Reliable and Fast DWARF-based Stack Unwinding

Théophile Bastian
Stephen Kell
Francesco Zappa Nardelli

ENS Paris, University of Kent, Inria

Webpage (incl. slides)

`https://huit.re/frdwarf`

Funding
ONR VerticA
Google Research Fellowship

```
$ ./a.out
Segmentation fault.
```

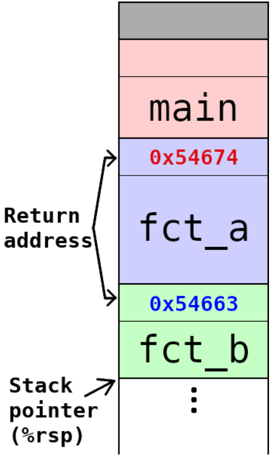
```
$ ./a.out
Segmentation fault.

(gdb) backtrace
#0   0x54625 in fct_b
#1   0x54663 in fct_a
#2   0x54674 in main
```

```
$ ./a.out
Segmentation fault.

(gdb) backtrace
#0   0x54625 in fct_b
#1   0x54663 in fct_a
#2   0x54674 in main
```

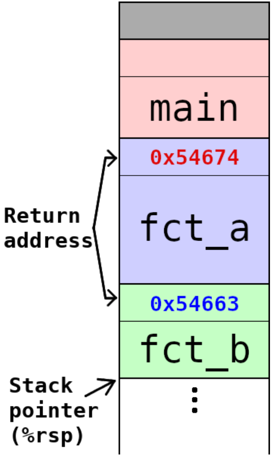
How does it work?

```
$ ./a.out
Segmentation fault.

(gdb) backtrace
#0    0x54625 in fct_b
#1    0x54663 in fct_a
#2    0x54674 in main
```

How does it work?

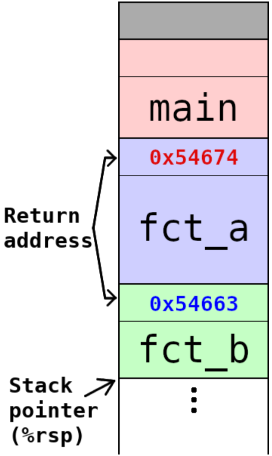


How do we get the return address?



How do we get the return address?

What if we only have %rsp?



DWARF unwinding data

PC	CFA	rbx	rbp	r12	r13	r14	r15	ra
0084950	rsp+8	u	u	u	u	u	u	c-8
0084952	rsp+16	u	u	u	u	u	c-16	c-8
0084954	rsp+24	u	u	u	u	c-24	c-16	c-8
0084956	rsp+32	u	u	u	c-32	c-24	c-16	c-8
0084958	rsp+40	u	u	c-40	c-32	c-24	c-16	c-8
0084959	rsp+48	u	c-48	c-40	c-32	c-24	c-16	c-8
008495a	rsp+56	c-56	c-48	c-40	c-32	c-24	c-16	c-8
0084962	rsp+64	c-56	c-48	c-40	c-32	c-24	c-16	c-8
0084a19	rsp+56	c-56	c-48	c-40	c-32	c-24	c-16	c-8
0084a1d	rsp+48	c-56	c-48	c-40	c-32	c-24	c-16	c-8
0084a1e	rsp+40	c-56	c-48	c-40	c-32	c-24	c-16	c-8

DWARF unwinding data

PC	CFA	rbx	rbp	r12	r13	r14	r15	ra
0084950	rsp+8	u	u	u	u	u	u	c-8
0084952	rsp+16	u	u	u	u	u	c-16	c-8
0084954	rsp+24	u	u	u	u	c-24	c-16	c-8
0084956	rsp+32	u	u	u	c-32	c-24	c-16	c-8
0084958	rsp+40	u	u	c-40	c-32	c-24	c-16	c-8
0084959	rsp+48	u	c-48	c-40	c-32	c-24	c-16	c-8
008495a	rsp+56	c-56	c-48	c-40	c-32	c-24	c-16	c-8
0084962	rsp+64	c-56	c-48	c-40	c-32	c-24	c-16	c-8
0084a19	rsp+56	c-56	c-48	c-40	c-32	c-24	c-16	c-8
0084a1d	rsp+48	c-56	c-48	c-40	c-32	c-24	c-16	c-8
0084a1e	rsp+40	c-56	c-48	c-40	c-32	c-24	c-16	c-8

For each instruction. . .
(identified by its
program counter)

