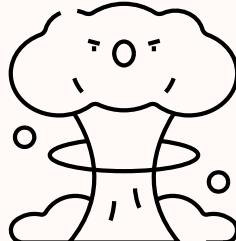
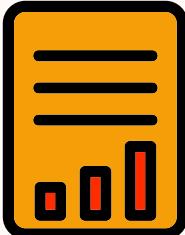
```
do_stuff()  
↳ do_stuff_locked()  
↳ some()
```





... 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()



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

gitlab.freedesktop.org/gstreamer/gst-plugins-bad/-/commit/026f1550b1fa8e13a8fb44eeebf292f382ad1

GStreamer > gst-plugins-bad > Commits > [026f1550](#)

Commit [026f1550](#) authored 2 years ago by  Andrew West Committed by Tim-Philipp Müller 2 years ago

[Browse files](#) [Options ↻](#)

codecparsers: h264parser: guard against ref_pic_markings overflow

→ parent [5128cbd5](#)  1.16 •••

1 merge request [#1705](#) Backport of "codecparsers: h264parser: guard against ref_pic_markings overflow" into 1.16

 Pipeline #[214798](#) waiting for manual action with stages     in 9 seconds

[Changes](#) 1 [Pipelines](#) 1

Showing 1 changed file ▾ with 7 additions and 3 deletions

[Hide whitespace changes](#) [Inline](#) [Side-by-side](#) [View file @026f1550](#)

gst-lib/gst/codecparsers/gsth264parserc

```
@@ -712,13 +712,17 @@ gst_h264_slice_parse_dec_ref_pic_marking (GstH264SliceHdr * slice,
```

712 712 dec_ref_pic_m->n_ref_pic_marking = 0;

713 713 while (1)

714 714 refpicmarking =

715 715 &dec_ref_pic_m->ref.pic_marking[dec_ref.pic.m->n_ref.pic_marking];

716 716 READ_U8 (nn, mem_mngt_ctrl_op);

717 716 if (mem_ngmt_ctrl_op == 0)

718 718 break;

719 720 if (dec_ref.pic.m->n_ref.pic_marking >=

720 720 G_N_ELEMENTS (dec_ref.pic.m->ref.pic_marking))

721 721 goto error;

722 722 refpicmarking =

723 723 &dec_ref.pic_m->ref.pic_marking[dec_ref.pic.m->n_ref.pic_marking];

724 724 refpicmarking->memory_management_control_operation = mem_mngt_ctrl_op;

725 725

726 726 if (mem_ngmt_ctrl_op == 1 || mem_mngt_ctrl_op == 3)

