



Red Hat  
**Platform**  
**Tools Team**

## Support for mini-debuginfo in LLDB

How to read the `.gnu_debugdata` section

---

Konrad Kleine



February 2, 2020

Red Hat



- LLDB, C/C++, ELF, DWARF since 2019
- joined and worked on OpenShift in 2016

## Reach out

-  <https://github.com/kwk/talks/>
-  <https://www.linkedin.com/in/konradkleine>



## Improve LLDB for Fedora and RHEL *release* binaries<sup>1</sup>

- 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

---

<sup>1</sup>Mini-debuginfo used since Fedora 18 (2013, Release Notes 4.2.4.1.) and RHEL 7.x

## 💡 Why was mini-debuginfo invented and how?

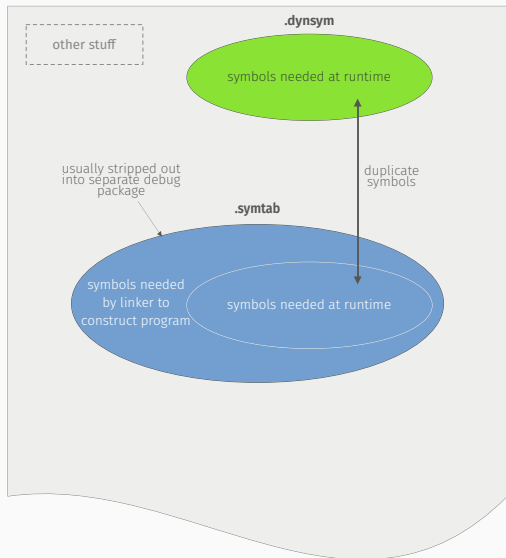
- Without installing debug infos
  - be able to generate a backtrace for crashes with ABRT<sup>2</sup>
  - ~~have full symbol table (`.symtab`)~~
  - ~~have line information (`.debug_line`)~~
  - *more than two sections make up an ELF file?!*
- Eventually only one relevant section
  - stripped `.symtab` (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*

---

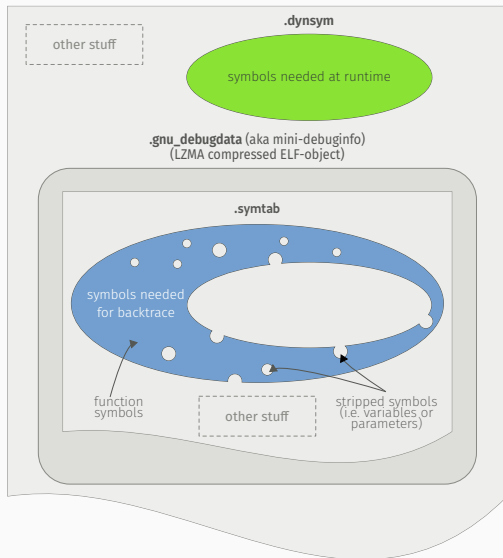
<sup>2</sup>Automatic Bug Reporting Tool

# Symbol tables in an ELF file

regular ELF file



ELF file with mini-debuginfo



## 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

```
1 # Show symbols
2 ~$ eu-readelf -s zip.gdd
3
4 Symbol table [28] '.symtab' contains 202 entries:
5 82 local symbols String table: [29] '.strtab'
6 Num:          Value      Size Type      Bind    Vis      Ndx Name
7 0: 0000000000000000      0 NOTYPE    LOCAL  DEFAULT  UNDEF
8 1: 0000000000408db0    494 FUNC      LOCAL  DEFAULT   15 freeup
9 2: 0000000000408fa0   1015 FUNC      LOCAL  DEFAULT   15 DisplayRunningStats
10 3: 00000000004093a0    128 FUNC      LOCAL  DEFAULT   15 help
11 [...]
```

help looks promising<sup>3</sup>.

```
12 ~$ eu-readelf --symbols /usr/bin/zip | grep help
13 ~$
```

---

<sup>3</sup>Promising as in: we may be able to trigger it with `/usr/bin/zip --help`.

...and do a demo!

Didn't work?