DWARF unwinding data

PC	CFA	rbx	rbp	r12	r13	r14	r15	ra
0084950	rsp+8	u	u	u	u	u	u	c-8
0084952	rsp+16	u	u	u	u	u	c-16	c-8
0084954	rsp+24	u	u	u	u	c-24	c-16	c-8
0084956	rsp+32	u	u	u	c-32	c-24	c-16	c-8
0084958	rsp+40	u	u	c-40	c-32	c-24	c-16	c-8
0084959	rsp+48	u	c-48	c-40	c-32	c-24	c-16	c-8
008495a	rsp+56	c-56	c-48	c-40	c-32	c-24	c-16	c-8
0084962	rsp+64	c-56	c-48	c-40	c-32	c-24	c-16	c-8
0084a19	rsp+56	c-56	c-48	c-40	c-32	c-24	c-16	c-8
0084a1d	rsp+48	c-56	c-48	c-40	c-32	c-24	c-16	c-8
0084a1e	rsp+40	c-56	c-48	c-40	c-32	c-24	c-16	c-8

For each instruction. . .
(identified by its
program counter)

**. . . an expression
to compute its
return address
location on the stack**

The real DWARF

```
30 24 34 FDE pc=004020..004040
  DW_CFA_def_cfa_offset: 16
  DW_CFA_advance_loc: 6 to 0000000000004026
  DW_CFA_def_cfa_offset: 24
  DW_CFA_advance_loc: 10 to 0000000000004030
  DW_CFA_def_cfa_expression (DW_OP_breg7 (rsp): 8;
   DW_OP_breg16 (rip): 0; DW_OP_lit15; DW_OP_and;
   DW_OP_lit11; DW_OP_ge; DW_OP_lit3; DW_OP_shl;
   DW_OP_plus)
  [...]
```

The real DWARF

```
30 24 34 FDE pc=004020..004040
  DW_CFA_def_cfa_offset: 16
  DW_CFA_advance_loc: 6 to 0000000000004026
  DW_CFA_def_cfa_offset: 24
  DW_CFA_advance_loc: 10 to 0000000000004030
  DW_CFA_def_cfa_expression (DW_OP_breg7 (rsp): 8;
   DW_OP_breg16 (rip): 0; DW_OP_lit15; DW_OP_and;
   DW_OP_lit11; DW_OP_ge; DW_OP_lit3; DW_OP_shl;
   DW_OP_plus)
  [...]
```

⟶ bytecode for a Turing-complete stack machine

⟶ which is interpreted on demand at runtime
to reconstruct the table

What does this imply?

Your compiler generates code for two machines: your processor and the DWARF VM.

```
$ gcc -S foo.c
```

```
main:
.cfi_startproc
pushq   %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq    %rsp, %rbp
.cfi_def_cfa_register 6
subq    $32, %rsp
movl    %edi, -20(%rbp)
movq    %rsi, -32(%rbp)
```

.cfi_*: inline DWARF!

What does this imply?

Your compiler generates code for two machines: your processor and the DWARF VM.

```
$ gcc -S foo.c
```

```
main:
.cfi_startproc
pushq   %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq    %rsp, %rbp
.cfi_def_cfa_register 6
subq    $32, %rsp
movl    %edi, -20(%rbp)
movq    %rsi, -32(%rbp)
```

.cfi_*: inline DWARF!

$\Longrightarrow$ Cumbersome to generate for the compiler

- $\rightsquigarrow$ might do it wrong
- $\rightsquigarrow$ might not do it at all

$\Longrightarrow$ If you write inline asm, you must write inline DWARF!

```
    .section    .eh_frame,"a",@progbits
5:  .long    7f-6f        # Length of Common Information Entry
6:  .long    0x0          # CIE Identifier Tag
    .byte    0x1          # CIE Version
    .ascii "zR\\0"        # CIE Augmentation
    .uleb128 0x1          # CIE Code Alignment Factor
    .sleb128 -4           # CIE RA Column
    .byte    0x8          # Augmentation size
    .uleb128 0x1          # FDE Encoding (pcrel sdata4)
    .byte    0x1b         # DW_CFA_def_cfa
    .byte    0xc
    .uleb128 0x4
    .uleb128 0x0
    .align 4
7:  .long    17f-8f       # FDE Length
8:  .long    8b-5b        # FDE CIE offset
    .long    1b-.         # FDE initial location
    .long    4b-1b        # FDE address range
    .uleb128 0x0          # Augmentation size
    .byte    0x16         # DW_CFA_val_expression
    .uleb128 0x8
    .uleb128 10f-9f
9:  .byte    0x78         # DW_OP_breg8
    .sleb128 3b-1b
```

```
        .section    .eh_frame,"a",@progbits
5:      .long    7f-6f        # Length of Common Information Entry
6:      .long    0x0          # CIE Identifier Tag
        .byte    0x1          # CIE Version
        .ascii "zR\\0"        # CIE Augmentation
        .uleb128 0x1          # CIE Code Alignment Factor
        .sleb128
        .byte
        .uleb128                                              a4)
        .byte
        .byte
        .uleb128
        .uleb128
        .align 4
7:      .long
8:      .long
        .long
        .long
        .uleb128
        .byte
        .uleb128 0x8
        .uleb128 10f-9f
9:      .byte    0x78         # DW_OP_breg8
        .sleb128 3b-1b
```

