



Challenges in Answering Infosec Questions at Scale



Alexander Barsamian
John Murphy

Answering Infosec Questions for Large Networks

- NetFlow-based analysis
- “At scale” == 100 Gbps and up
 - ~125 K flow updates per second +
- In the form of five items that NetFlow can be useful for, which become difficult at these scales

Where Do the Constraints Come From?

- Questions that are difficult at large scale:
 - Can data exfiltration be detected under DDoS conditions?
 - Can we trace the communications of a compromised host? (network epidemiology)
 - When we identify a Command and Control host, can we figure out who else on our network is talking to it?
 - Can we identify hosts performing likely reconnaissance?
 - How can we keep alerting timely for mitigation under high-volume events like DDoS?
- These and similar questions define the needs of the problem, and the shape of the approach

Finding our constraints

FIVE QUESTIONS

Problem: Data Exfiltration Under DDoS

- Find the needle when the haystack is particularly large and possibly designed to hide needles
- Any magician will tell you: big movements hide small (important) movements
- DNS tunneling is hard enough to find, but what about in the middle of a DNS reflection attack?
- Find the successful SSH logins during a brute-force attack

Problem: C2 Tracking

- Identifying outside IP addresses and ports used to command and control hosts in a botnet
- Because of changes in botnet operation, frequently requires a seesaw investigation:
 - Which IP addresses contacted this known server?
 - Did they use any unusual ports?
 - Who *else* did they contact on those unusual ports?
 - ...
- Impossible to know ahead of time (pre-infection) what we'll need to look up later

Problem: Compromised IP Tracing

- First indication of a compromise may come very late
 - Takes weeks, on average, to discover a compromise
- After compromise is discovered, questions arise:
 - “Who else in my network might have been hit?”
 - “What services on my network did this host start using after the compromise?”
- Time element critical
 - Essentially building a Feynman diagram of an infection
- Context, context, context

Problem: Finding Reconnaissance

- Basic scanning behavior: who contacted the most unique service endpoints in my network?
 - Who **inside** my network?
- How do we collect the raw information needed to form a statistical baseline?
 - What looks like scanning behavior for one host might be normal for another
 - Average number of service endpoints (host x port) contacted over a short stretch of time

Problem: Timely DoS Alerting

- In addition to historical investigation, live mitigation is increasingly desired
 - Signaling analysts by email “Deal with this now!”
 - Triggering automatic mitigation
 - Offsite scrubbing (requiring BGP changes)
 - In-line configuration changes
- Looking for a wide variety of attack conditions
 - Not just large packet volume, but resource starvation at the OS level
 - We all know what it looks like: the trouble is making sure the automated analysis stays fast enough and doesn’t choke
 - Frequently only interested in the last minute’s worth of traffic

Constraints from Questions

- Full-fidelity record
 - Aggregation is useful for many things, but the critical evidence may be a single session, possibly just a few packets
- Long histories
 - Hurts when multiplied by high bandwidth, but no getting away from the need to look at data months-old
- Interactive analysis
 - Can't predict what you'll need ahead of time; need to be able to refine searches ad-hoc
- Most recent data needs to be snappy

Constraints from the sheer amount of data

NETWORK SIZE

“At Scale”

- Those questions can be difficult enough on small networks
- Add to the overall problem of network analysis on large networks
- 10 Gbps, 100 Gbps, and up
 - ISPs, large service providers, some very large corporations
- On a given day, a typical host contacting our network might send 30 MB of data and receive 200 MB, say 300k packets switched per user

1D Problem

- Keeping up with incoming rates
 - Roughly 25k updates per second per 10 Gbits of traffic
- 4 GHz CPU has 4B cycles per second; at 100k flows per second, 40K CPU cycles per flow
 - Just a hashtable lookup 2.5K cycles
- Some basic limits on incoming NetFlow data
 - Just receiving/unpacking data, not storing or doing anything with it
 - 120 Kfps – limit of a single CPU thread in unpacking NetFlow
 - Multi-core is a must
 - 800 Kfps – equivalent of 1 Gbps incoming flow records
 - Limit of gigabit cards
 - 2100Kfps
 - Limit of a PCI2.0 card using 10 Gbps interfaces

2D Problem

- Bandwidth x Time
- Single bi-flow record ~200B
- Volume of incoming data (especially during DDoS attack)
 - Multiple updates can be consolidated into a single timestamped record
 - ~200 GB / day / 10 Gbits (10Gbit/s => 25k fps)
 - ~6 TB / month / 10 Gbits

Why NetFlow Instead of pcap?