727 727

728 728

729 729

730 730

731 731

732 732

733 733

734 734

735 735

736 736

737 737

738 738

739 739

740 740

741 741

742 742

743 743

744 744

745 745

746 746

747 747

748 748

749 749

750 750

751 751

752 752

753 753

754 754

755 755

756 756

757 757

758 758

759 759

760 760

761 761

762 762

763 763

764 764

765 765

766 766

767 767

768 768

769 769

770 770

771 771

772 772

773 773

774 774

775 775

776 776

777 777

778 778

779 779

780 780

781 781

782 782

783 783

784 784

785 785

786 786

787 787

788 788

789 789

790 790

791 791

792 792

793 793

794 794

795 795

796 796

797 797

798 798

799 799

800 800

801 801

802 802

803 803

804 804

805 805

806 806

807 807

808 808

809 809

810 810

811 811

812 812

813 813

814 814

815 815

816 816

817 817

818 818

819 819

820 820

821 821

822 822

823 823

824 824

825 825

826 826

827 827

828 828

829 829

830 830

831 831

832 832

833 833

834 834

835 835

836 836

837 837

838 838

839 839

840 840

841 841

842 842

843 843

844 844

845 845

846 846

847 847

848 848

849 849

850 850

851 851

852 852

853 853

854 854

855 855

856 856

857 857

858 858

859 859

860 860

861 861

862 862

863 863

864 864

865 865

866 866

867 867

868 868

869 869

870 870

871 871

872 872

873 873

874 874

875 875

876 876

877 877

878 878

879 879

880 880

881 881

882 882

883 883

884 884

885 885

886 886

887 887

888 888

889 889

890 890

891 891

892 892

893 893

894 894

895 895

896 896

897 897

898 898

899 899

900 900

901 901

902 902

903 903

904 904

905 905

906 906

907 907

908 908

909 909

910 910

911 911

912 912

913 913

914 914

915 915

916 916

917 917

918 918

919 919

920 920

921 921

922 922

923 923

924 924

925 925

926 926

927 927

928 928

929 929

930 930

931 931

932 932

933 933

934 934

935 935

936 936

937 937

938 938

939 939

940 940

941 941

942 942

943 943

944 944

945 945

946 946

947 947

948 948

949 949

950 950

951 951

952 952

953 953

954 954

955 955

956 956

957 957

958 958

959 959

960 960

961 961

962 962

963 963

964 964

965 965

966 966

967 967

968 968

969 969

970 970

971 971

972 972

973 973

974 974

975 975

976 976

977 977

978 978

979 979

980 980

981 981

982 982

983 983

984 984

985 985

986 986

987 987

988 988

989 989

990 990

991 991

992 992

993 993

994 994

995 995

996 996

997 997

998 998

999 999

1000 1000

1001 1001

1002 1002

1003 1003

1004 1004

1005 1005

1006 1006

1007 1007

1008 1008

1009 1009

1010 1010

1011 1011

1012 1012

1013 1013

1014 1014

1015 1015

1016 1016

1017 1017

1018 1018

1019 1019

1020 1020

1021 1021

1022 1022

1023 1023

1024 1024

1025 1025

1026 1026

1027 1027

1028 1028

1029 1029

1030 1030

1031 1031

1032 1032

1033 1033

1034 1034

1035 1035

1036 1036

1037 1037

1038 1038

1039 1039

1040 1040

1041 1041

1042 1042

1043 1043

1044 1044

1045 1045

1046 1046

1047 1047

