Support for mini-debuginfo in LLDB

How to read the .gnu_debugdata section

Konrad Kleine

February 2, 2020

Red Hat - LLDB
Overall goal

Improve LLDB for Fedora and RHEL release binaries\(^1\)
- when no debug symbols installed
  - not all function symbols *directly* available (only `.dynsym`)
    - backtraces/coredumps mostly show addresses

Approach
- Make LLDB understand mini-debuginfo
  - that’s where more function symbols are

---

\(^1\)Mini-debuginfo used since Fedora 18 (2013, Release Notes 4.2.4.1.) and RHEL 7.x
Why was mini-debuginfo invented and how?

• Without installing debug infos
  • be able to generate a backtrace for crashes with ABRT
  • have full symbol table (.symtab)
  • have line information (.debug_line)
  • more than two sections make up an ELF file?!

• Eventually only one relevant section
  • stripped .symtab (simplified: just function symbols)
  • rest was too big
  • ELF format remained
  • no replacement for separate full debug info
  • not related to DWARF
    • just symbol tables

---

2Automatic Bug Reporting Tool
Symbol tables in an ELF file

**Regular ELF file**
- `.symtab`: Symbols needed by linker to construct program
- `.dynsym`: Symbols needed at runtime
- Duplicate symbols
- Usually stripped out into separate debug package

**ELF file with mini-debuginfo**
- `.symtab`: Symbols needed for backtrace
- `.dynsym`: Symbols needed at runtime
- `.gnu_debugdata` (aka mini-debuginfo) (LZMA compressed ELF-object)
- Stripped symbols (i.e. variables or parameters)
- Function symbols
- Other stuff

Other stuff
Approach

Not focus on backtraces
- but make LLDB see mini-debuginfo
  - set and hit breakpoint
  - dump symbols `((lldb) image dump symtab)`

Take existing Fedora binary (`/usr/bin/zip`)
- identify a symbol/function
  - not from `.dynsym`
  - from within `.gnu_debugdata`
- shootout: GDB vs. LLDB
## Identify symbol not directly accessible

```bash
# Show symbols
~$ eu-readelf -s zip.gdd

Symbol table [28] '.symtab' contains 202 entries:
82 local symbols  String table: [29] '.strtab'

<table>
<thead>
<tr>
<th>Num</th>
<th>Value</th>
<th>Size</th>
<th>Type</th>
<th>Bind</th>
<th>Vis</th>
<th>Ndx</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000000000000000</td>
<td>0</td>
<td>NOTYPE</td>
<td>LOCAL</td>
<td>DEFAULT</td>
<td>UNDEF</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0000000000408db0</td>
<td>494</td>
<td>FUNC</td>
<td>LOCAL</td>
<td>DEFAULT</td>
<td>15</td>
<td>freeup</td>
</tr>
<tr>
<td>2</td>
<td>000000000000408fa0</td>
<td>1015</td>
<td>FUNC</td>
<td>LOCAL</td>
<td>DEFAULT</td>
<td>15</td>
<td>DisplayRunningStats</td>
</tr>
<tr>
<td>3</td>
<td>000000000004093a0</td>
<td>128</td>
<td>FUNC</td>
<td>LOCAL</td>
<td>DEFAULT</td>
<td>15</td>
<td>help</td>
</tr>
</tbody>
</table>

[...]```

**help looks promising**.

```bash
~$ eu-readelf --symbols /usr/bin/zip | grep help
~$
```

---

3 Promising as in: we may be able to trigger it with `/usr/bin/zip --help`. 
Let’s be brave and do a demo!

Didn’t work?
READY TO SHIP?
What tests exist for mini-debuginfo?

- find symbol from `.gnu_debugdata`
- warning when mini-debuginfo w/o LZMA support
- error when decompressing corrupted xz
- full example with compiled and modified code analogue to gdb’s documentation
You might wonder...

What was the hardest part?

- 😊 setting a breakpoint worked
- 😞 hitting a breakpoint didn’t work
  - non-runnable/sparse ELF files in YAML form didn’t cut it
- 📚 dealing with tests
  - `yaml2obj` always produced `.symtab`
    - made my tests go nuts
- ⚖️ polishing for upstream
- ⏳ got more time?

---

“`yaml2obj` takes a YAML description of an object file and converts it to a binary file.”

(https://llvm.org/docs/yaml2obj.html)
Thank you!

- [GitHub](https://github.com/kwk/talks/)
- [LinkedIn](https://www.linkedin.com/in/konradkleine)
- [Feedback](https://submission.fosdem.org/feedback/10393)
- [Sourceware](https://sourceware.org/gdb/current/onlinedocs/gdb/MiniDebugInfo.htm)
Appendix
// REQUIRES: system-linux, lzma, xz
// RUN: gcc -g -o %t %s
// RUN: %t 1 2 3 4 | FileCheck %s

#include <stdio.h>

int main(int argc, char* argv[]) {
  // CHECK: Number of {{.*}}: 5
  printf("Number of arguments: %d\n", argc);
  return 0;
}

• features added: lzma, xz
  • just some CMake canonisation and Python config

  if config.lldb_enable_lzma:
    config.available_features.add('lzma')
  if find_executable('xz') != None:
    config.available_features.add('xz')
Check to find symbol `multiplyByFour` in mini-debuginfo

