dnstap: Use Cases and Performance Analysis

Merike Kaeo, CTO
merike@fsi.io

FARSIGHT SECURITY

Discussion Points

- Latest Code Developments
- pDNS and dnstap
- Initial Performance Tests
- Status of dnstap Enabled Systems and Tools
- Use Cases / Wish List
- What’s Next

Changes Since 2016

fstrm (the dnstap transport) C tools and library

- Bug Fixes
  - Make sure fstrm_writer_close() changes writer state to closed even if it’s impossible to send/receive control frames (e.g. when other side loses the socket)
    - https://github.com/farsightsec/fstrm/pull/18
  - Destroy condition variable and mutexes in fstrm_iothr_destroy()
    - https://github.com/farsightsec/fstrm/pull/25

Changes Since 2016

fstrm (the dnstap transport) C tools and library

- Portable fixes (adding OS X compatibility)
  - osx / clock_gettime(): https://github.com/farsightsec/fstrm/pull/21
  - osx / pthread_condattr_setclock(): https://github.com/farsightsec/fstrm/pull/35
- New functionality
  - Add support for output file splitting in fstrm_capture
    - https://github.com/farsightsec/fstrm/pull//28
  - Add support for '-' as a filename for stdin/stdout to support a request to be able to accept input from standard input, so he can compress and decompress the files before feeding them into dnstap
    - https://github.com/farsightsec/fstrm/pull//22

Changes Since 2016

golang-framestream (go implementation of dnstap transport

- Implement bidirectional protocol needed for UNIX domain socket transport
  - https://github.com/farsightsec/golang-framestream/pull/2
- Bug fix to the above, in current master branch

golang-dnstap (dnstap utility written in go)

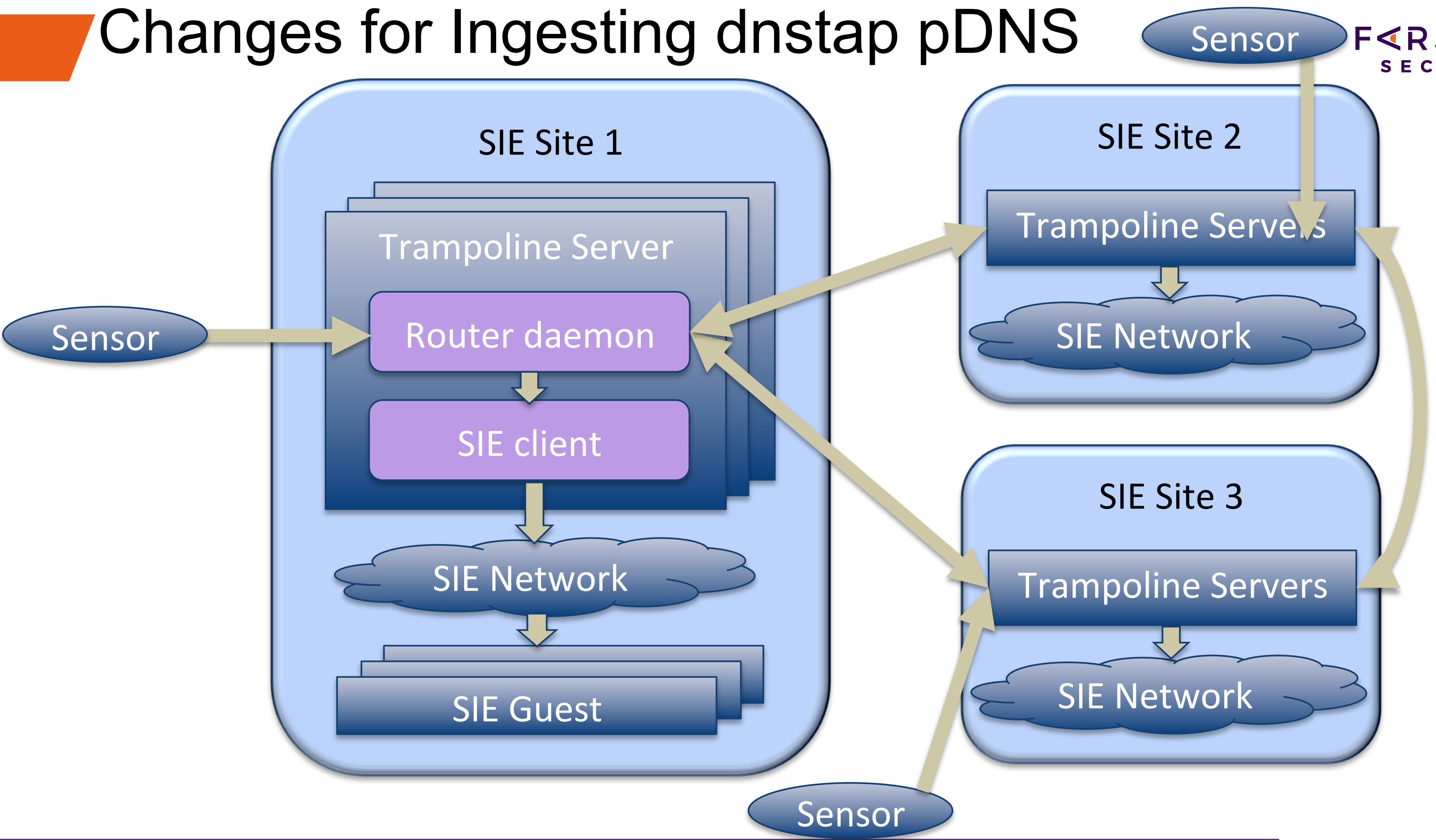
- Re-enabled UNIX domain socket functionality, making it usable for logging again.
  - https://github.com/dnstap/golang-dnstap/pull/4
- Enabled output file rotation with SIGHUP reopening
  - https://github.com/dnstap/golang-dnstap/pull/6

