Using Library Compartmentalisation

Nick Connolly

Introduction

- CAP Computer *(Student)*
- DataCore Software *(Co-Founder and Chief Scientist)*
  - Pioneered Storage Virtualisation
  - Ported SPDK & DPDK to Morello
- Rtegrity *(Co-Founder)*
  - Library compartmentalisation for storage stack
- Arm *(Principal Software Engineer)*
  - Speaking in a personal capacity
  - Not part of Morello development team

BACKGROUND

Data is a critical resource that needs to be secured. This usually means encrypting data at rest and sometimes in motion.

The challenge is ensuring that data is protected whilst it is in use.

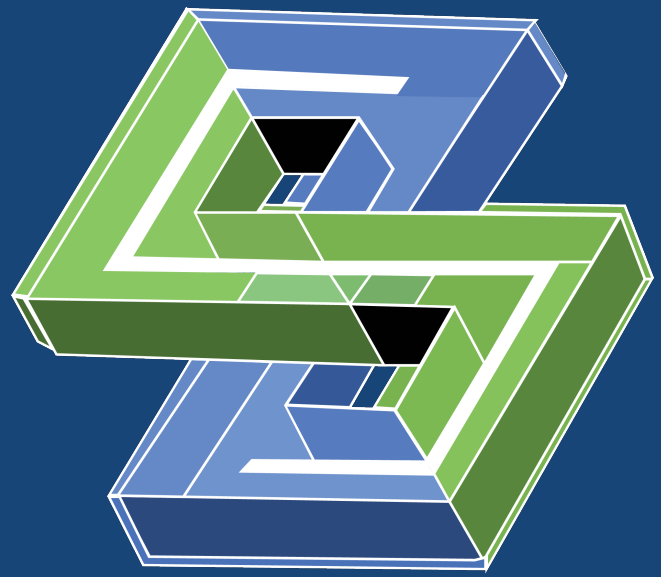
Primary Goal

To investigate the viability and impact of applying library compartmentalisation to an SPDK based storage stack

- SPDK co-exists inside the application
- Means that vulnerabilities in:
  - SPDK can compromise the application
  - The application can compromise the storage stack
- Isolate the storage stack from the application
  - Maybe isolate SPDK from its own dependencies?

Storage Performance Development Kit (SPDK)

- Consists of libraries and tools for writing
  - User-mode storage applications
- It is cutting edge
  - Leverages the latest NVMe features
  - Polling for maximum performance
  - Production Ready (mainstream products)
- Highly performant and scalable
  - Uses a lockless, thread-per-core design
  - Data structures are cache aligned and packed
  - Optimised for specific CPU features



Getting Started

- Choose the compartmentalisation boundary
  - Transitions have an execution cost
  - Balance the overhead against the security gain
- SPDK consists of about 250 libraries
  - Too many boundaries to tackle at once
- Library compartmentalisation is experimental
  - Based on implementation in CheriBSD 22.12
  - It's functional, but it’s still being developed
  - Documentation is early stage (well written papers)
  - Minimal example code

