Fast and Reliable
DWARF-Based Stack Unwinding

with Théophile Bastian & Stephen Kell
$ gdb ./a.out
(gdb) run
Starting program: ./a.out

Program received signal **Segmentation fault**.
0x000000000040049f in bar ()

(gdb)
$ gdb ./a.out
(gdb) run
Starting program: ./a.out

Program received signal **Segmentation fault.**
0x0000000000040049f in bar ()

(gdb) bt
  #0  0x0000000000040049f in bar ()
  #1  0x000000000004004b0 in foo ()
  #2  0x000000000004004bc in main ()
$ gdb ./a.out
(gdb) run
Starting program: ./a.out

Program received signal **Segmentation fault**.
0x0000000000040049f in bar ()

(gdb) bt
#0 0x0000000000040049f in bar ()
#1 0x00000000000400b0 in foo ()
#2 0x00000000000400bc in main ()

Backtrace Call-stack in memory
$ gdb ./a.out
(gdb) run
Starting program: ./a.out

Program received signal **Segmentation fault**.
0x000000000040049f in bar ()

(gdb) bt
#0  0x000000000040049f in bar ()
#1  0x00000000004004b0 in foo ()
#2  0x00000000004004bc in main ()

**Given a call-stack**
is there a **reliable** and **efficient** way to produce a backtrace?
DWARF unwinding tables

$ readelf -Wf a.out

<table>
<thead>
<tr>
<th>LOC</th>
<th>CFA</th>
<th>rbx</th>
<th>rbp</th>
<th>r12</th>
<th>r13</th>
<th>r14</th>
<th>r15</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000004004d0</td>
<td>rsp+8</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c-8</td>
</tr>
<tr>
<td>00000000004004d2</td>
<td>rsp+16</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>00000000004004d7</td>
<td>rsp+24</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>00000000004004dc</td>
<td>rsp+32</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>00000000004004e1</td>
<td>rsp+40</td>
<td>u</td>
<td>u</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>00000000004004e9</td>
<td>rsp+48</td>
<td>u</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>00000000004004f1</td>
<td>rsp+56</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>00000000004004fe</td>
<td>rsp+64</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>000000000040052a</td>
<td>rsp+56</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>000000000040052b</td>
<td>rsp+48</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>000000000040052c</td>
<td>rsp+40</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>000000000040052e</td>
<td>rsp+32</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>000000000040053a</td>
<td>rsp+24</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>0000000000400532</td>
<td>rsp+16</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>0000000000400534</td>
<td>rsp+8</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
</tbody>
</table>
DWARF unwinding tables

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

<table>
<thead>
<tr>
<th>LOC</th>
<th>CFA</th>
<th>rbx</th>
<th>rbp</th>
<th>r12</th>
<th>r13</th>
<th>r14</th>
<th>r15</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000004004d0</td>
<td>rsp+8</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c-8</td>
</tr>
<tr>
<td>00000000004004d2</td>
<td>rsp+16</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c-16</td>
</tr>
<tr>
<td>00000000004004d7</td>
<td>rsp+24</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>00000000004004dc</td>
<td>rsp+32</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>00000000004004e1</td>
<td>rsp+40</td>
<td>u</td>
<td>u</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>00000000004004e8</td>
<td>rsp+48</td>
<td>u</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>00000000004004e9</td>
<td>rsp+56</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>00000000004004f0</td>
<td>rsp+48</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>00000000004004f1</td>
<td>rsp+56</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>00000000004004f2</td>
<td>rsp+64</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>000000000040052a</td>
<td>rsp+56</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>000000000040052b</td>
<td>rsp+48</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>000000000040052c</td>
<td>rsp+40</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>000000000040052d</td>
<td>rsp+32</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>0000000000400530</td>
<td>rsp+24</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>0000000000400532</td>
<td>rsp+16</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>0000000000400534</td>
<td>rsp+8</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
</tbody>
</table>
**DWARF unwinding tables**

*For each instruction... (identified by the program counter)*

<table>
<thead>
<tr>
<th>LOC</th>
<th>CFA</th>
<th>rbx</th>
<th>rbp</th>
<th>r12</th>
<th>r13</th>
<th>r14</th>
<th>r15</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000000004004d0</td>
<td>rsp+8</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c–8</td>
</tr>
<tr>
<td>00000000004004d2</td>
<td>rsp+16</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c–16 c–8</td>
</tr>
<tr>
<td>00000000004004d7</td>
<td>rsp+24</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c–24 c–16 c–8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000004004dc</td>
<td>rsp+32</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c–32 c–24 c–16 c–8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000004004e1</td>
<td>rsp+40</td>
<td>u</td>
<td>u</td>
<td>c–40 c–32 c–24 c–16 c–8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000004004e9</td>
<td>rsp+48</td>
<td>u</td>
<td>c–48 c–40 c–32 c–24 c–16 c–8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000004004f1</td>
<td>rsp+56</td>
<td>c–56 c–48 c–40 c–32 c–24 c–16 c–8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000000004004f8</td>
<td>rsp+64</td>
<td>c–56 c–48 c–40 c–32 c–24 c–16 c–8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000000040052a</td>
<td>rsp+56</td>
<td>c–56 c–48 c–40 c–32 c–24 c–16 c–8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000000040052b</td>
<td>rsp+48</td>
<td>c–56 c–48 c–40 c–32 c–24 c–16 c–8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000000040052c</td>
<td>rsp+40</td>
<td>c–56 c–48 c–40 c–32 c–24 c–16 c–8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000000040052e</td>
<td>rsp+32</td>
<td>c–56 c–48 c–40 c–32 c–24 c–16 c–8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000000000400530</td>
<td>rsp+24</td>
<td>c–56 c–48 c–40 c–32 c–24 c–16 c–8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000000000400532</td>
<td>rsp+16</td>
<td>c–56 c–48 c–40 c–32 c–24 c–16 c–8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000000000400534</td>
<td>rsp+8</td>
<td>c–56 c–48 c–40 c–32 c–24 c–16 c–8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[...]
### DWARF unwinding tables