1048 1048

1049 1049

1050 1050

1051 1051

1052 1052

1053 1053

1054 1054

1055 1055

1056 1056

1057 1057

1058 1058

1059 1059

1060 1060

1061 1061

1062 1062

1063 1063

1064 1064

1065 1065

1066 1066

1067 1067

1068 1068

1069 1069

1070 1070

1071 1071

1072 1072

1073 1073

1074 1074

1075 1075

1076 1076

1077 1077

1078 1078

1079 1079

1080 1080

1081 1081

1082 1082

1083 1083

1084 1084

1085 1085

1086 1086

1087 1087

1088 1088

1089 1089

1090 1090

1091 1091

1092 1092

1093 1093

1094 1094

1095 1095

1096 1096

1097 1097

1098 1098

1099 1099

1100 1100

1101 1101

1102 1102

1103 1103

1104 1104

1105 1105

1106 1106

1107 1107

1108 1108

1109 1109

1110 1110

1111 1111

1112 1112

1113 1113

1114 1114

1115 1115

1116 1116

1117 1117

1118 1118

1119 1119

1120 1120

1121 1121

1122 1122

1123 1123

1124 1124

1125 1125

1126 1126

1127 1127

1128 1128

1129 1129

1130 1130

1131 1131

1132 1132

1133 1133

1134 1134

1135 1135

1136 1136

1137 1137

1138 1138

1139 1139

1140 1140

1141 1141

1142 1142

1143 1143

1144 1144

1145 1145

1146 1146

1147 1147

1148 1148

1149 1149

1150 1150

1151 1151

1152 1152

1153 1153

1154 1154

1155 1155

1156 1156

1157 1157

1158 1158

1159 1159

1160 1160

1161 1161

1162 1162

1163 1163

1164 1164

1165 1165

1166 1166

1167 1167

1168 1168

1169 1169

1170 1170

1171 1171

1172 1172

1173 1173

1174 1174

1175 1175

1176 1176

1177 1177

1178 1178

1179 1179

1180 1180

1181 1181

1182 1182

1183 1183

1184 1184

1185 1185

1186 1186

1187 1187

1188 1188

1189 1189

1190 1190

1191 1191

1192 1192

1193 1193

1194 1194

1195 1195

1196 1196

1197 1197

1198 1198

1199 1199

1200 1200

1201 1201

1202 1202

1203 1203

1204 1204

1205 1205

1206 1206

1207 1207

1208 1208

1209 1209

1210 1210

1211 1211

1212 1212

1213 1213

1214 1214

1215 1215

1216 1216

1217 1217

1218 1218

1219 1219

1220 1220

1221 1221

1222 1222

1223 1223

1224 1224

1225 1225

1226 1226

1227 1227

1228 1228

1229 1229

1230 1230

1231 1231

1232 1232

1233 1233

1234 1234

1235 1235

1236 1236

1237 1237

1238 1238

1239 1239

1240 1240

1241 1241

1242 1242

1243 1243

1244 1244

1245 1245

1246 1246

1247 1247

1248 1248

1249 1249

1250 1250

1251 1251

1252 1252

1253 1253

1254 1254

1255 1255

1256 1256

1257 1257

1258 1258

1259 1259

1260 1260

1261 1261

1262 1262

1263 1263

1264 1264

1265 1265

1266 1266

1267 1267

1268 1268

1269 1269

1270 1270

1271 1271

1272 1272

1273 1273

1274 1274

1275 1275

1276 1276

1277 1277

1278 1278

1279 1279

1280 1280

1281 1281

1282 1282

1283 1283

1284 1284

1285 1285

1286 1286

1287 1287

1288 1288

1289 1289

1290 1290

1291 1291

1292 1292

1293 1293

1294 1294

1295 1295

1296 1296

1297 1297

1298 1298

1299 1299

1300 1300

1301 1301

1302 1302

1303 1303

1304 1304

1305 1305

1306 1306

1307 1307

1308 1308

1309 1309

1310 1310

1311 1311

1312 1312

1313 1313

1314 1314

1315 1315

1316 1316

1317 1317

1318 1318

