Taming of the kernel dump
Petr Tesarik
Talk about libkdumpfile.

Motivation: identifying kernel versions for unknown dumps.

Goals:
- Fast, broad support
- Support other OS than linux
- Working within the kexec kernel.
- Simple API - pull requests welcome :)

Now, supports 5 architectures and a bunch of core dump formats.

Support zoo
- supports 5 architectures, but compared to crash/makedumpfile, it supports far fewer
- Supports quite a range of core dump formats

Adding support for architectures is well-documented based on the recent architecture addition work.

Wild ideas:
- gdbserver - actually working!
- Write support
  - API design is an interesting problem
  - It's on the agenda, not actively worked on.

Questions:
- Omar: can we do a better job sharing page table support between drgn and libkdumpfile?
  - Yes, it would be nice to do, but adds a hard dependency on libkdumpfile
  - A pull request was offered :)
  - Open question: userspace address translation (how to support multiple userspace pids)
- Stephen: any thoughts on making it easier to do the "linux support" per-architecture (e.g. finding page table base)
  - It's an open question, most of the time is spent focusing on supporting what's in the wild already.

drgn - writing to memory in production
Omar Sandoval

Starting with an introduction to drgn - stack traces, helpers, variable access, etc. However, everything in the demo is read-only. RW features would be nice:
- write a variable?
- set a breakpoint?
- Would be great for development workflows, e.g. a gdbstub via QEMU

API Proposal

- Low-level: `prog.write(address, bytes)` - basic write function
- Object-based: `Object.write_(value)` - drgn can take care of converting a Python value into the correct type format, and determine address+size

- Up for debate, these are just proposals
- Breakpoint:
  - `prog.set_breakpoint(address)`
  - `prog.set_breakpoint("function_name"`
  - `prog.set_breakpoint("file.c:lineno"`
  - Add an API such as `prog.get_thread_event()` to wait for breakpoint events.
  - Event (thread) would have a `resume()` method.
- Breakpoint: API suggestion: frequently I only want breakpoints to be hit if they are called from a specific function
  - Could be done via Drgn APIs without needing to add support
  - but if it was a common and slow operation, could add a fast path (e.g. BPF to filter)
- Suggestion: watchpoint support?
- What about error injection?
  - E.g. set breakpoint on the `ret`, overwrite register value.
  - That would need to have a new item in the API for overwriting registers.

Why In Production?
- There have been several issues where you could easily mitigate a critical error, if only you could overwrite one little value! (Concrete example at Meta w/ btrfs)
- Need to be very careful about these sorts of things:
  - Race conditions are a thing
  - Not a replacement for a live patch

How to???
- Memory writing is not supported by `/proc/kcore`, but it is via QEMU in development. What about production use cases?
  - Return `/dev/kmem` :P - probably not an option, we've all celebrated at its funeral.
  - Custom kernel module? (That's basically `/dev/kmem`). This is not much better...
- KGDB - already provides breakpoints, and memory writing. Provides a gdb stub.
  - Not really designed for the live system - stops every CPU.
- Question: is it really bad if KGDB stops all CPUS?
  - Then, drgn can't run... It's great if you're running from a second machine, not great for production.
  - Follow-up: what if you set a breakpoint and then Drgn calls a function that hits it?
  - Omar hopes to do this from the same local machine.
  - kprobes? Would still need a way to call out to Drgn.
- Question: what about lockdown?
  - Clearly this mechanism should not try to work around lockdown. It should be disabled by that.
- Access control?
  - Seems like a user/capabilities approach would not be sufficient, given the community's current stance on `/dev/kmem`
  - What if we have a keyring that would sign drgn and the script?
- Q: What about translating breakpoint code into BPF?

Beyond DWARF: Debugging the Kernel with Drgn, BTF/CTF, and kallsyms
By Stephen Brennan

- Stephen works on Linux Sustaining team
at Oracle, investigating customer bugs
(investigate vmcores rom customers, etc).

- Debugging with a customer is hard!
A frequent problem is having the customer
to install debuginfo pkgs. Internally,
it's easier (shared debug symbols server).

- Debuginfo could be a showstopper for some,
like Arch Linux doesn't even have by
default debug symbols for kernel.
In other distros, it's a custom process,
distro-specific. Would like that it was
easier to do it...

- DWARF is big - for vmlinux alone,
400M-1G. Good features, but a big "price".
Majority of Drgn features though, rely
on 3 things: symbol table, type info and
maybe (for some cases), stack unwinder.

- Well, kernel already has a symbol table
and reliable stack unwiding (frame ptrs or
ORC!). It also could come with some type
info beyond DWARF (BTF!). Oracle kernels
come with CTF (to be discussed ahead).

- So, goals: mix&match approach for drgn =>
symbol tables (built-in kallsyms, ELF symtab),
modules exports.
Debuginfo: CTF / BTF
??? 3rd point

- Examples
1) DWARF-less: kallsyms, FP/ORC
2) vmlinux available but no modules: modules
exports, BTF
3) ??