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

...specify how to compute the location on the stack of the return address

<table>
<thead>
<tr>
<th>LOC</th>
<th>CFA</th>
<th>rbx</th>
<th>rbp</th>
<th>r12</th>
<th>r13</th>
<th>r14</th>
<th>r15</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000000000000</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
<td></td>
</tr>
<tr>
<td>0000000000000001</td>
<td>rsp+16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000000000000002</td>
<td>rsp+8</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
</tbody>
</table>

Relied upon to implement **stack unwinding**

By **debuggers**
For each instruction... (identified by the program counter)

...specify how to compute the location on the stack of the return address

Relied upon to implement **stack unwinding**

By **debuggers** but also by **program analysis tools**
### DWARF unwinding tables

<table>
<thead>
<tr>
<th>LOC</th>
<th>CFA</th>
<th>rbx</th>
<th>rbp</th>
<th>r12</th>
<th>r13</th>
<th>r14</th>
<th>r15</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000000000000</td>
<td>rsp+16</td>
<td>c-36</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
<tr>
<td>0000000000000400534</td>
<td>rsp+8</td>
<td>c-56</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-16</td>
<td>c-8</td>
</tr>
</tbody>
</table>

...specify how to compute the location on the stack of the return address

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

Relied upon to implement **stack unwinding**

By **debuggers** but also by **program analysis tools**

And by the **C++ runtime** to implement **exceptions***!!!
Debug tables are not only for debugging
<table>
<thead>
<tr>
<th>ELF Header</th>
<th>Program Headers</th>
<th>Section Headers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.init</td>
<td>.eh_frame_hdr</td>
</tr>
<tr>
<td></td>
<td>.plt</td>
<td>.eh_frame</td>
</tr>
<tr>
<td></td>
<td>.text</td>
<td>.gcc_except_table</td>
</tr>
<tr>
<td></td>
<td>.fini</td>
<td>.dynamic</td>
</tr>
<tr>
<td></td>
<td>.eh_frame_hdr</td>
<td>.got</td>
</tr>
<tr>
<td></td>
<td>.eh_frame</td>
<td>.data</td>
</tr>
<tr>
<td></td>
<td>.gcc_except_table</td>
<td>.symtab</td>
</tr>
<tr>
<td></td>
<td>.dynamic</td>
<td>.strtab</td>
</tr>
</tbody>
</table>

MacOS use a different binary format, but rely on .eh_frame tables too.
MacOS use a different binary format, but rely on .eh_frame tables too.
<table>
<thead>
<tr>
<th>ELF Header</th>
<th>Program Headers</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.init</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.plt</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.text</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.fini</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.eh_frame_hdr</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.eh_frame</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.gcc_except_table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.dynamic</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.got</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.data</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.symtab</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.strtab</td>
<td></td>
</tr>
</tbody>
</table>

- Code
- DWARF Unwinding Tables
- Data for dynamic linking
- Statically allocated variables
- Symbol table

MacOS use a different binary format, but rely on .eh_frame tables too.
Unwinding Tables on disk

0000f670  14 00 00 00 00 00 00 00 01 7a 52 00 01 78 10 01 | ...........zR..x..|
0000f680  1b 0c 07 08 90 01 07 10 14 00 00 00 1c 00 00 00 | ........................................*.............................|
0000f690  90 0d ff ff 2a 00 00 00 00 00 00 00 00 00 00 00 | ..........................| .......................... |
0000f6a0  14 00 00 00 00 00 00 00 01 7a 52 00 01 78 10 01 | ...........zR..x..|
0000f6b0  1b 0c 07 08 90 01 00 00 24 00 00 00 1c 00 00 00 | ........................................$.......|
0000f6c0  10 0d ff ff 40 00 00 00 00 0e 10 46 0e 18 4a 0f | ....@.....F..J. | ....@.....F..J. |
0000f6d0  0b 77 08 80 00 3f 1a 3b 2a 33 24 22 00 00 00 00 | .w...?.;*3$".....| ....@.....F..J. |
0000f6e0  1c 00 00 00 44 00 00 00 2e 0e ff ff 07 00 00 00 | ........................................D........| .......................... |
0000f6f0  00 41 0e 10 86 02 43 0d 06 42 0c 07 08 00 00 00 | .................................. A....C..B..... | .................................. A....C..B..... |
0000f700  1c 00 00 00 64 00 00 00 15 0e ff ff 25 00 00 00 | ........................................d........%.. | .................................. A....C..` |
0000f710  00 41 0e 10 86 02 43 0d 06 60 0c 07 08 00 00 00 | .................................. A....C..` | .................................. A....C..` |
0000f720  1c 00 00 00 84 00 00 00 1a 0e ff ff 11 00 00 00 | ................................................. | .................................. A....C..` |
0000f730  00 41 0e 10 86 02 43 0d 06 4c 0c 07 08 00 00 00 | ................................................. | .................................. A....C..` |
0000f740  1c 00 00 00 a4 00 00 00 0b 0e ff ff 1a 00 00 00 | ................................................. | .................................. A....C..` |
0000f750  00 41 0e 10 86 02 43 0d 06 55 0c 07 08 00 00 00 | ................................................. | .................................. A....C..` |
0000f760  1c 00 00 00 c4 00 00 00 05 0e ff ff 1c 00 00 00 | ................................................. | .................................. A....C..` |

6-1
Unwinding Tables on disk

DWARF Debugging Information Format

Version 4

DWARF Debugging Information Format Committee
http://www.dwarfstd.org
DWARF Debug Unwinding Table

DW_CFA_def_cfa_offset: 16
DW_CFA_advance_loc: 6 to 0000000000400376
DW_CFA_def_cfa_offset: 24
DW_CFA_advance_loc: 10 to 0000000000400380
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)
[...]
DWARF Debug Unwinding