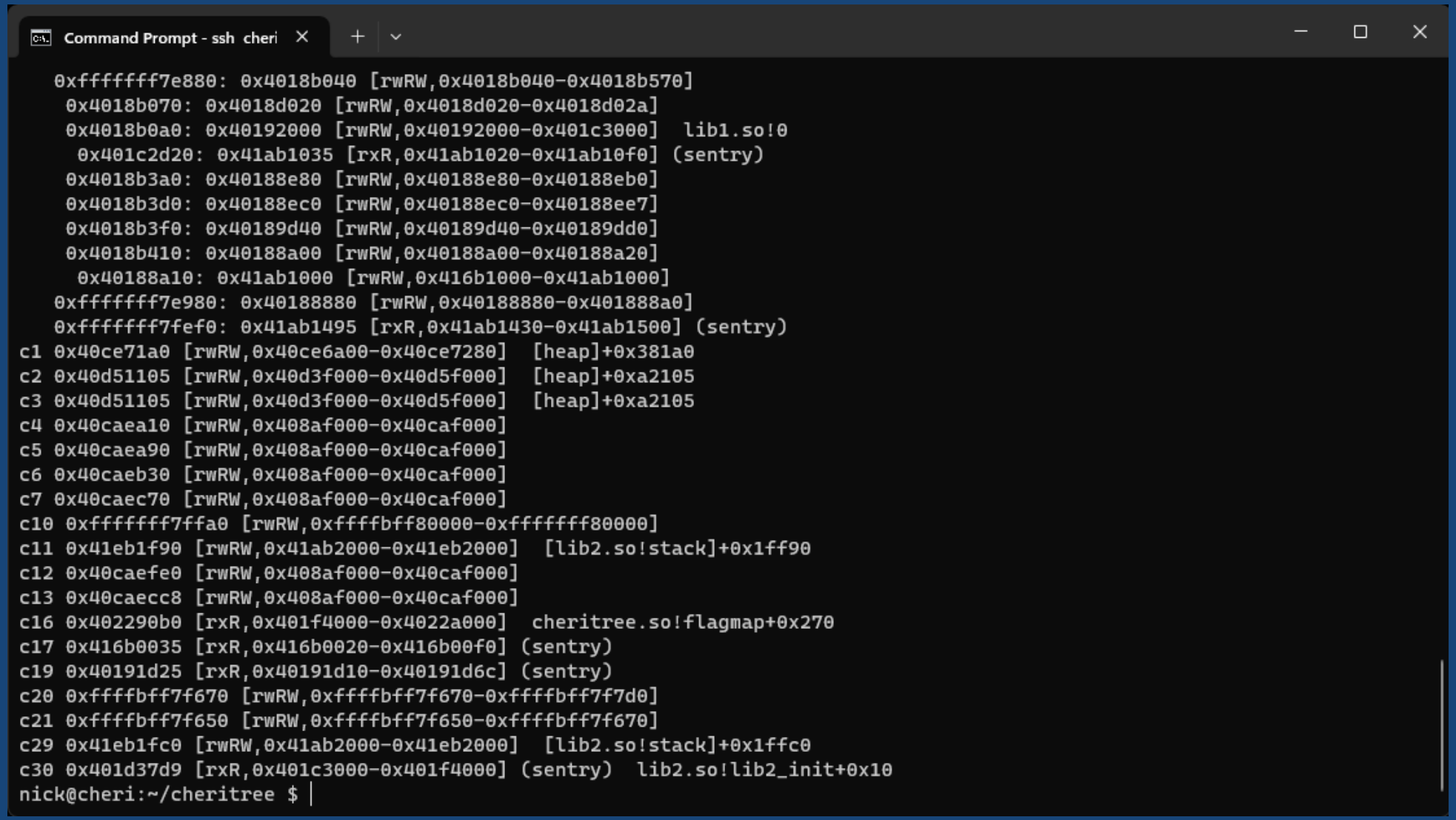
Test Program

- Started by creating a simple test program
  - Executable and three shared libraries
  - Two linked and one loaded with dlopen()
  - Probe function calls and variables across boundaries
- Allows easy experimentation
  - One executable, run with a different dynamic linker
  - Has proved to be very helpful
  - CHERI was great at finding coding issues during development!
- … but, how do you know the isolation is actually working?

CheriTree

- Created CheriTree library
  - Investigate capabilities accessible from current context
  - Saves register values and stack state
  - Prints a capability derivation tree
- Linked with the application
  - Programmatic interface (callable from gdb)
  - Designed to work with Linux or CheriBSD
- Open Source (BSD-3)
  - https://github.com/rtegrity/cheritree

CheriTree (cont ...)



Flexible I/O Generator (FIO)

- Fio makes a good test harness
  - Platform for generating I/O loads
  - Provides a ‘ready-made’ storage application
- Ported fio to a purecap build
  - Only minor fixes required
  - https://github.com/rtegrity/ports-fio
- SPDK includes an fio plugin module
  - Creates a convenient boundary with storage stack
  - Plugin can be built as one shared library or multiple

Performance Testing

- Library compartmentalisation adds a performance cost
  - CPU cycles are useful for tuning the code
  - For I/O key metrics are throughput and latency
- Performance tested against RAM disk
  - Eliminates real I/O latencies
  - Avoids user space to kernel transitions
- Tested with minimal block size (512 byte reads)
  - Focus on the cost of an I/O operation itself
- Represents worst case ‘real-world’ overhead

Resolved Issues

- Library compartmentalisation is experimental
  - Issues are to be expected!
  - Dapeng Gao has been very responsive

- Details:
  - LD_C18N_LIBRARY_PATH undocumented
  - Deadlock loading dependencies
  - Exception accessing thread local storage
  - Calls within a library can pass through the trampoline (build with -Wl,-Bsymbolic)
  - Unused registers not being cleared
  - Tracing of transitions

