The Realities of DTrace on FreeBSD

Talk Overview

- History
- Motivation
- Recent Improvements
- How people use DTrace
- Future Improvements

History

- Invented by Sun in 2005
- Ported to FreeBSD in 2008
    - Notes from original porting effort still in public_html folder
- Ported to Mac OS for 10.5
- Maintained separately by each OS team with some cross patching

Motivations

- Improve tracing on large systems
- Distributed Systems
- Use tracing for teaching complex operating systems (U. Cambridge, Robert Watson's course)
    - https://www.cl.cam.ac.uk/teaching/1617/L41/materials.html
- Making it production ready

Crisis Meeting (picture of Mexican standoff from Reservoir Dogs)

- Wall Street high perf computing with three teams: Networking people, Mathematicians, Systems people
- Networking people understand bits/sec, Mathematicians write terrible code (here's this equation …)
    - Each group have a different language
- Tracing is a common language in a large, distributed system

CADETS: Casual, Adaptive, Distributed and Efficient Tracing System

- focus on security, distributed tracing
- (as opposed to putting log files in Splunk)

Ground-Up Local Instrumentation

- compilers
    - LOOM(program driven instrumentation)
    - LLVM IR fat binaries support JIT (re-) instrumentation
- OS: Dtrace scriptable full-system dynamic tracing

Distributed Instrumentation Using DTrace

- Use Case: always-on DTrace
- Poll trace information out of a live system running at full speed
- Dtrace protections can be used against us
- Some improvements necessary
- github.com/cadets/dtrace-scripts

DTrace Design Principles

- No overhead **when not in use**
    - Ship single binary. Suitable for high perf work yet still debuggable
- Never panic the kernel
- Protect kernel at all costs
- D is like C but safe
    - no loops, no basic blocks (only the conditional operator), can't force kernel to loop forever
- Tunings were set for slower 2005 hardware

Running out of steam

- dtrace: 2179050 drops on CPU 0 ...
- "DTrace is broken!"

Tuning

- bufsize: defaults to 4M
    - increase if you have many drops
- switchrate: defaults to 1Hz
    - increase if you have drops
- dynvarsize: defaults to 1M
    - increase if you have many variable drops

Recent Improvements

- Machine Readable Output
    - Traditional DTrace UI is very textual
    - Machine readable output enables different frontends and visualizations
- New Providers
    - FreeBSD audit subsystem: useful for security use case of CADETS
    - mac and Mac_framwork
    - opencrypto
    - sctp
    - xbb
- Performance Analysis
- Documenting the Internals
    - original source code well documented, external Oracle docs less so
    - not just what, but **how** and **why**

Machine Readable Output Demo

- dtrace -O json -n ‘syscall:write:entry’
    - each event includes a timestamp (for visualization etc)

D Language Improvements

- Original design
    - awk like language
    - thread and clause local variables
    - Subroutines to handle common tasks
- copyoutmbuf (added by George)
    - There are no mbufs in Solaris
    - Reading chained mbufs in D
    - Important for network stacks
    - (Dump IP packet example: IPv6 packet starts with 0x45)
- if statements
    - D only has a conditional operator
    - the syntactic-sugar-if still has error message readability problems

Audit Provider

- originally for Common Criteria (Orange Book)

DTrace Performance

- Original design goal: shouldn’t degrade performance
    - Drops Records: not great for security use case
    - kernel can kill tracing under high load
- Solutions
    - our monitor cycle
    - buffer size configurable with sysctl
    - Improve the compiler
        - LLVM

Aside - Loom

- instrumentation framework
- LLVM based
- weaves instrumentation into LLVM IR

- instrumentation defined in policy files
- instrumentation can be done as long as LLVM IR is available

USDT (static userland tracepoints)
USDT Performance
- Problems
    - DTrace tool modifies binaries
    - doesn't play well with make

LOOM based User Tracing
Dynamic Userland Tracing
- Early stage of development
    - sycall (dt_probe)
    - instrumentation via LOOM

DTrace is not the Only One (mentioning other systems for completeness)
- eBPF
    - low level, too primitive
- bcc, ply
    - high level tracing frontends
- Brendan Gregg: eBPF achieved feature parity with DTrace in 2017

DTrace Source Flow
(There's a Windows port of DTrace)
Illumos -> FreeBSD
-> MacOS
-> Linux …

OpenDTrace
- Cross Platform
    - (The way the original DTrace works on ARMv7 is kind of ugly)
- Highly Portable
- RFD (Request for Discussion) Process
- github.com/orgs/opendtrace

OpenDTrace Specification
- Specs of DIF, DOF, CTF (compact trace format)
- Better testing
- support new execution substrates (JIT)
- Allow for clean room reimplementation (e.g. desire to avoid CDDL license)

OpenDTrace Features
- Basic Blocks (George wants if-statements)
- Bounded Loops (joke: there might be a 42 iteration limit)
- Modules
- Higher performance
- Test Suite
- More OS Ports
- Broad Architecture Support
- Finer Grained Libraries
- Usable from other languages: Python, Rust, Go
- Support uC, game consoles, laptops, distributed systems

Applying OpenDTrace
- more kernel subsystem providers

How you can help
- look at the opendtrace organization on github
- checkout docs and source

Q&A

- opendtrace only about DTrace code? A: yes
- interwork with eBPF? A: no. Goal is to crush them (joke)
- Scalability? A: research project not over. DTrace no more or less scalable than other solutions
- Talk more about win32 port? A: no