- What is CTF? Compact C Type Format
Built-in to GCC/GDB/binutils, put in
.ctf ELF section. Shouldn't be stripped.
Kernel CTF: special case, linked and
placed as standalone file, packaged with
kernel usually.

- Next steps
(1) Pluggable Symbol Finder: one thing you
can't do it right now is plug an "additional"
symbol table (proposal under review). An
example was provided in python.

(2) Implementing kallsyms: to be able to
plug kallsyms on drgn (under review/draft).
It's a C implementation, could either
read /proc/kallsyms directly (given user
has permissions) or load from a core
dump, by reading the data structure.

- Issues:
(#1) No symbol sizes: some info on kallsyms
(like saying a symbol is Dynamic and Global),
but no size info. Percpu makes it harder,
also overlapping symbols (x86 entry).

(#2) Kallsyms enconding changes: "recent"
change that increased the symbol name sizes
(Rust might have big ones, due to namespacing).

- Back to steps:
(3) Add CTF implementation
Drgn has 2 abstractions that are pluggable:
type finder and obj finder. Kernel CTF has no
symbol table, but maps symbol names to their
types. Implementation not so simple though.

(3b) Add BTF implementation? After CTF is done,
there's an old/stale BTF implementation by
Stephen, maybe gonna revive it (depending
on if there's going to be users).

Discussion questions:
Would you use CTF/BTF in drgn?
How do you manage your debuginfo?

Questions:
[audience 1]: About the percpu, no addr is that small,
this is an offset from the base (in the example)
Stephen: However is implement in arch, drgn must
know how it works.

[2] Smaller runtime debuginfo would be
great, a pain to download it. What you'd give
up by going this path?
Stephen: one big thing that is lost is
source code mapping, also lose some detail
regarding function unwiding (nested inline
function calls, for example), also lose variable
names.

[Philip]: One thing that would also be interesting,
is for OOT modules (that vendors don't provide
debug info).

[4]: Has anyone tried to make dwarf symbols
smaller?
Stephen: very good question, Omar might give some info.

Omar: If you control debuginfo, that's feasible. But since
it's under distro control, they include it all.

[the person that did the question]: the question was more
like if we could stop enabling it all and restricting
a bit what to include.

Omar:??

[5]: Part of the issue is that, when you
build the debuginfo,

Omar: So, you mentioned CTF is not included in the kernel image.

Stephen: yeah

Omar: that is one thing BTF has compared to CTF, right?

Stephen: yeah

Omar: you mentioned debuginfod, drgn
also supports that - we could support
for the kernel as well, but there's the
download time (convenience)...server
side works took 3h

Stephen: RPMs are not the greatest format for seeking to random files within.

When kdump is way too much
Guilherme Piccoli

Motivation - steam deck - based on Arch linux. Want to capture logs on panic!
- kdump is a good option, but but heavyweight
- pstore is an option?

What to collect on panic?
- vmcore
  - It's really big

  - Contains memory contents, might be too much information
- dmesg
  - This is good, but would like to have more information.
  - Could have userspace processes send more data as well? Include userspace logs.

Infrastructure (kdump)
- Most are familiar, reserve memory and load a backup kernel.
- Reserved memory is difficult to determine! `crashkernel=xx`. Normally >200MiB now. Difficult to estimate. Users would also like to be able to use that memory.
- Size of the vmcore can be quite large, which can be problems for the user.
- There are privacy issues with kdump vmcores. Risks after rebooting with the privacy.
- Plus, there are risks during the kexec operation, (PCI devices, etc).

Pstore - lightweight ways
- Saves the kernel log to persistent storage backend (RAM, UEFI, ACPI ERST, block device)
- Common for embedded devices, steam deck, chromebook
- Benefits
  - Can use it for ftrace, dmesg, pmsg (userspace messages!)
  - No userspace memory reservation (`crashkernel`). Uses UEFI backend.
- Drawbacks
  - Can't store a full vmcore.
  - Runs after panic notifiers (for now)
  - Difficult configuration (for now)

New tool: kdumpst
- For arch linux, available in AUR already. Allows to configure vmcore or pstore.
- Pstore is the default, only supports ramoops, but planning to support UEFI.
- Used by default for Steam deck, but would like MOAR LOGS.

`panic_print` - prints additional data at panic time
- Can print each task's status, other info. (I missed some of the items).
- Runs after the panic notifiers :(

What are panic notifiers?
- Callbacks to be executed on `panic()`
- Any driver can register any one for any reason.
  - Some are necessary!
  - Some are risky
  - The kernel has `crash_kexec_post_notifiers` which gives the option to run kexec after panic notifiers. But it's not a very flexible solution.