1319 1319

1320 1320

1321 1321

1322 1322

1323 1323

1324 1324

1325 1325

1326 1326

1327 1327

1328 1328

1329 1329

1330 1330

1331 1331

1332 1332

1333 1333

1334 1334

1335 1335

1336 1336

1337 1337

1338 1338

1339 1339

1340 1340

1341 1341

1342 1342

1343 1343

1344 1344

1345 1345

1346 1346

1347 1347

1348 1348

1349 1349

1350 1350

1351 1351

1352 1352

1353 1353

1354 1354

1355 1355

1356 1356

1357 1357

1358 1358

1359 1359

1360 1360

1361 1361

1362 1362

1363 1363

1364 1364

1365 1365

1366 1366

1367 1367

1368 1368

1369 1369

1370 1370

1371 1371

1372 1372

1373 1373

1374 1374

1375 1375

1376 1376

1377 1377

1378 1378

1379 1379

1380 1380

1381 1381

1382 1382

1383 1383

1384 1384

1385 1385

1386 1386

1387 1387

1388 1388

1389 1389

1390 1390

1391 1391

1392 1392

1393 1393

1394 1394

1395 1395

1396 1396

1397 1397

1398 1398

1399 1399

1400 1400

1401 1401

1402 1402

1403 1403

1404 1404

1405 1405

1406 1406

1407 1407

1408 1408

1409 1409

1410 1410

1411 1411

1412 1412

1413 1413

1414 1414

1415 1415

1416 1416

1417 1417

1418 1418

1419 1419

1420 1420

1421 1421

1422 1422

1423 1423

1424 1424

1425 1425

1426 1426

1427 1427

1428 1428

1429 1429

1430 1430

1431 1431

1432 1432

1433 1433

1434 1434

1435 1435

1436 1436

1437 1437

1438 1438

1439 1439

1440 1440

1441 1441

1442 1442

1443 1443

1444 1444

1445 1445

1446 1446

1447 1447

1448 1448

1449 1449

1450 1450

1451 1451

1452 1452

1453 1453

1454 1454

1455 1455

1456 1456

1457 1457

1458 1458

1459 1459

1460 1460

1461 1461

1462 1462

1463 1463

1464 1464

1465 1465

1466 1466

1467 1467

1468 1468

1469 1469

1470 1470

1471 1471

1472 1472

1473 1473

1474 1474

1475 1475

1476 1476

1477 1477

1478 1478

1479 1479

1480 1480

1481 1481

1482 1482

1483 1483

1484 1484

1485 1485

1486 1486

1487 1487

1488 1488

1489 1489

1490 1490

1491 1491

1492 1492

1493 1493

1494 1494

1495 1495

1496 1496

1497 1497

1498 1498

1499 1499

1500 1500

1501 1501

1502 1502

1503 1503

1504 1504

1505 1505

1506 1506

1507 1507

1508 1508

1509 1509

1510 1510

1511 1511

1512 1512

1513 1513

1514 1514

1515 1515

1516 1516

1517 1517

1518 1518

1519 1519

1520 1520

1521 1521

1522 1522

1523 1523

1524 1524

1525 1525

1526 1526

1527 1527

1528 1528

1529 1529

1530 1530

1531 1531

1532 1532

1533 1533

1534 1534

1535 1535

1536 1536

1537 1537

1538 1538

1539 1539

1540 1540

1541 1541

1542 1542

1543 1543

1544 1544

1545 1545

1546 1546

1547 1547

1548 1548

1549 1549

1550 1550

1551 1551

1552 1552

1553 1553

1554 1554

1555 1555

1556 1556

1557 1557

1558 1558

1559 1559

1560 1560

1561 1561

1562 1562

1563 1563

1564 1564

1565 1565

1566 1566

1567 1567

1568 1568

1569 1569

1570 1570

1571 1571

1572 1572

1573 1573

1574

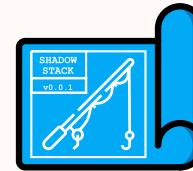
Shadow Stack maturity



Fish

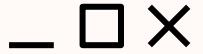


Rod



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()