Bytecode instructions

Arbitrary expressions accessing registers and memory locations

DW_CFA_def_cfa_offset: 16
DW_CFA_advance_loc: 6 to 0000000000400376
DW_CFA_def_cfa_offset: 24
DW_CFA_advance_loc: 10 to 0000000000400380
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)

[...]
DWARF Debug Unwinding

Bytecode instructions

Interpreted on demand to inspect the call-stack

A Turing-complete stack-based machine
  can dereference memory and access values in machine registers
  — in a place where few expect it
Badly specified

Two subtly incompatible formats: debug_frame and eh_frame
Badly specified

Two subtly incompatible formats: debug_frame and eh_frame

Airs – Ian Lance Taylor

.eh_frame
January 10, 2011 at 11:12 pm · Filed under Programming

When gcc generates code that handles exceptions, it produces tables that describe how to unwind the stack. These tables are found in the .eh_frame section. The format of the .eh_frame section is very similar to the format of a DWARF .debug_frame section. Unfortunately, it is not precisely identical. I don’t know of any documentation which describes this format. The following should be read in conjunction with the relevant section of the DWARF standard, available from http://dwarfstd.org.
Badly specified

Two subtly incompatible formats: *debug_frame* and *eh_frame*
Badly specified
  Two subtly incompatible formats: `debug_frame` and `eh_frame`

Burden for the compiler to generate
  Each compiler pass must keep the table in sync with code
foo:

```assembly
    .cfi_startproc
    pushq  %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq   %rsp, %rbp
    .cfi_def_cfa_register 6
    movl   %edi, -4(%rbp)
    movl   %esi, -8(%rbp)
    movl   -4(%rbp), %edx
    movl   -8(%rbp), %eax
    addl   %edx, %eax
    popq   %rbp
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
```

$ gcc -S foo.c

9-2

9-2
gallium19.key - 27 November 2019
Badly specified

Two subtly incompatible formats: `debug_frame` and `eh_frame`

Burden for the compiler to generate

Each compiler pass must keep the table in sync with code
Badly specified
Two subtly incompatible formats: debug_frame and eh_frame

Burden for the compiler to generate
Each compiler pass must keep the table in sync with code

Potential vector for arbitrary code execution
Proof-of-concept attack built eight years ago
Exploiting the hard-working DWARF: Trojan and Exploit Techniques With No Native Executable Code

James Oakley, Sergey Bratus
Computer Science Dept.
Dartmouth College
Hanover, New Hampshire
james.oakley@alum.dartmouth.org, sergey@cs.dartmouth.edu

Each compiler pass must keep the table in sync with code.

Potential vector for arbitrary code execution
Proof-of-concept attack built eight years ago
Badly specified
  Two subtly incompatible formats: \textit{debug\_frame} and \textit{eh\_frame}

Burden for the compiler to generate
  Each compiler pass must keep the table in sync with code

Potential vector for arbitrary code execution
  Proof-of-concept attack built five years ago

Complex, Buggy, Untested
Why doesn’t the Linux Kernel rely on DWARF tables?
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, Kernel mailing list, 2012
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.*
Validation of unwinding tables
<foo>:
    push   %r15
    push   %r14
    mov    $0x3,%eax
    push   %r13
    push   %r12
    push   %rbp
    push   %rbx
    sub    $0x68,%rsp
    cmp    $0x1,%edi
    movl   $0x0,0x14(%rsp)
    je     49e4a0
    add    $0x68,%rsp
    pop    %rbx
    pop    %rbp
<foo>:
  push   %r15
  push   %r14
  mov    $0x3,%eax

Assume:
- eh_frame table has been generated by the compiler
- we can interpret the eh_frame bytecode to generate the unwinding tables

  cmp    $0x1,%edx
  movl   $0x0,0x14(%rsp)
  je     49e4a0
  add    $0x68,%rsp
  pop    %rbx
  pop    %rbp
### Assume:

- `eh_frame` table has been generated by the compiler
- We can interpret the `eh_frame` bytecode to generate the unwinding tables

```assembly
<table>
<thead>
<tr>
<th>Instruction</th>
<th>CFA</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>push %r15</td>
<td>rsp+8</td>
<td>c-8</td>
</tr>
<tr>
<td>push %r14</td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
<tr>
<td>mov $0x3,%eax</td>
<td>rsp+24</td>
<td>c-8</td>
</tr>
<tr>
<td>cmp $0x1,%eax</td>
<td>rsp+160</td>
<td>c-8</td>
</tr>
<tr>
<td>movl $0x0,0x14(%rsp)</td>
<td>rsp+160</td>
<td>c-8</td>
</tr>
<tr>
<td>je 49e4a0</td>
<td>rsp+160</td>
<td>c-8</td>
</tr>
<tr>
<td>add $0x68,%rsp</td>
<td>rsp+160</td>
<td>c-8</td>
</tr>
<tr>
<td>pop %rbx</td>
<td>rsp+56</td>
<td>c-8</td>
</tr>
<tr>
<td>pop %rbp</td>
<td>rsp+48</td>
<td>c-8</td>
</tr>
</tbody>
</table>
```
<foo>:
push %r15
push %r14
mov $0x3,%eax
push %r13
push %r12
push %rbp
push %rbx
sub $0x68,%rsp
cmp $0x1,%edi
movl $0x0,0x14(%rsp)
je 49e4a0
add $0x68,%rsp
pop %rbx
pop %rbp
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+160 c-8
rsp+160 c-8
rsp+160 c-8
rsp+56 c-8
rsp+48 c-8
When `foo` is called, before executing its first instruction:

return address is stored at `*rsp`

```
push   %r15
push   %r14
mov    $0x3,%eax
push   %r13
push   %r12
push   %rbp
push   %rbx
sub    $0x68,%rsp
```
After `push %r15`, `rsp` has been decreased by 8:

return address is stored at `*(rsp+8)`
After push %r14, rsp has been decreased by 8:

return address is stored at *(rsp+16)
<table>
<thead>
<tr>
<th>Assembler Instruction</th>
<th>CFA</th>
<th>RA</th>
</tr>
</thead>
<tbody>
<tr>
<td>push %r15</td>
<td>rsp+8</td>
<td>c-8</td>
</tr>
<tr>
<td>push %r14</td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
<tr>
<td>mov $0x3,%eax</td>
<td>rsp+24</td>
<td>c-8</td>
</tr>
<tr>
<td>push %r13</td>
<td>rsp+24</td>
<td>c-8</td>
</tr>
<tr>
<td>push %r12</td>
<td>rsp+32</td>
<td>c-8</td>
</tr>
<tr>
<td>push %rbp</td>
<td>rsp+40</td>
<td>c-8</td>
</tr>
<tr>
<td>push %rbx</td>
<td>rsp+48</td>
<td>c-8</td>
</tr>
<tr>
<td>sub $0x68,%rsp</td>
<td>rsp+56</td>
<td>c-8</td>
</tr>
</tbody>
</table>