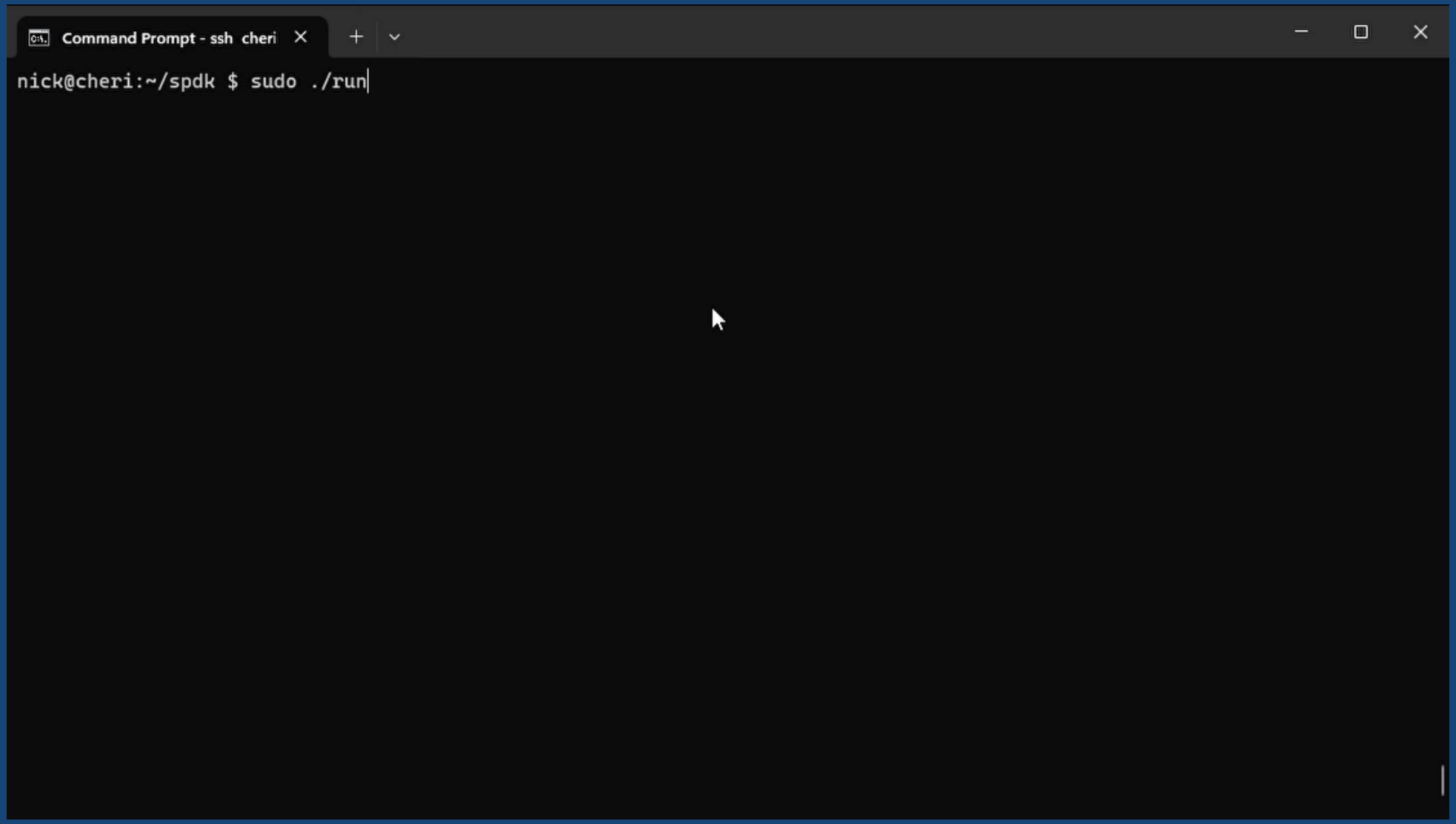
Workaround

- Function calls with more than 8 arguments fail
  - Should be fixed in CheriBSD 23.11
- Change signature to use variadic arguments
  - *function(a1,a2,a3,a4,a5,a6,a7,a8,…)*
  - Use *va_decl / va_arg* to access other arguments
- To identify impacted calls, copy the ELF file to Linux
  - *sudo apt install dwarves*
  - *pfunct -P elf-file | egrep ',.*,.*,.*,.*,.*,.*,.*,'*

Outstanding Issues

- Calls through function pointers are not supported
  - E.g. (*func)(arg)
  - Common idiom for I/O completions
- Stack traces don’t work across compartments
  - Makes it hard to investigate issues
- SPDK and fio both use shared memory
  - Attempts to store capabilities fail

Demo



Conclusions

- Library compartmentalisation is experimental
  - Generally speaking, “it just works”
- Real world performance impact < 5%
  - Code has not been optimised yet
  - CheriBSD 23.11 subsequently released
- Shows significant potential
  - To protect the storage stack and the application
- Changes are in:
  - https://github.com/spdk-morello/spdk [c18n branch]

CheriBSD 23.11

- Only just started investigation – minor issues:
  - Change in DRIVER_MODULE definition
  - e2fsprogs-libuuid (64c) doesn't exist
  - Git (64c) fails to extract system source
- Duplicated previous benchmark results
  - Results seem faster than 22.11 (~18% for 64c)
  - C18n impact seems lower
  - Multi-library results are suspect (call through function pointer)
- Benchmark API is still work in progress
  - Complex build system

QUESTIONS?

Nick Connolly
www.linkedin.com/in/nick-connolly