Changes To Injest dnstap Enabled pDNS

FARSIGHT SECURITY

- New NMSG payload format to carry dnstap payloads
- The dnstap-sensor packs the appropriate dnstap messages into those payloads
  - Type RESOLVER_RESPONSE
- Deduplicator now fetch responses from
  - dnstap-in-NMSG payloads
  - dnsQR-in-NMSG payloads
- Anyone processing dnstap data from NMSG broadcasts needs to pick out a different field from message
- Many fields in dnsQR are recreated in the dnstap type to smooth over transition (QNAMR, RRType, etc)

Changes for Ingesting dnstap pDNS



Want To Know Performance Implications

- Expected DNS Query Logging Improvements
  - Should be faster due to eliminating bottlenecks like text formatting and synchronous I/O
- Expected pDNS Improvements
  - Avoid complicated state reconstruction issues by capturing messages instead of packets since DNS server and network stack already doing:
    - Bailiwick reconstruction: DNS server already knows which servers are responsible for which zones
    - DNS query/response matching: DNS server already has its state table so unsuccessful spoofs are excluded
    - UDP fragment reassembly
    - UDP checksum verification
    - TCP stream reassembly

Performance Testing Goals

- Initial thoughts
  - Compare CPU impact of dnstap code in recursive name server
    - Use baseline with no sensor enabled
    - CPU usage of dnstap enabled sensor
    - CPU usage of pcap-based sensor
  - Compare CPU usage with different receiver choices
    - None
    - fstrm_capture
    - golang-dnstap
    - dnstap-sensor
  - Measure rate of queries the recursive resolver makes
  - Compare resultant size, completeness and contents of captured data

Test Considerations

- Initial thoughts
  - Use controlled traffic to external authoritative servers using ch 202 DITL
  - Possibly use AWS to set up recursive resolvers
- Where we ended up
  - 100% internal test set-up
  - Developed standard test case with synthesized queries and a repeatable query data set

Want to limit as many external variables as possible although at some point you also want "real world" data