After mov $0x3,%eax:

return address is still stored at *(rsp+16)
The unwinding table is redundant wrt the binary code

It captures some abstract execution of the code
<foo>:
  push   %r15
  push   %r14
  mov    $0x3,%eax
  push   %r13
  push   %r12
  push   %rbp
  push   %rbx
  sub    $0x68,%rsp
  cmp    $0x1,%edi
  movl   $0x0,0x14(%rsp)
  je     49e4a0
  add    $0x68,%rsp
  pop    %rbx
  pop    %rbp

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+160    c-8
  rsp+160    c-8
  rsp+160    c-8
  rsp+56     c-8
  rsp+48     c-8
<foo>:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>CFA</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>push %r15</td>
<td>rsp+8</td>
<td>c-8</td>
</tr>
<tr>
<td>push %r14</td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
<tr>
<td>mov $0x3,%eax</td>
<td>rsp+24</td>
<td>c-8</td>
</tr>
<tr>
<td>push %r13</td>
<td>rsp+24</td>
<td>c-8</td>
</tr>
<tr>
<td>push %r12</td>
<td>rsp+32</td>
<td>c-8</td>
</tr>
<tr>
<td>push %rbp</td>
<td>rsp+40</td>
<td>c-8</td>
</tr>
<tr>
<td>push %rbx</td>
<td>rsp+48</td>
<td>c-8</td>
</tr>
<tr>
<td>sub $0x68,%rsp</td>
<td>rsp+56</td>
<td>c-8</td>
</tr>
<tr>
<td>cmp $0x1,%edi</td>
<td>rsp+160</td>
<td>c-8</td>
</tr>
<tr>
<td>movl $0x0,0x14(%rsp)</td>
<td>rsp+160</td>
<td>c-8</td>
</tr>
<tr>
<td>je 49e4a0</td>
<td>rsp+160</td>
<td>c-8</td>
</tr>
<tr>
<td>add $0x68,%rsp</td>
<td>rsp+160</td>
<td>c-8</td>
</tr>
<tr>
<td>pop %rbx</td>
<td>rsp+56</td>
<td>c-8</td>
</tr>
<tr>
<td>pop %rbp</td>
<td>rsp+48</td>
<td>c-8</td>
</tr>
</tbody>
</table>
Suppose that we know that in an execution:

- the return address is stored at 0xFFFFF1158

We read %rsp and it stores 0xFFFFF1000

<table>
<thead>
<tr>
<th>Instruction</th>
<th>CFA</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>push %r15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>push %r14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mov $0x3,%eax</td>
<td></td>
<td></td>
</tr>
<tr>
<td>push %r13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>push %r12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>push %rbp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>push %rbx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sub $0x68,%rsp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmp $0x1,%edi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>movl $0x0,0x14(%rsp)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>je 49e4a0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add $0x68,%rsp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pop %rbx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pop %rbp</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```ruby
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>CFA</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;foo&gt;:</td>
<td>push   %r15</td>
<td>ra</td>
</tr>
<tr>
<td></td>
<td>push   %r14</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mov    $0x3,%eax</td>
<td></td>
</tr>
<tr>
<td></td>
<td>push   %r13</td>
<td></td>
</tr>
<tr>
<td></td>
<td>push   %r12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>push   %rbp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>push   %rbx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sub    $0x68,%rsp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cmp    $0x1,%edi</td>
<td></td>
</tr>
<tr>
<td></td>
<td>movl   $0x0,0x14(%rsp)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pop    %rbp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pop    %rbx</td>
<td></td>
</tr>
</tbody>
</table>

**Suppose** that we know that **in an execution:**
- the return address is stored at \texttt{0xFFFFF1158}

We read \texttt{%rsp} and it stores \texttt{0xFFFFF1000}

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>CFA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>push   %rbp</td>
<td>ra</td>
</tr>
<tr>
<td></td>
<td>push   %rbx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sub    $0x68,%rsp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cmp    $0x1,%edi</td>
<td></td>
</tr>
<tr>
<td></td>
<td>movl   $0x0,0x14(%rsp)</td>
<td></td>
</tr>
</tbody>
</table>

**Evaluating** the entry on the **current register values,**
should compute the **concrete address** of the return address.
Dynamic Validation of Unwinding Tables

Abstract state

stack of concrete addresses where return address are stored

Abstract instruction semantics

call: push the content of %rsp on top of the abstract state
ret: pop one value from the abstract state

Validation at each instruction

evaluate the return address eh_frame entry for ip
compare with the value in abstract state
Dynamic Validation of Unwinding Tables

Abstract state

stack of concrete addresses where return address are stored

Abstract instruction semantics

call: push the content of %rsp on top of the abstract state
ret: pop one value from the abstract state

Validation at each instruction

Evaluate return address eh_frame entry for ip  
with the value in abstract state

callee-saved registers require some care
The eh_frame_check tool

Goal: validate eh_frame entries along one execution path

**gdb:**
- step-by-step binary execution, access to processor state

**Python:**
- parsing ELF and DWARF binary code *(building on pyelftool)*
- evaluating DWARF expressions
- scripting gdb to implement the dynamic analysis
The *eh_frame_check* tool