- Full pcap would be wonderful, but at these scales just isn't practical
 - Hard to store
 - Hard to search
- Also limited utility, thanks to increasing use of encryption

NetFlow Basics

- Acts as a pen register for the network
- Size is around 1/1000th the size of the same traffic's full packet capture
 - Varies by traffic type
- Universally available (if not NetFlow, J-Flow or IPFIX)
 - sFlow also an option, though it has its downsides
- Many potential vantage points into the network
 - Router, firewall, switch, even load-balancer

Not Psychic

- Didn't know yesterday everything we needed for today's analysis
- Can't create filters or stream processing rules to pull out everything we need



ALL TOGETHER

Overview

- Identified five big questions (among others) which gave us our constraints
- Identified the bottlenecks we'd face
- In the process of finding a solution that fit our constraints, we made other problems harder

Three Problems

- In order to answer those five questions, three problems came up
- General flow analysis at scale
 - Receiving, storing, retrieving, and processing flow data north of 100K flows per second
- Top unique at scale
 - A processing problem, finding a general-purpose “rank entity type X by unique count Y”
- Disk storage
 - Disk is always a bottleneck – what challenges does it add?

Just getting data in and asking a few questions

GENERAL ANALYSIS AT SCALE

Description

- Problem is deceptively simple:
 - Take in flow data from multiple sources
 - Store it for random-access retrieval up to months later
 - Perform retrieval as quickly as possible
 - Make data available for general analysis, selecting data according to:
 - The time frames in which they occur (start or end time within specified bounds)
 - Various aspects of the session record, particularly IP
- How did we solve this for > 1M flows per second?

Why It's Interesting

- This kind of general analysis underlies everything else we want to do using NetFlow data
 - Simple questions at small scales become hard at large scale and often require specialized solutions
 - Keeping it generalized retains flexibility
- The better we can perform this task, the larger scales we can tackle
- Fast interrogations of network behavior enable analysis to get a “feel” for the network

Why It's Hard at Scale

- Bottlenecks in processing
 - Hard limits on flow rate
- Expense
 - RAM is much faster than disk, but also much more expensive
- Hardware limitations
 - If we decide we need 80 cores, 30 TB of RAM and 2000 TB of disk for the task, can't just shove it all in one chassis and call it a day
- Next year there'll be more

Amdahl's Law

- **Make the common case fast**
- What's the common case?
 - Recent data (last day) accessed most often
 - Human analysts
 - Automated tools (e.g. alerts)
 - Beyond that, fewer patterns:
 - The further back in time analysts go, the harder it is to draw conclusions about what they're most frequently looking for
 - IP addresses most common

Solution in Two Parts

- Local and Global
- Global
 - Hierarchical / Waterfall-style cluster
 - Exploits ability of individual hosts to take in much more flow data than they can individually handle
- Local
 - Three-level FIFO queue structure exploits “common cases” to allocate resources



TOP UNIQUE AT SCALE

Description

- Want a list of some entity by the unique values of a particular field:
 - E.g. top IP address by destination port
 - For each host, count the number of unique destination port numbers it sent data to (regardless of which host that port was on?) and show me the hosts with the highest counts
- More in-depth analysis than “Top X by Bytes”

Why It's Interesting

- Gives a useful view into potential reconnaissance behavior, specifically scanning
- Horizontal scans
 - Rank IP address by number of unique partners
- Vertical scans
 - Rank IP address by number of unique ports contacted
- Can be used to create effective profiles of certain types of behavior
 - e.g. web browsing behavior characterized by many external partners with a narrow set of ports

Why it's Hard at Scale

- Parallelizing the problem is necessary, but as always adds complications
 - The relevant sessions involving IP A might be scattered across individual nodes
 - Each node has a limited view of the data
 - For efficiency, want to do this in one bottom-up pass
 - Can't just mindlessly combine
 - Much of the data on each node may be redundant
 - E.g. Host A contacted port 80 eighty times, and three out of four nodes has a record of that