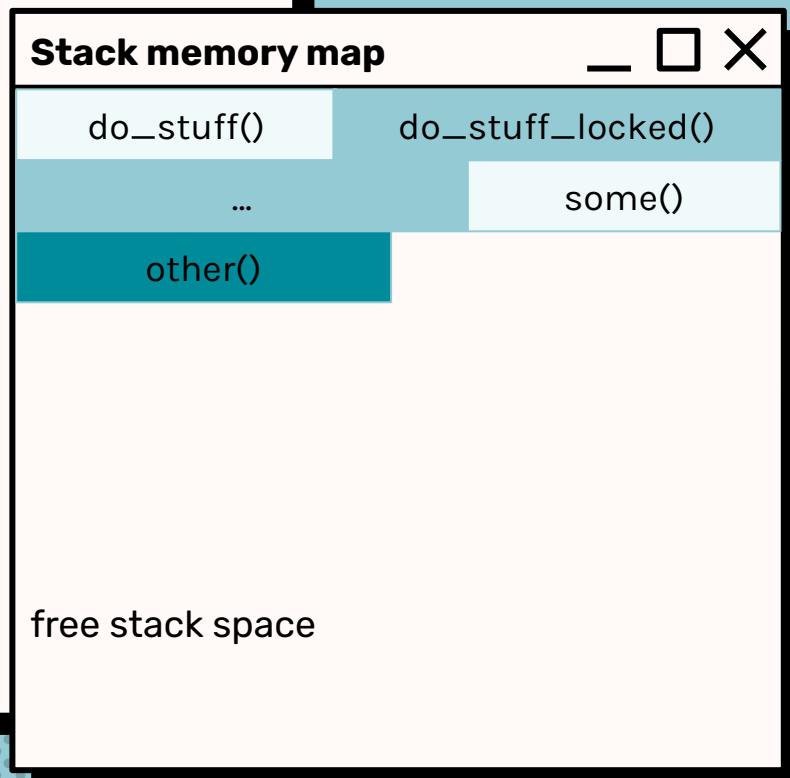
free stack space



Up-growing?

Call trace:

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

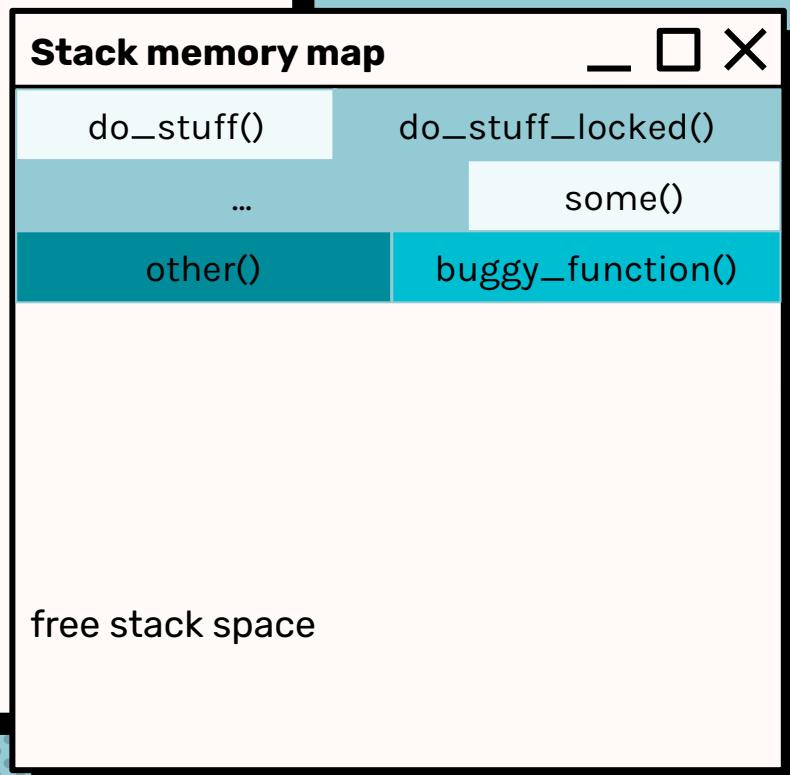




Up-growing?

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

do_stuff()

do_stuff_locked()

...

some()

other()

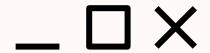
buggy_function()

!!

free stack space

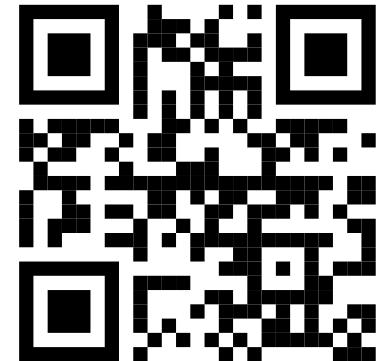


Can we switch?



Q & A

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



FEEDBACK

99

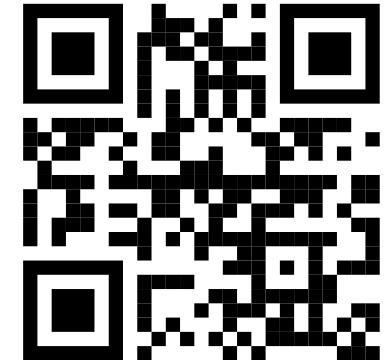


Thank you

Bartosz Moczulski

will return in

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



FEEDBACK