Goal: validate *eh_frame* entries along one execution path

**gdb:**
- step-by-step binary execution, access to processor state

**Python:**
- parsing ELF and DWARF binary code (*building on pyelftool*)
- evaluating DWARF expressions
- scripting gdb to implement the dynamic analysis

Can process a few \( k \) asm instructions/sec: good for now
short a,b,g;
long c;
char d;
int e, f;

void main() {
    for (; f; f++)
        for (; e; e++)
            for (; c; c++) {
                g = a % b;
                for (; d <= 1; d++)
                    ;
            }
}
<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>CFA</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>4004e0</td>
<td>push %rbx</td>
<td>rsp+8</td>
<td>c-8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
<tr>
<td>40061d</td>
<td>pop %rbx</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40061e</td>
<td>retq</td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
</tbody>
</table>
Abstract state [ 0xFFFF1000 ]

<table>
<thead>
<tr>
<th>&lt;main&gt;:</th>
<th>CFA</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>4004e0: push %rbx</td>
<td>rsp+8</td>
<td>c-8</td>
</tr>
<tr>
<td>..</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40061d: pop %rbx</td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
<tr>
<td>40061e: retq</td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
</tbody>
</table>
### Abstract state [0xFFFF1000]

%-rsp  0xFFFF1000

#### <main>:

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>CFA</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>4004e0</td>
<td>push %rbx</td>
<td>rsp+8</td>
<td>c-8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40061d</td>
<td>pop %rbx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40061e</td>
<td>retq</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Abstract state  [ 0xFFFF1000 ]
%rsp  0xFFFF1000

<table>
<thead>
<tr>
<th></th>
<th>&lt;main&gt;:</th>
<th>CFA</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>4004e0:</td>
<td>push %rbx</td>
<td>rsp+8</td>
<td>c-8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[...]</td>
<td></td>
</tr>
<tr>
<td>40061d:</td>
<td>pop %rbx</td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
<tr>
<td>40061e:</td>
<td>retq</td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
</tbody>
</table>
Abstract state [0xFFFF1000]

%rsp  0xFFFFFF0FF8

<table>
<thead>
<tr>
<th>Code Address</th>
<th>Instruction</th>
<th>CFA</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>4004e0</td>
<td>push %rbx</td>
<td>rsp+8</td>
<td>c-8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
<tr>
<td>[..]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40061d</td>
<td>pop %rbx</td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
<tr>
<td>40061e</td>
<td>retq</td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
</tbody>
</table>
Abstract state  [ 0xFFFF1000 ]
%
rsps  0xFFFF0FF8

<table>
<thead>
<tr>
<th>&lt;main&gt;:</th>
<th>CFA</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>4004e0: push %rbx</td>
<td>rsp+8</td>
<td>c-8</td>
</tr>
<tr>
<td></td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
<tr>
<td>[...]</td>
<td>[...]</td>
<td>![Green Checkmark]</td>
</tr>
<tr>
<td>40061d: pop %rbx</td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
<tr>
<td>40061e: retq</td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
</tbody>
</table>
Abstract state [0xFFFF1000]

%rsp  0xFFFF0FF8

<table>
<thead>
<tr>
<th>&lt;main&gt;:</th>
<th>CFA</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>4004e0: push %rbx</td>
<td>rsp+8</td>
<td>c-8</td>
</tr>
<tr>
<td></td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
<tr>
<td>[...]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40061d: pop %rbx</td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
<tr>
<td>40061e: retq</td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
</tbody>
</table>
Abstract state [0xFFFF1000]

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>CFA</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rsp</td>
<td>0xFFFFF0FF8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<main>:

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>CFA</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>4004e0</td>
<td>push %rbx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[..]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40061d</td>
<td>pop %rbx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40061e</td>
<td>retq</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Abstract state  [ 0xFFFF1000 ]
%rsp      0xFFFF1000

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>CFA</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>4004e0</td>
<td>push %rbx</td>
<td>rsp+8</td>
<td>c-8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40061d</td>
<td>pop %rbx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40061e</td>
<td>retq</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Abstract state  [ 0xFFFF1000 ]

%rsp    0xFFFF1000

<main>

4004e0: push %rbx

[

40061d: pop %rbx

[..]

40061e: retq

[

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

0xFFFF1000+16-8 != 0xFFFF1000
At \(-00, -02,\) or \(-0s\). \textit{Not at \(-01\).}

