ORACLE

Beyond DWARF

drgn support for CTF and BTF

Stephen Brennan

November 13, 2023

Outline

1. Motivation
2. Goals
3. Implementation
4. Discussion

Introduction

- I work at Oracle on the Linux Sustaining team.
- We investigate and fix kernel bugs for customers.
- Customers configure kdump and provide vmcores, which we analyze in a secure server.
- Ideally, we analyze w/ crash, drgn, etc., and find & fix the bug.
- Frequently, it’s not that simple… need to work with the customer for more info.

Customer Debugging

- Debugging on a customer system is hard.
- For internal debugging, we have a shared debuginfo repository for easy access of DWARF debuginfo.
- For customers, you need them to install debuginfo packages.
- This is… frequently a problem.

New Users

- Debuginfo can be a brick wall for people interested in using Drgn.
- Some distributions flat-out don’t have kernel debuginfo (BTW, it’s Arch).
- Drgn documentation does a good job explaining it, yet it is full of
  - distro-specific steps
  - adding/changing package repositories
  - rarely-used commands (`debuginfo-install` … ?)
  - debuginfod??

DWARF is BIG

- For the `vmlinux` file alone, debuginfo (at Oracle, at least) ranges from 400MiB to 1 GiB. Then, add modules.
- For the price, you get some great features:
  - source code mapping
  - CFI
- But the majority of drgn’s features rely on only three things:
  - symbol table
  - type information
  - stack unwinder

Observations

- The kernel contains a symbol table!
- The kernel has pretty reliable stack unwinder: frame pointers, or ORC for x86_64
- The kernel frequently already comes with type info (beyond DWARF):
  - Most distribution kernels come with BPF, and thus BTF, enabled.
  - Oracle UEK kernels come with CTF

Goal

- Drgn could support a “mix & match” approach to debuginfo!
- Symbol tables:
  - vmlinux: built-in kallsyms (or `/proc/kallsyms`), ELF symtab
  - module: exports, and also module kallsyms
- Debuginfo:
  - DWARF, CTF, BTF
- Caveat: none of this is to replace DWARF
  - A lightweight option that can do 80% for 20% of the cost

Examples

- Example 1: DWARF-less
  - Use CTF, /proc/kallsyms, and FP or ORC
- Example 2: vmlinux available, but no modules
  - Vmlinux: DWARF, ELF symbol table.
  - Modules: module exports & kallsyms, [BC]TF
- Example 3: standalone vmcore
  - Use /proc/kallsyms, BTF (built-in), FP or ORC

What is CTF?

- Compact C Type Format
- Built-in to GCC, GDB, binutils, etc:
  - `gcc -gctf ...`
  - use GDB like normal
- Normally, it is put in the `.ctf` ELF section
- It should not be stripped
- Kernel CTF is special
  - CTF info is linked and placed into a standalone file on the filesystem. Included within the standard kernel package.

Step 1: Pluggable Symbol Finder

- This is in review.
- Allows some really fun stuff - with this implementation you can add a module symbol finder based on exports / kallsyms, and this can allow you to get stack traces with module & function names.
- You can write a symbol finder in Python!

Step 1: Pluggable Symbol Finder

```
>>> def symbol_finder(name, addr, one):
...     if name == "the_secret" or addr == 42:
...         return [drgn.Symbol(
...             "the_secret", 42, 8,
...             drgn.SymbolBinding.GLOBAL,
...             drgn.SymbolKind.OBJECT,
...         )]
...     return []
>>>
>>> prog.add_symbol_finder(symbol_finder)
>>> prog.symbol("the_secret")
Symbol(name='the_secret', address=0x2a, size=0x8, binding=<SymbolBinding.GL
```

Step 2: Implementing Kallsyms

- This is in a draft state
- A C implementation based on the new Symbol Finder API
- Supports `/proc/kallsyms` for live use cases (if the user has permission)
- Supports reading data structures out of core dump too

Issue #1: No symbol sizes

- ELF includes symbol sizes:

```
typedef struct
{
  Elf64_Word     st_name;   /* Symbol name (string tbl index) */
  unsigned char  st_info;   /* Symbol type and binding */
  unsigned char  st_other;  /* Symbol visibility */
  Elf64_Section  st_shndx;  /* Section index */
  Elf64_Addr     st_value;  /* Symbol value */
  Elf64_Xword    st_size;   /* Symbol size */
} Elf64_Sym;
```

Issue #1: No symbol sizes

- `/proc/kallsyms` does not:

```
ffffffff8e45db00 D slab_mutex
ffffffff8e45db20 D slab_caches
```

Issue #1: No symbol sizes

```
000000000002d0c0 A apf_reason
000000000002e000 A __per_cpu_end
ffffffff8c600000 T startup_64
ffffffff8c600000 T _stext
ffffffff8c600000 T _text
```

Thanks, `x86`!

Issue #2: Kallsyms encoding changes

```
commit 73bbb94466fd3f8b313eeb0b0467314a262dddb3
Author: Miguel Ojeda <ojeda@kernel.org>
Date:   Mon Apr 5 04:58:39 2021 +0200

    kallsyms: support "big" kernel symbols

    Rust symbols can become quite long due to namespacing introduced
    by modules, types, traits, generics, etc.

    Increasing to 255 is not enough in some cases, therefore
    introduce longer lengths to the symbol table.

    In order to avoid increasing all lengths to 2 bytes (since most
    of them are small, including many Rust ones), use ULEB128 to
    keep smaller symbols in 1 byte, with the rest in 2 bytes.
```

Step 3: Add CTF implementation

- Drgn has two important abstractions here, both are already pluggable:
  - Type finder: looks up the definition of a type.
  - Object finder: looks up a variable name and returns an object (with type)
- Kernel CTF has no symbol table, but it maps symbol names to their types.
- CTF implementation links to libctf (part of binutils)

Step 3: Add BTF implementation?

I have an old BTF implementation that is quite stale.
I hope to revive it once CTF is merged, but it depends on whether there would be users.

Discussion

- I’d love to hear other thoughts on smaller “runtime debuginfo”
- Would you use CTF in drgn?
- Would you use BTF in drgn?
- How do you manage debuginfo?
- How do your users manage debuginfo?