In glibc, lowlevellock.h:
off by one error in
unwinding data.

```
(gdb) backtrace
#0 0x406c2c in _L_lock_19
#1 0x406c2c in _L_lock_19
#2 0x4069c6 in abort
#3 0x401017 in main
```

```
    .section    .eh_frame,"a",@progbits
5:
6:
```

Complex & slow

```
    .uleb128 0x1        # CIE Code Alignment Factor
    .sleb128 -4         # CIE RA Column
    .byte    0x8        # Augmentation size
    .uleb128 0x1        # FDE Encoding (pcrel sdata4)
    .byte    0x1b       # DW_CFA_def_cfa
    .byte    0xc
    .uleb128 0x4
    .uleb128 0x0
    .align 4
7:  .long    17f-8f     # FDE Length
8:  .long    8b-5b      # FDE CIE offset
    .long    1b-.       # FDE initial location
    .long    4b-1b      # FDE address range
    .uleb128 0x0        # Augmentation size
    .byte    0x16       # DW_CFA_val_expression
    .uleb128 0x8
    .uleb128 10f-9f
9:  .byte    0x78       # DW_OP_breg8
    .sleb128 3b-1b
```

```
.section	.eh_frame,"a",@progbits
```

Complex & slow

```
.uleb128 0x1          # CIE Code Alignment Factor
.sleb128 -4           # CIE RA Column
```

Pervasive:

relied upon by profilers, debuggers, aaand. . .

```
.section    .eh_frame,"a",@progbits
```

Complex & slow

```
.uleb128 0x1         # CIE Code Alignment Factor
.sleb128 -4          # CIE RA Column
```

Pervasive:

relied upon by profilers, debuggers, aaand. . . C++ exceptions.

$\rightsquigarrow$ **not only for debuggers!**

"Sorry, but last time was too f... painful. The whole (and only) point of unwinders is to make debugging easy when a bug occurs. But the dwarf unwinder had bugs itself, or our dwarf information had bugs, and in either case it actually turned several trivial bugs into a total undebuggable hell."

— Linus Torvalds, 2012

"Sorry, but last time was too f... painful. The whole (and only) point of unwinders is to make debugging easy when a bug occurs. But the dwarf unwinder had bugs itself, or our dwarf information had bugs, and in either case it actually turned several trivial bugs into a total undebuggable hell."

— Linus Torvalds, 2012

This is where we still are!

"Sorry, but last time was too f…painful. The whole (and only) point of unwinders is to make debugging easy when a bug occurs. But the dwarf unwinder had bugs itself, or our dwarf information had bugs, and in either case it actually turned several trivial bugs into a total undebuggable hell."

"If you can mathematically prove that the unwinder is correct — even in the presence of bogus and actively incorrect unwinding information — and never ever follows a bad pointer, I'll reconsider."

— Linus Torvalds, 2012

Correctness by construction:
synthesis of unwinding tables

```
<foo>:
  push   %r15
  push   %r14
  mov    $0x3,%eax
  push   %r13
  push   %r12
  push   %rbp
  push   %rbx
  sub    $0x68,%rsp
  add    $0x68,%rsp
  pop    %rbx
```

<foo>:		**CFA**	**ra**
push	%r15	rsp+8	c-8
push	%r14	rsp+16	c-8
mov	$0x3,%eax	rsp+24	c-8
push	%r13	rsp+24	c-8
push	%r12	rsp+32	c-8
push	%rbp	rsp+40	c-8
push	%rbx	rsp+48	c-8
sub	$0x68,%rsp	rsp+56	c-8
add	$0x68,%rsp	rsp+160	c-8
pop	%rbx	rsp+56	c-8

```
<foo>:
  push   %r15
  push   %r14
  mov    $0x3,%eax
  push   %r13
  pop    %rbx
```