<table>
<thead>
<tr>
<th>&lt;main&gt;</th>
<th>CFA</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>(4004e0): push %rbx</td>
<td>rsp+8</td>
<td>c-8</td>
</tr>
<tr>
<td></td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
<tr>
<td>[..]</td>
<td>[..]</td>
<td></td>
</tr>
<tr>
<td>(40061d): pop %rbx</td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
<tr>
<td>(40061e): retq</td>
<td>rsp+16</td>
<td>c-8</td>
</tr>
</tbody>
</table>
At -00, -02, or -0s. Not at -01.

```plaintext
<main>:
  4004e0:  push %rbx
  ...
  40061d:  pop  %rbx
  40061e:  retq

<table>
<thead>
<tr>
<th>CFA</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsp+8</td>
<td>c-8</td>
</tr>
<tr>
<td>rsp+16</td>
<td>c-8</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Just a coincidence: at -01 rbx is not saved.
Validation of unwinding tables is effective
The sad truths

Generating `eh_frame` is a *burden* for the compiler
Some compilers do not generate `eh_frame` at all
The sad truths

Generating \texttt{eh\_frame} is a \textit{burden} for the compiler

Some compilers do not generate \texttt{eh\_frame} at all

\texttt{OCaml eh\_frame} code was contributed by JaneStreet
The sad truths

Generating **eh_frame** is a *burden* for the compiler

Some compilers do not generate **eh_frame** at all

**OCaml eh_frame** code was contributed by JaneStreet

To enable binary profilers!!!
The sad truths

Generating \texttt{eh\_frame} is a \textit{burden} for the compiler.

Some compilers do not generate \texttt{eh\_frame} at all.

OCaml \texttt{eh\_frame} code was contributed by JaneStreet.

Did you ever attempt \textbf{debugging} \texttt{jit-generated assembly} gone wrong?
The sad truths

Generating `eh_frame` is a burden for the compiler

Some compilers do not generate `eh_frame` at all

OCaml `eh_frame` code was contributed by JaneStreet

Did you ever attempt debugging
jit-generated assembly gone wrong?

Or manually code unwinding instructions in inline asm?
#define LLL_STUB_UNWIND_INFO_START
".section .eh_frame,\"a\",@progbits\n"
"5:\t" ".long 7f-6f
"6:\t" ".long 0x0
  ".byte 0x1
  ".ascii \"zR\\0\"
  ".uleb128 0x1
  ".sleb128 -4
  ".byte 0x8
  ".uleb128 0x1
  ".byte 0x1b
  ".byte 0xc
  ".uleb128 0x4\n\t"
  ".uleb128 0x0\n\t"
  ".align 4\n"
"7:\t" ".long 17f-8f
"8:\t" ".long 8b-5b
  ".long 1b-.
  ".long 4b-1b
  ".uleb128 0x0
  ".byte 0x16
  ".uleb128 0x8\n\t"
  ".uleb128 10f-9f\n"
"9:\t" ".byte 0x78
  ".sleb128 3b-1b\n"

\# Length of Common Information Entry\n"
\# CIE Identifier Tag\n\t"
\# CIE Version\n\t"
\# CIE Augmentation\n\t"
\# CIE Code Alignment Factor\n\t"
\# CIE RA Column\n\t"
\# Augmentation size\n\t"
\# FDE Encoding (pcrel sdata4)\n\t"
\# DW_CFA_def_cfa\n\t"
\# FDE Length\n"
\# FDE CIE offset\n\t"
\# FDE initial location\n\t"
\# FDE address range\n\t"
\# Augmentation size\n\t"
\# DW_CFA_val_expression\n\t"
\# DW_OP_breg8\n\t"

glibc: lowlevellock.h
Despite great care, off by 1 offset error…

".byte 0x1b
".byte 0xc
 "uleb128 0x4\n\t"
 "uleb128 0x0\n\t"
 "align 4\n"
"7:\t" ".long 17f-8f
"8:\t" ".long 8b-5b
 \t".long 1b-.
 \t".long 4b-1b
 "uleb128 0x0
 "byte 0x16
 "uleb128 0x8\n\t"
 "uleb128 10f-9f\n"
"9:\t" ".byte 0x78
 "sleb128 3b-1b\n"

\textit{glibc: lowlevellock.h}
#define LLL_STUB_UNWIND_INFO_START

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

Despite great care, off by 1 offset error...

Breakpoint 2, 0x0000000000406c2c in _L_lock_19 ()
(gdb) disass
Dump of assembler code for function _L_lock_19:
=> 0x0000000000406c2c <+0>:  lea  0x2ba13d(%rip),%rdi # 0x6c0d70 <lock>
  0x0000000000406c33 <+7>:  sub   $0x80,%rsp
  0x0000000000406c3a <+14>:  callq  0x436d10 <__lll_lock_wait_private>
  0x0000000000406c3f <+19>:  add   $0x80,%rsp
  0x0000000000406c46 <+26>:  jmpq  0x4069c6 <abort+70>
End of assembler dump.
(gdb) bt
#0 0x0000000000406c2c in _L_lock_19 ()
#1 0x0000000000406c3f in _L_lock_19 ()
#2 0x00000000004069c6 in abort ()
#3 0x0000000000401017 in main () at foo1.c:4
(gdb)
Synthesis of unwinding tables
Synthesis Strategy

Upon entering a function:

\[ \text{CFA} = rsp - 8 \quad \text{RA} = \text{CFA} + 8 \]

Check the semantics of each instruction in each basic block:

- Is `rbp` used as a base pointer?
- Is `rsp` modified?
  - update the computation of CFA accordingly

Chain across basic blocks following the control-flow graph.
subq $0x10, %rsp  
... 
cmp %rax, %rbx  
jne .L5

F

T

cmp %rdx, %rbx  
je .L1

F

T

subq $0x8, %rsp  
...  
addq $0x8, $rsp  
jmp .L5

addq $0x10, %rsp  
ret
subq $0x10, %rsp   
... 
cmp %rax, %rbx   
jne .L5

F

subq $0x8, %rsp   
... 
addq $0x8, $rsp   
jmp .L5

T

cmp %rdx, %rbx   
je .L1

F

T

addq $0x10, %rsp
ret
subq $0x10, %rsp
...
cmp %rax, %rbx
jne .L5

F  rsp+24 c-8  T  rsp+24 c-8

cmp %rdx, %rbx
je .L1

subq $0x8, %rsp
...
addq $0x8, $rsp
jmp .L5

addq $0x10, %rsp
ret
subq $0x10, %rsp  
...  
cmp %rax, %rbx  
jne .L5  

F  rsp+24  c-8  
T  rsp+24  c-8  

cmp %rdx, %rbx  
je .L1  

F  rsp+24  c-8  
T  rsp+24  c-8  

subq $0x8, %rsp  
...  
addq $0x8, $rsp  
jmp .L5  

rsp+24  c-8  

addq $0x10, %rsp  
ret
Fixpoint immediate!

subq $0x10, %rsp  
...  
cmp %rax, %rbx  
jne .L5

F  
rsp+24  c-8

T  
rsp+24  c-8

cmp %rdx, %rbx  
je .L1

F  
rsp+24  c-8

T  
rsp+24  c-8

subq $0x8, %rsp  
...  
addq $0x8, $rsp  
jmp .L5

rspb+24  c-8

rspb+24  c-8
Fixpoint fallback