Generating The Traffic For Testing

FARSIGHT SECURITY

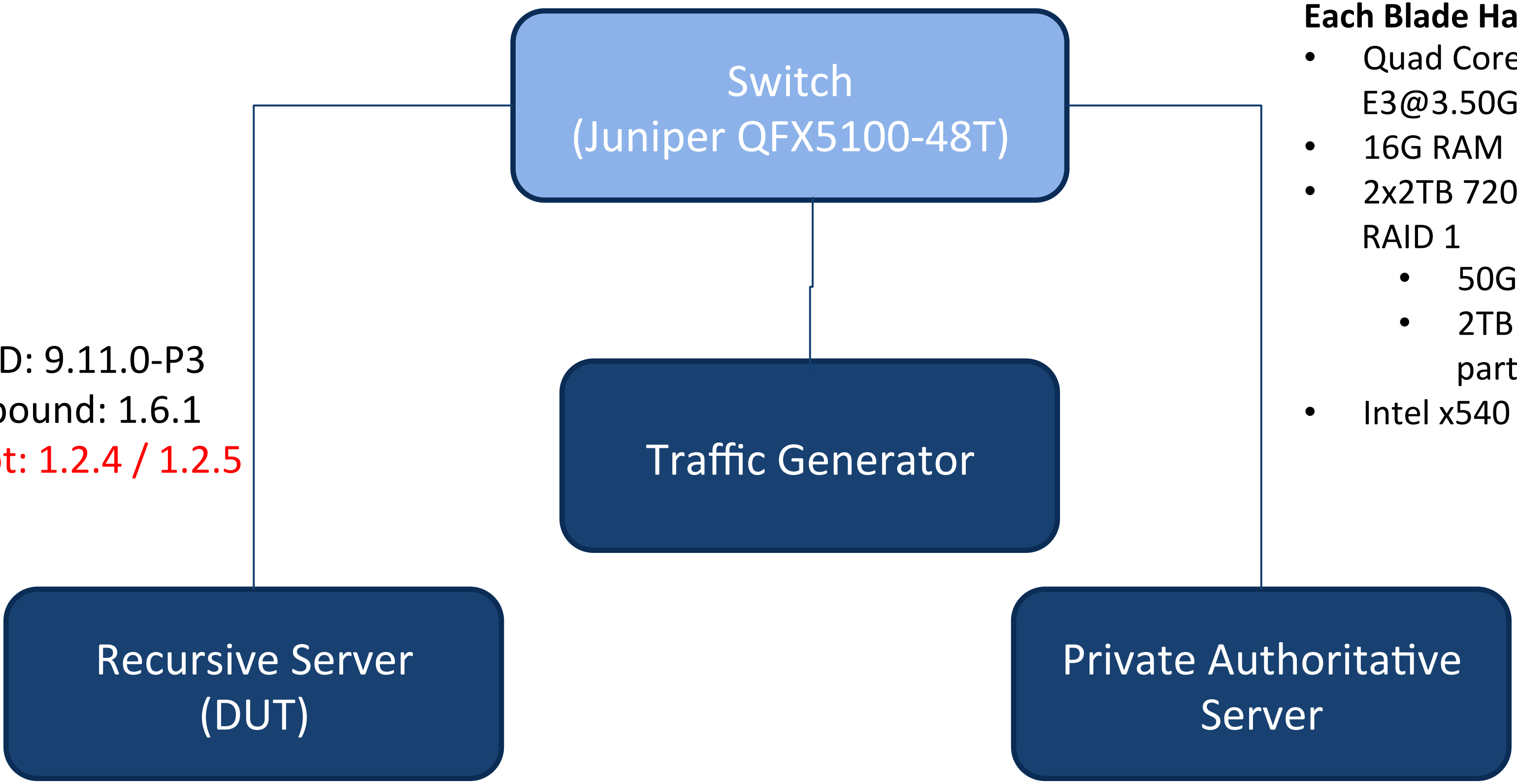
- Bash script created that generates an arbitrary number of FQDNs resembling a PRSD attack on a given zone
  - Take a stream of random binary data (from the Linux urandom(4) PRNG)
  - Convert the stream into lowercase alpha-numeric strings of length
  - Prepend the string to our domain suffix, that is ".prsd.fsi.io".
- The PRSD dataset totals 10M unique FQDNs

```
~/dnstap$ pigz -dc datasets/prsd10M.gz | head
 5bj0kurfrtl071aeodhm8gyga15ef3ho.prsd.fsi.io
ze17z5warbm5ms5kw00shz1y0h4b8aqt.prsd.fsi.io
eo3qzffgl7rnni90wep0ff4ac5czgln6.prsd.fsi.io
oudxwevzko3awf4qtjk7yw5f1gd6gyil.prsd.fsi.io
72acpomki8vzcodtzmbuud0nr1xo1v0e.prsd.fsi.io
zwauxsjhjlzquuxsy8ymur5gpcvxd0l1.prsd.fsi.io
2own8tpwjvsvec52gi4jw9g1sayfmtdl.prsd.fsi.io
```

Testing Nuances

- Each thread uses single source UDP port
  - At any moment in time there are at most 10 distinct source UDP ports in use for sending the queries.
  - The 100,000 queries per second are uniformly distributed among the threads, so the server should see 10,000 queries per second from a single UDP port.
- Each thread restarted after random delay of 5-10 min (uniform distribution between 5-10 mins)
  - When a thread is re-started it stops sending new queries, but a new thread is started immediately to replace it.
  - Then, the UDP listening socket is closed after 3 second timeout.
  - This means the UDP source port will change, but again, and any given point time, there are at most 10 distinct source UDP ports in use for sending the queries from the whole looker-upper process.
- The requests are written to kernel in groups of 1,000 datagrams each 10ms.
- The write/read procedures are completely decoupled and run in parallel.
  - The write process sends a query as soon as it's made available to its input, so it's completely unaware of what happens on the receiver listening on the socket (as long as it exists).