CFA	ra
rsp+8	c-8
rsp+16	c-8
rsp+24	c-8
rsp+24	c-8
rsp+56	c-8

Assumptions

- the compiler generated the unwinding data
- we have a reliable DWARF interpreter

<foo>:

		CFA	ra
push	%r15	rsp+8	c-8
push	%r14	rsp+16	c-8
mov	$0x3,%eax	rsp+24	c-8
push	%r13	rsp+24	c-8
push	%r12	rsp+32	c-8
push	%rbp	rsp+40	c-8
push	%rbx	rsp+48	c-8
sub	$0x68,%rsp	rsp+56	c-8
add	$0x68,%rsp	rsp+160	c-8
pop	%rbx	rsp+56	c-8

Upon function call, ra = *(%rsp)

<foo>:

		CFA	ra
push	%r15	rsp+8	c-8
push	%r14	rsp+16	c-8
mov	$0x3,%eax	rsp+24	c-8
push	%r13	rsp+24	c-8
push	%r12	rsp+32	c-8
push	%rbp	rsp+40	c-8
push	%rbx	rsp+48	c-8
sub	$0x68,%rsp	rsp+56	c-8
add	$0x68,%rsp	rsp+160	c-8
pop	%rbx	rsp+56	c-8

push decreases %rsp by 8: $ra = *(\%rsp + 8)$

```
<foo>:                        CFA      ra
  push   %r15                 rsp+8    c-8
  push   %r14                 rsp+16   c-8
  mov    $0x3,%eax            rsp+24   c-8
  push   %r13                 rsp+24   c-8
  push   %r12                 rsp+32   c-8
  push   %rbp                 rsp+40   c-8
  push   %rbx                 rsp+48   c-8
  sub    $0x68,%rsp           rsp+56   c-8
  add    $0x68,%rsp           rsp+160  c-8
  pop    %rbx                 rsp+56   c-8
```

and again: $ra = *(\%rsp + 16)$

<foo>:

		CFA	ra
push	%r15	rsp+8	c-8
push	%r14	rsp+16	c-8
mov	$0x3,%eax	rsp+24	c-8
push	%r13	rsp+24	c-8
push	%r12	rsp+32	c-8
push	%rbp	rsp+40	c-8
push	%rbx	rsp+48	c-8
sub	$0x68,%rsp	rsp+56	c-8
add	$0x68,%rsp	rsp+160	c-8
pop	%rbx	rsp+56	c-8

This mov leaves %rsp untouched:

$ra = *(\%rsp + 16)$

```
<foo>:
  push   %r15
  push   %r14
  mov    $0x3,%eax
  push   %r13
  push   %r12
  push   %rbp
  push   %rbx
  sub    $0x68,%rsp
  add    $0x68,%rsp
  pop    %rbx
```

CFA	**ra**
rsp+8	c-8
rsp+16	c-8
rsp+24	c-8
rsp+24	c-8
rsp+32	c-8
rsp+40	c-8
rsp+48	c-8
rsp+56	c-8
rsp+160	c-8
rsp+56	c-8

The unwinding table captures an abstract execution of the code...

```
<foo>:
  push  %r15
  push  %r14
  mov   $0x3,%eax
  push  %r13
  push  %r12
  push  %rbp
  push  %rbx
  sub   $0x68,%rsp
  add   $0x68,%rsp
  pop   %rbx
```

CFA	ra
rsp+8	c-8
rsp+16	c-8
rsp+24	c-8
rsp+24	c-8
rsp+32	c-8
rsp+40	c-8
rsp+48	c-8
rsp+56	c-8
rsp+160	c-8
rsp+56	c-8

…and thus is redundant with the binary.

Synthesis strategy

- Upon entering a function, we know

$$\mathsf{CFA} = \%rsp - 8 \qquad \mathsf{RA} = \mathsf{CFA} + 8$$

- The semantics of each instruction specifies how it changes the CFA.
  - Heuristic to decide whether we index with %rbp or %rsp
- With a symbolic execution with an abstract semantics, we can synthesize the unwinding table line by line.
- Control flow: forward data-flow analysis
- The fixpoints are immediate, cf article

Implemented on top of CMU's BAP

Demo time!

Unwinding data is slo

Unwinding data is sloo

Unwinding data is slooo

Unwinding data is sloooo

Unwinding data is slooooo

Unwinding data is slooooooo

Unwinding data is slooooooo

Unwinding data is slooooooooo

Unwinding data is slooooooooow.