```c
int main() {
    read_integer(z_max);
    for(int z=0; z < z_max; z++) {
        int x[z];
        ...do something with x...
    }
}
```

Compiler relies on base pointer even if -fomit-frame-pointer is specified

<table>
<thead>
<tr>
<th>LOC</th>
<th>CFA</th>
<th>rbx</th>
<th>rbp</th>
<th>r12</th>
<th>r13</th>
<th>r14</th>
<th>r15</th>
<th>ra</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000000400526</td>
<td>rsp+8</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c-8</td>
</tr>
<tr>
<td>00000000000400527</td>
<td>rsp+16</td>
<td>u</td>
<td>c-16</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c-8</td>
</tr>
<tr>
<td>0000000000040052a</td>
<td>rbp+16</td>
<td>u</td>
<td>c-16</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>c-8</td>
</tr>
<tr>
<td>00000000000400537</td>
<td>rbp+16</td>
<td>c-56</td>
<td>c-16</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-8</td>
</tr>
<tr>
<td>000000000004005cf</td>
<td>rsp+8</td>
<td>c-56</td>
<td>c-16</td>
<td>c-48</td>
<td>c-40</td>
<td>c-32</td>
<td>c-24</td>
<td>c-8</td>
</tr>
</tbody>
</table>
Fixpoint fallback

Compiler relies on base pointer

Implementation on top of CMU's Binary Analysis Platform

**Theory:** symbolic evaluation of ASM, tracking registers

**Practice:** heuristics aware of few ASM patterns

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Offsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000000400526</td>
<td>rsp+8</td>
<td>u</td>
</tr>
<tr>
<td>00000000000400527</td>
<td>rsp+16</td>
<td>u, c-16</td>
</tr>
<tr>
<td>0000000000040052a</td>
<td>rbp+16</td>
<td>u, c-16</td>
</tr>
<tr>
<td>00000000000400537</td>
<td>rbp+16</td>
<td>c-56, c-16, c-48, c-40, c-32, c-24, c-8</td>
</tr>
<tr>
<td>000000000004005cf</td>
<td>rsp+8</td>
<td>c-56, c-16, c-48, c-40, c-32, c-24, c-8</td>
</tr>
</tbody>
</table>
Validate unwinding tables against the binary code

Synthesise unwinding tables from the binary/ASM code

Identify bugs in generation of tables

Make unwinding tables available to other systems
  - integrate with compiler's inline assembly extensions?
  - run on demand on JITted code?
Interpreting DWARF bytecode is *slooooow*…
…despite libunwind aggressive caching
Interpreting DWARF bytecode is **sloooow**…

…despite libunwind aggressive caching

The **perf** profiler cannot unwind the stack in an interrupt handler
Interpreting DWARF bytecode is sloooow…
…despite libunwind aggressive caching

The `perf` profiler cannot unwind the stack in an interrupt handler

During profiling stack copies are recorded for offline unwinding
Ahead-of-time compilation of DWARF unwind tables
DW_CFA_advance_loc: 0 to 0x615
DW_CFA_def_cfa: r7 (rsp) ofs 8
DW_CFA_offset: r16 (rip) at cfa-8
DW_CFA_advance_loc: 5 to 0x620
DW_CFA_def_cfa_offset: 48
...

DW_CFA_advance_loc: 0 to 0x615
DW_CFA_def_cfa: r7 (rsp) ofs 8
DW_CFA_offset: r16 (rip) at cfa-8
DW_CFA_advance_loc: 5 to 0x620
DW_CFA_def_cfa_offset: 48
...
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;
        ...
    }
}
eh_elfs

Dwarf unwind bytecode compiled to ASM

Each .eh_table is compiled to a separate eh_elfs .so file