Discussion:
- Is pstore risky?
- ramoops limitations
  - There's a risk of FW corrupting memory on boot
- Panic notifier risk?
- Not enough data still? even with `panic_print`

Q&A:

- Stephen Brennan: how to keep `panic_print` safe? trylocks?
  - yes, try locks
- Grant Grundler (Google) - we do use pstore, there are still privacy concerns and we're careful. There are two types of panics - precise (fault, e.g.), and imprecise (hung task panic). For imprecise, we would like more information: how to get it?
- Omar: maybe you could have BPF programs for collecting more specific info? When a customer crash happens, if you need more info, write a BPF program to collect it and send it out to your devices, so next crash will include it. Long debug cycles, but worth it?

Minidump to debug end user crashes
Mukesh Ojha & Elliot Berman

Problem:
- for development, phones dump the full memory image, ~ 12GiB or higher, it's getting out of hand.
- minidump is an option to nominate what memory regions (e.g. ARM specific) you want to store.

Alternatives:
- kdump
  - Uses a lot of memory reservation for the crashkernel
  - Need two kernels
  - Time & space overhead.
- pstore
  - Less overhead, but ramoops still needs a reservation
- minidump

Minidump Overview
- Firmware enabled
- User writes a table of contents defining what entries you want, each entry has an array of regions to dump.
- Firmware uses this to create an ELF dump file with each entry/regions included.
- Firmware can have a limited number of regions in the past, however it sounds like now the limitations are less.
- The hope is to allow in-kernel users to register their structures or critical information to be included as a region/entry.
- Patches are on v5 upstream.

Remote processor support
- I missed much of this but refer to slides.

Info we'd like to include
- The initial/boot dmesg logs
- MMIO tracing, IRQ statistics
- Run queue information

Minidump is not just for kernel crashes, you can trigger it for non-kernel issues.

Feedback! Can it be used outside Qualcomm? Useful for other SoC vendors, or other use cases?

- Stephen: the idea of using opt-in support for kernel data, is very interesting compared to makedumpfile's

filtering approach. Does it support userspace registering information?
  - No, the userspace support was mentioned in a different context for remote VM.
- Grant: two questions
  - Why is it lighter weight than pstore?
    - Just that ramoops needs copying, whereas this doesn't.
  - What are examples of non-kernel crashes?
    - Sometimes chipset errors, security regions, etc.
- Omar: we'd like to avoid more dumps that look like ELF but are missing critical data, could you include VMCOREINFO note?
  - Yeah, we'd be interested in looking into it.
- (Missed the question asker's name): Do you interpret the dmesg structures, or do you just copy memory data?
  - Copy memory data. Users of the dump will need to know the kernel structures, and so know the kernel image as well.
  - Omar: VMCOREINFO helps with knowing the kernel version.
- Guiherme Piccoli: your firmware writes to SD, do you need MMC drivers in order to make this work?
  - I missed this answer
- Hari Bathini: Is the primary motive with minidump to avoid having to reserve memory that is needed for a full kdump?
  - Yes, among other things.
- Omar: do you register the physical addresses of the task structs to be debugged?
  - This is something we want to be doing, but haven't implemented yet.

Livedump
Lukas Hruska

What is livedump?
- Create a consistent image of memory, but without needing to stop the machine (panic, kdump, reboot, etc).
- Already introduced by Yoshida Maasanori in 2012, this was used as a starting point. Thank you for the work!

Features:
- You don't need to restart. Uses copy-on-write to handle writes to frozen pages.
- Currently x86 only, but will extend

Implementation: split into parts.

wrprotect
- Makes page frames RO.
- Only supporting 4K pages right now
- Need to store the previous page frame permissions, to ensure that we restore the correct page permissions later.
- Can split large pages into smaller ones in order to do fine-grained access control.
- Registers a custom page fault handler. Gets a lot of page faults initially!
  - For protection errors, can check whether if the error is because of our updated permissions, or if it would have happened for the original pages as well.

Dumping

- Thread goes through each page and dumps it. If the page has already been dumped, skip.

Queue Size Problem:
- I/O might cause page fault which requires I/O, risk of recursion.
- Setting a larger queue size can help.

PF Handling Variants
- Small inconsistencies in data may be ok.
- Can do:
  - 100% consistent variant, but would cause disruption and performance loss
  - More inconsistent option with minimal impact.

Todo:
- vmap area support: needs to handle synchronization
  - Other issues too.
  - Omar: most debuggers don't actually need/want the vmap region dumped, as long as you have the corresponding phys/direct map data. You could skip this altogether and let the debugger handle the page table translation for vmalloc.
- Interrupt instead of stop-machine state.
- Performance analysis for splitting hugepages -- there's likely regressions here.
- Storage support

Q&A:
- Stephen: Dump filtering like makedumpfile? Do you do this yet? Could save a lot of work.
- Stephen: For I/O recursion issues, can you set a flag during I/O and have "emergency reserve" space to use in recursive page faults?