### Symtab (reminder)

- normally, `.dynsym` is a subset
- **but** for mini-debuginfo `.dynsym` symbols are stripped<sup>4</sup> from symtab

### Implications for LLDB (and other tools)

- parse `.dynsym` when
  - no `.symtab` found **or**
  - mini-debuginfo present and smuggled in

---

<sup>4</sup><https://sourceware.org/gdb/current/onlinedocs/gdb/MiniDebugInfo.html>

...and do *yet another* 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-runable**/sparse ELF files in YAML form didn't cut it
- 📖 dealing with tests
  - `yaml2obj`<sup>5</sup> always produced `.symtab`
    - made my tests go nuts

### Community aspects

- ⚖️ polishing for upstream
- 🐙 repo migration to github, review in phabricator

---

<sup>5</sup>*"yaml2obj takes a YAML description of an object file and converts it to a binary file."*

(<https://llvm.org/docs/yaml2obj.html>)



# Thank you!

Please, share your feedback ★★★★★

<https://submission.fosdem.org/feedback/10393>

## Appendix

### lldb/test/Shell/Breakpoint/example.c:

```
1 // REQUIRES: system-linux, lzma, xz
2 // RUN: gcc -g -o %t %s
3 // RUN: %t 1 2 3 4 | FileCheck %s
4
5 #include <stdio.h>
6 int main(int argc, char* argv[]) {
7     // CHECK: Number of {{.*}}: 5
8     printf("Number of arguments: %d\n", argc);
9     return 0;
10 }
```

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

```
1 if config.lldb_enable_lzma:
2     config.available_features.add('lzma')
3 if find_executable('xz') != None:
4     config.available_features.add('xz')
```



## Real example of sparse ELF test file

🔍 Check to find symbol `multiplyByFour` in mini-debuginfo

```
1 # REQUIRES: lzma
2 # RUN: yaml2obj %s > %t.obj
3 # RUN: llvm-objcopy --remove-section=.symtab %t.obj
4 # RUN: %lldb -b -o 'image dump symtab' %t.obj | FileCheck %s
5 # CHECK: [ 0] 1 X Code 0x00000000004005b0 0x000000000000000f 0x00000012 multiplyByFour
6
7 --- !ELF
8 FileHeader:
9   Class:      ELFCLASS64
10  Data:       ELFDATA2LSB
11  Type:       ET_EXEC
12  Machine:    EM_X86_64
13  Entry:      0x00000000004004C0
14 Sections:
15   - Name:     .gnu_debugdata
16     Type:     SHT_PROGBITS
17     AddressAlign: 0x0000000000000001
18     Content:   FD377A585A000004E6 # ...
19 ...
```

- notice line 3 manually removes `.symtab`
- meanwhile `yaml2obj` was fixed

## Extract + decompress .gnu\_debugdata from /usr/bin/zip

```
1  # Dump section
2  ~$ objcopy --dump-section .gnu_debugdata=zip.gdd.xz zip
3
4  # Determine file type of section
5  ~$ file zip.gdd.xz
6  zip.gdd.xz: XZ compressed data
7
8  # Decompress section
9  ~$ xz --decompress --keep zip.gdd.xz
10
11 # Determine file type of decompressed section
12 ~$ file zip.gdd
13 zip.gdd: ELF 64-bit LSB executable, x86-64, version 1 [...]
```



## Set and hit breakpoint on help with GDB 8.3<sup>6</sup>

```
1 ~$ gdb --nx --args /usr/bin/zip --help
2
3 Reading symbols from /usr/bin/zip...
4 Reading symbols from .gnu_debugdata for /usr/bin/zip...
5 (No debugging symbols found in .gnu_debugdata for /usr/bin/zip)
6 Missing separate debuginfos, use: dnf debuginfo-install zip-3.0-25.fc31.x86_64
7
8 (gdb) b help
9 Breakpoint 1 at 0x4093a0
10
11 (gdb) r
12 Starting program: /usr/bin/zip --help
13
14 Breakpoint 1, 0x00000000004093a0 in help ()
15 (gdb)
```

Success and two things to note:

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

---

<sup>6</sup>GDB 8.3 is what ships with Fedora 31



```
1 ~$ lldb -x /usr/bin/zip -- --help
2
3 (lldb) target create "/usr/bin/zip"
4 Current executable set to '/usr/bin/zip' (x86_64).
5 (lldb) settings set -- target.run-args "--help"
6
7 (lldb) b help
8 Breakpoint 1: no locations (pending).
9 WARNING: Unable to resolve breakpoint to any actual locations.
10
11 (lldb)
```

Back to demo 1.

---

<sup>7</sup>LLDB 9.0.0 is what ships with Fedora 31

## Show that LLDB can now find help symbol

```
1 $ lldb -x /usr/bin/zip -- --help
2 ...
3
4 (lldb) b help
5 Breakpoint 1: where = zip'help, address = 0x00000000004093a0
6
7 (lldb) r
8 Process 277525 launched: '/usr/bin/zip' (x86_64)
9 Process 277525 stopped
10 * thread #1, name = 'zip', stop reason = breakpoint 1.1
11     frame #0: 0x00000000004093a0 zip'help
12 zip'help:
13 -> 0x4093a0 <+0>: pushq   %r12
14     0x4093a2 <+2>: movq    0x2af6f(%rip), %rsi        ; + 4056
15     0x4093a9 <+9>: movl    $0x1, %edi
16     0x4093ae <+14>: xorl    %eax, %eax
17 (lldb)
```