Test Set-Up



BIND: 9.11.0-P3
Unbound: 1.6.1
Knot: 1.2.4 / 1.2.5

Each Blade Has:

- Quad Core Xeon E3@3.50GHz
- 16G RAM
- 2x2TB 7200 RPM disk, RAID 1
  - 50GB root
  - 2TB storage partitions
- Intel x540 10Gbps NIC

Test Methodology

- Method A: tcpdump
  - Use name service configuration files without dnstap enabled
  - Start name service
  - Execute tcpdump to collect DNS requests from client resolver
  - Collect resource metrics [via sar, dstat, etc)
  - Execute PSRD attack
- Method B: dnstap
  - Use name service configuration files with dnstap supported
  - Start name service
  - Start fstrm runit service
  - Collect resource metrics [via sar, dstat, etc)
  - Execute PSRD attack

Performance Comparisons

- CPU
- Memory considerations
- Other noteworthy parameters found in testing

Some Questions On Results

- The results appear useful, and are consistent with dnstap having negligible impact on CPU usage but they leave me with a couple questions:
  - The BIND-tcpdump CPU graph has a very different "shape" toward the end of the run, where CPU usage drops by about half, before receding to zero + noise. Was there an issue with the test run, or is there an explanation for this? (Note that the corresponding graph for unbound does not show this feature)
  - The looker-upper log files for the dnstap runs are consistently slightly smaller than those for the corresponding tcpdump test runs. I haven't deciphered the log contents yet, but could this indicate dropped queries or lower throughput in the dnstap runs?

Performance Testing

- How can you trust the numbers and results you see?
- Can we / should we create a testing methodology standard at DNS-OARC?
- What are noteworthy parameters to be tested?

DNSTAP Wish List: Recursive Server

- More fine-grained instrumentation
  - Ability to audit records that have been received **and accepted** by a recursive DNS server into its cache
  - Don’t have any visibility into a new NS RRset overwriting and existing NS RRset for the same name

DNSTAP Wish List: Authoritative Server

- When responding authoritatively, knowing what QNAME is and what zone was it answered from
  - Can point to some misconfiguration issues they see with delegations when a zone and subzone registered (anything below delegation point but part of parent zone should be ignored)
- Policy information as to why an authoritative answer was given
- ACL aware and view aware
- Debugging historical behavior
- Added reporting on recursive query load

Use Case: Threat Detection and Response

- Question
  - If packet gets flagged for malicious content it is flagged by source IP (e.g. 1.2.3.4)
  - Investigate by asking ‘how did user get to src IP ?’
  - Generally this is by DNS reference (e.g. “evil-host.tld”)
  - Logs so far reference responding authoritative server; it mentions "evil-host.tld" but gives the authoritative nameserver IP address that responded with the A record, not the contents of the A record, so 1.2.3.4 is not present in my logs.
- One Reply
  - The data you're looking for is going to be included in the DNS answer section of the DNS response message payload
  - There might be more than one A record for "evil-host.tld" in the response

Decoding DNS Logs

- Three Known Dedicated Tools
  - Dnstap-ldns (standalone)
    - https://github.com/dnstap/dnstap-ldns
  - Golang-dnstap (standalone)
    - https://github.com/dnstap/golang-dnstap
  - Dnstap-read (included with BIND)
    - https://ftp.isc.org/isc/bind/9.11.0a1/doc/arm/man.dnstap-read.html
- All by default use compact one line per message representation
- All have capability for a more verbose format
  - Enabled with the '-y'parameter
  - Includes full decode of the DNS response section

Status of dnstap enabled Systems and Tools

- OS Updates
  - CoreDNS server will get dnstap support
    - https://twitter.com/miekg/status/860254715078201344
  - Knot patch released for Recursive Resolver (version1.2.5)
- Tool Updates
  - Would there be interest in importing DNSTAP data into DSC?

QUESTIONS ?

FARSIGHT SECURITY