```c
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;
    ...
}
```
eh_elfs

Dwarf unwind bytecode compiled to ASM
Each .eh_table is compiled to a separate eh_elfs .so file

libunwind-eh_elf

Alternative libunwind implementation based on eh_elf tables
Same API as libunwind: (almost) relink-and-play replacement
**eh_elfs**

Dwarf unwind bytecode compiled to ASM

**Implementation tricks:**

Code outlining performed at compile time

  eg. on libc, 20827 entries but only 302 *unique* entries

Explicit binary search rather than switch/case

Same API as libunwind: (almost) relink-and-play replacement
13x speedup on libunwind calls made by “perf gzip”

25x speedup on libunwind calls made by “perf hackbench”

2.5x space overhead
Wider horizons
<2><5f>: Abbrev Number: 3 (DW_TAG_formal_parameter)
  <60>   DW_AT_location : (DW_OP_fbreg: -16)
  <63>   DW_AT_name : z
  <67>   DW_AT_decl_file : 1
  <68>   DW_AT_decl_line : 1
  <69>   DW_AT_type : <0x7c>

<2><6d>: Abbrev Number: 0

<1><6e>: Abbrev Number: 4 (DW_TAG_base_type)
  <6f>   DW_AT_name : int
  <73>   DW_AT_encoding : 5 (signed)
  <74>   DW_AT_byte_size : 4

<1><7c>: Abbrev Number: 5 (DW_TAG_pointer_type)
  <7d>   DW_AT_type : <0x6e>

44-1
gallium19.key - 27 November 2019
<2><5f>: Abbrev Number: 3 (DW_TAG_formal_parameter)
   <60>   DW_AT_location    : (DW_OP_fbreg: -16)
   <63>   DW_AT_name        : z
   <67>   DW_AT_decl_file   : 1
   <68>   DW_AT_decl_line   : 1
   <69>   DW_AT_type        : <0x7c>
<2><6d>: Abbrev Number: 0

<1><6e>: Abbrev Number: 4 (DW_TAG_base_type)
   <6f>   DW_AT_name        : int
   <73>   DW_AT_encoding    : 5 (signed)
   <74>   DW_AT_byte_size   : 4

<1><7c>: Abbrev Number: 5 (DW_TAG_pointer_type)
   <7d>   DW_AT_type        : <0x6e>
<2><5f>: Abbrev Number: 3 (DW_TAG_formal_parameter)
<60> DW_AT_location : (DW_OP_fbreg: -16)
<63> DW_AT_name : z
<67> DW_AT_decl_file : 1
<68> DW_AT_decl_line : 1
<69> DW_AT_type : <0x7c>
<2><6d>: Abbrev Number: 0

<1><6e>: Abbrev Number: 4 (DW_TAG_base_type)
<6f> DW_AT_name : int
<73> DW_AT_encoding : 5 (signed)
<74> DW_AT_byte_size : 4

<1><7c>: Abbrev Number: 5 (DW_TAG_pointer_type)
<7d> DW_AT_type : <0x6e>
<2><5f>: Abbrev Number: 3 (DW_TAG_formal_parameter)
   DW_AT_location      : (DW_OP_fbreg: -16)
   DW_AT_name          : z
   DW_AT_decl_file     : 1
   DW_AT_decl_line     : 1
   DW_AT_type          : <0x7c>

<2><6d>: Abbrev Number: 0

<1><6e>: Abbrev Number: 4 (DW_TAG_base_type)
   DW_AT_name          : int
   DW_AT_encoding      : 5 (signed)
   DW_AT_byte_size     : 4

<1><7c>: Abbrev Number: 5 (DW_TAG_pointer_type)
   DW_AT_type          : <0x6e>

<44>   DW_AT_location      : (DW_OP_fbreg: -4)
<47>   DW_AT_name          : x
<4b>   DW_AT_decl_file     : 1
<4c>   DW_AT_decl_line     : 1
<4d>   DW_AT_type          : <0x6e>

<47>   DW_AT_name          : x
<4b>   DW_AT_decl_file     : 1
<4c>   DW_AT_decl_line     : 1
<4d>   DW_AT_type          : <0x6e>

<7a>   DW_AT_encoding      : 5 (signed)
<7b>   DW_AT_byte_size     : 2
<1><7c>: Abbrev Number: 5 (DW_TAG_pointer_type)
<7d>   DW_AT_type          : <0x6e>

<1><81>: Abbrev Number: 0
.debug_line: synchronising source and object

CU: /Users/zappa/tmp/hand.c:

<table>
<thead>
<tr>
<th>File name</th>
<th>Line number</th>
<th>Starting address</th>
</tr>
</thead>
<tbody>
<tr>
<td>hand.c</td>
<td>5</td>
<td>0x8048604</td>
</tr>
<tr>
<td>hand.c</td>
<td>6</td>
<td>0x804860a</td>
</tr>
<tr>
<td>hand.c</td>
<td>9</td>
<td>0x8048613</td>
</tr>
<tr>
<td>hand.c</td>
<td>10</td>
<td>0x804861c</td>
</tr>
<tr>
<td>hand.c</td>
<td>9</td>
<td>0x8048630</td>
</tr>
<tr>
<td>hand.c</td>
<td>11</td>
<td>0x804863c</td>
</tr>
<tr>
<td>hand.c</td>
<td>15</td>
<td>0x804863e</td>
</tr>
<tr>
<td>hand.c</td>
<td>16</td>
<td>0x8048647</td>
</tr>
<tr>
<td>hand.c</td>
<td>17</td>
<td>0x8048653</td>
</tr>
<tr>
<td>hand.c</td>
<td>18</td>
<td>0x8048658</td>
</tr>
</tbody>
</table>

Column numbers are also supported
If DWARF tables are correct by combining informations in different tables

1. we have metadata about computation at runtime

   Example: type information for all stack/register allocated variables

   Outcome: identify roots for a *precise garbage collector* for C runtime type-checking for C

2. we can relate source and machine code

   Outcome: *precise provenance informations* translation validation for existing C compilers
If DWARF tables are correct
by combining informations in different tables

1. we have

   Example: the
   Outcome:

2. we can

   Outcome:
Dynamically Diagnosing Type Errors in Unsafe Code

Stephen Kell

1. we have metadata about computation at runtime

Example: type information for all stack/register allocated variables

Outcome: identify roots for a precise garbage collector for C

(check out Kell’s libcrunch!)

2. we can relate source and machine code

Outcome: precise provenance information

translation validation for existing C compilers

```
if (obj->type == OBJ_COMMIT) {
    if (process_commit(walker, (struct commit *)iobj))
        return -1;
    return 0;
}
```

check this at run-time
Dynamically Diagnosing Type Errors in Unsafe Code

Stephen Kell

1. we have metadata about computation at runtime

   Example: type information for all stack/register allocated variables

   Outcome: identify roots for a precise garbage collector

   runtime type-checking for C (check out Kell’s libcrunch!)

2. we can relate source and machine code

   Outcome: precise provenance information, translation validation

   if (obj->type == OBJ_COMMIT) {
     if (process_commit(walker, (struct commit *)iobj))
       return -1;
     return 0;
   }

   binary compatible
   source compatible
   reasonable performance
   not C-specific

   check this at run-time
If DWARF tables are correct by combining informations in different tables

1. we have metadata about computation at runtime

   *Example*: type information for all stack/register allocated variables

   *Outcome*: identify roots for a *precise garbage collector* for C
               *runtime type-checking* for C

2. we can relate source and machine code

   *Outcome*: *precise provenance informations translation validation* for existing C compilers
If DWARF tables are correct
by combining informations in different tables

1. we have metadata about computation at runtime

   Example: type information for all stack/register allocated variables

   Outcome: identify roots for a \textit{precise garbage collector} for C
            \textit{runtime type-checking} for C

2. we can relate source and machine code

   Outcome: \textit{precise provenance informations}
            \textit{translation validation} for existing C compilers

Ongoing on Cheri-C
DWARF tables hide at the heart of our computing infrastructure

Poorly specified and badly designed
Pervaded by subtle bugs
Mostly ignored by the research community
Not understood by most programmers

Potential largely unexploited
DWARF tables hide at the heart of our computing infrastructure

- Poorly specified and badly designed
- Pervaded by subtle bugs
- Mostly ignored by the research community
- Not understood by most programmers

Exciting research to come!

PhD / PostDoc positions available!