Unwinding data is slooooooow.

So much that `perf` cannot unwind online!
It must copy to disk the whole call stack every few instants
and analyze it later at report time!

Unwinding data compilation

```
30 24 34 FDE pc=004020..004040
  DW_CFA_def_cfa_offset: 16
  DW_CFA_advance_loc: 6 to 0000000000004026
  DW_CFA_def_cfa_offset: 24
  DW_CFA_advance_loc: 10 to 0000000000004030
  DW_CFA_def_cfa_expression (DW_OP_breg7 (rsp): 8;
    DW_OP_breg16 (rip): 0; ...)
```

```
30 24 34 FDE pc=004020..004040
  DW_CFA_def_cfa_offset: 16
  DW_CFA_advance_loc: 6 to 0000000000004026
  DW_CFA_def_cfa_offset: 24
  DW_CFA_advance_loc: 10 to 0000000000004030
  DW_CFA_def_cfa_expression (DW_OP_breg7 (rsp): 8;
    DW_OP_breg16 (rip): 0; ...)
```

runtime

PC	CFA	rbx	rbp	ra
0084950	rsp+8	u	u	c-8
0084952	rsp+16	u	u	c-8
0084954	rsp+24	u	u	c-8
0084956	rsp+32	u	u	c-8

```
30 24 34 FDE pc=004020..004040
  DW_CFA_def_cfa_offset: 16
  DW_CFA_advance_loc: 6 to 0000000000004026
  DW_CFA_def_cfa_offset: 24
  DW_CFA_advance_loc: 10 to 0000000000004030
  DW_CFA_def_cfa_expression (DW_OP_breg7 (rsp): 8;
    DW_OP_breg16 (rip): 0; ...)
```

runtime

PC	CFA	rbx	rbp	ra
0084950	rsp+8	u	u	c-8
0084952	rsp+16	u	u	c-8
0084954	rsp+24	u	u	c-8
0084956	rsp+32	u	u	c-8

ahead of time

```
unwind_context_t _eh_elf(
        unwind_context_t ctx, uintptr_t pc)
{
    unwind_context_t out_ctx;
    switch(pc) {
        ...
        case 0x615 ... 0x618:
            out_ctx.rsp = ctx.rsp + 8;
            out_ctx.rip =
                *((uintptr_t*)(out_ctx.rsp - 8));
            out_ctx.flags = 3u;
            return out_ctx;
        ...
    }
}
```

gcc, AoT

ELF file: “eh_elf”

- libunwind: most common library for unwinding
- `libunwind-eh_elf`: modified version to support `eh_elfs`

⇝ Same API, almost "relink-and-play" for existing projects!

Unwinding speedup vs. libunwind:

x15 on `perf gzip`

x25 on `perf hackbench`

Space overhead vs. DWARF:

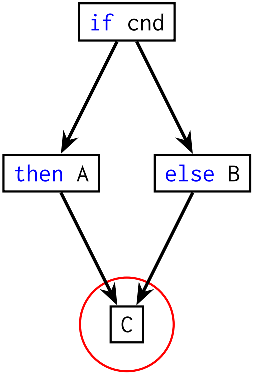
x2.6 – x3

What's next?

- Synthesis + compare = verification of unwinding data!
- Integrate synthesis into compilers & debuggers
  $\rightarrow$ support for inline assembly, fallback method, ...
- Integrate into `perf` for online unwinding
- Probably many more cool projects!

Come and chat if interested! :)

Fixpoint upon control flow merge



If eg.

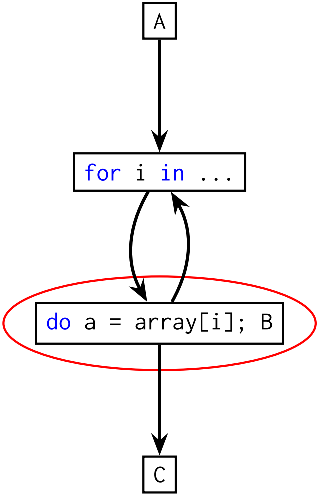
$$CFA(A) = c{-}48 \qquad CFA(B) = c{-}52$$

no possible unwinding data for C, even for the compiler!

Also, no possible clean function postlude!

$\implies$ $CFA(A) = CFA(B)$ and merge is immediate

Fixpoint upon loop control flow merge



Variable stack frame size!

We cannot hope for a simple invariant...
but the compiler cannot either.

$\Longrightarrow$ the compiler will
fallback to %rbp
even with `--fomit-frame-pointer`