```yaml
# REQUIRE: lzma
# RUN: yaml2obj %s > %t.obj
# RUN: llvm-objcopy --remove-section=.symtab %t.obj
# RUN: %lldb -b -o 'image dump symtab' %t.obj | FileCheck %s
# CHECK: [ 0] 1 X Code 0x000000000000004b0 0x000000000000000f 0x00000012 multiplyByFour
```

--- !ELF
FileHeader:
  Class: ELFCLASS64
  Data: ELFDATA2LSB
  Type: ET_EXEC
  Machine: EM_X86_64
  Entry: 0x00000000004004C0
Sections:
  - Name: .gnu_debugdata
    Type: SHT_PROGBITS
    AddressAlign: 0x0000000000000001
    Content: FD377A585A000004E6 # ...
  ...
Extract + decompress `.gnu_debugdata` from `/usr/bin/zip`

```bash
# Dump section
~$ objcopy --dump-section .gnu_debugdata=zip.gdd.xz zip

# Determine file type of section
~$ file zip.gdd.xz
zip.gdd.xz: XZ compressed data

# Decompress section
~$ xz --decompress --keep zip.gdd.xz

# Determine file type of decompressed section
~$ file zip.gdd
zip.gdd: ELF 64-bit LSB executable, x86-64, version 1 [...]
```

thank you
Set and hit breakpoint on `help` with GDB 8.3

```
~$ gdb --nx --args /usr/bin/zip --help
Reading symbols from /usr/bin/zip...
Reading symbols from .gnu_debugdata for /usr/bin/zip...
(No debugging symbols found in .gnu_debugdata for /usr/bin/zip)
Missing separate debuginfos, use: dnf debuginfo-install zip-3.0-25.fc31.x86_64
(gdb) b help
Breakpoint 1 at 0x4093a0
(gdb) r
Starting program: /usr/bin/zip --help
Breakpoint 1, 0x00000000004093a0 in help ()
(gdb)
```

Success and two things to note:

1. Symbols read from `.gnu_debugdata`
2. No debug symbols installed for zip

---

GDB 8.3 is what ships with Fedora 31
Set and hit breakpoint on **help** with LLDB 9.0.0

```
~$ lldb -x /usr/bin/zip -- --help

(lldb) target create "/usr/bin/zip"
Current executable set to '/usr/bin/zip' (x86_64).
(lldb) settings set -- target.run-args "--help"

(lldb) b help
Breakpoint 1: no locations (pending).
WARNING: Unable to resolve breakpoint to any actual locations.

(lldb)
```

---

6LLDB 9.0.0 is what ships with Fedora 31
Show that LLDB can now find help symbol

```bash
$ lldb -x /usr/bin/zip -- --help
...

(lldb) b help
Breakpoint 1: where = zip 'help, address = 0x00000000004093a0

(lldb) r
Process 277525 launched: '/usr/bin/zip' (x86_64)
Process 277525 stopped
* thread #1, name = 'zip', stop reason = breakpoint 1.1
  frame #0: 0x00000000004093a0 zip 'help
zip 'help:
  -> 0x4093a0 <+0>: pushq %r12
     0x4093a2 <+2>: movq 0x2af6f(%rip), %rsi ; + 4056
     0x4093a9 <+9>: movl $0x1, %edi
     0x4093ae <+14>: xorl %eax, %eax

(lldb)
```

❤️ shipping with LLVM 10 (Back to demo)