Basic Top N Construction

- With all the session records in one place, the process is simple. For example:
 - Keep a list of the entities you're ranking
 - E.g. IP address or port number
 - Iterate through all records
 - Insert or update the item's record in the list
 - When finished, sort, trim to N, and return

IP A	13 GB
IP C	12 GB
IP D	10 GB
IP F	7 GB
IP G	6 GB
IP Q	5 GB

Multi-Node Top N Reconstruction

- With multiple independent nodes, becomes tricky
- Need an assumption:
 - No duplicated records between nodes
- “Trim to N” step becomes problematic

IP A	13 GB
IP C	12 GB
IP D	10 GB
IP F	7 GB
IP G	6 GB
IP Q	5 GB

+

IP Z	8 GB
IP B	7 GB
IP C	6 GB
IP A	6 GB
IP H	5 GB
IP Q	5 GB

?
=


IP A	18 GB
IP C	18 GB
IP Q	10 GB
IP D	10 GB

Top 4

Top Unique, 1

- Top Unique is a special case, not as parallelizable
- Building a single list is easy

IP A TCP/80
IP B TCP/443
IP B TCP/80
IP A TCP/80
IP C TCP/53
IP C UDP/53



IP C	2
IP B	2
IP A	1

Top Unique, 2

- Compiling two summaries not possible

IP C	2
IP B	2
IP A	1

+

IP C	2
IP B	2
IP A	1

?
=

IP C	4
IP B	4
IP A	1

IP C	2
IP B	2
IP A	1

IP B	4
IP C	2
IP A	2

IP A TCP/80
 IP B TCP/443
 IP B TCP/80
 IP A TCP/80
 IP C TCP/53
 IP C UDP/53

IP A TCP/80
 IP B TCP/22
 IP B TCP/25
 IP A TCP/443
 IP C TCP/53
 IP C UDP/53

Top Unique Compilation

- Need to pass more information along in order to combine Top Unique lists... but only enough to make the determination

IP A TCP/80
IP B TCP/443
IP B TCP/80
IP A TCP/80
IP C TCP/53
IP C UDP/53



IP C	TCP/53, UDP/53
IP B	TCP/443, TCP/80
IP A	TCP/80

DB STORAGE AT SCALE

Description

- How do we design storage systems to support the forensics and analysis requirements, given what happens “at scale”
- Some core questions
 - What do you store
 - How do you store it
- Flow data is a lot more compact than fullcap
 - Good news, but can we improve on it?

Aggregation

- Aggregation: bins and buckets
 - Do all your analytics in time slices that fit in RAM, store the results, and toss the raw data
 - Upside: DB size is a function of history length only, not flow rate
 - Downside: you wind up throwing away the needles *and* the hay, and just saving the contour of the haystack. Not what you want for infosec.
- So maybe we went too far...

Storing all flow

- What kind of access patterns does this generate?
 - Seek-heavy, random small writes in one region (the file or files corresponding to the last few minutes/hours)...
 - ...long, sequential reads from another, when an analyst does a historical query.
 - This is a shockingly challenging workflow for spinning media, even with smart schedulers and big mmap io windows. Much easier for NAND.
- Disks that show 200 MB/second of throughput in large sequential reads will return 2 MB/second of throughput in a mixed random read/write scenario!

OS differences

Kernel IO scheduler is “deep magic”

- Same hardware, same workload (10k rpm HDD, 10kfps, 50/50 Cr to U ratio)
 - 48MB/sec for Linux 3.2
 - 17MB/sec for Windows 2012 Server
 - PrefetchVirtualMemory => 20Mb/sec
 - 5MB/sec for FreeBSD 10.0.
- Azure cloud: same ratios, different numbers
- Same workload, different hardware (SSD):
 - 220MB/sec for all!

Too many cooks

- 12 disk iSCSI array, Linux, 4MB/sec (wat.)
 - Turned off the scheduler (/sys/block/DEV/queue/scheduler) => 60MB/sec
 - Sometimes two “smart” algorithms cancel out each others’ benefits

Dirty Tricks: Disk Compression

- Our record format is compact (192 bytes), but lots of the things we are storing don't have a ton of entropy
- Observed gain with ZFS on Linux (LZJB): 3.3x
- Double whammy: 3x the history, 3x the performance!
- Some CPU cost, but querying is much more IO bound than CPU bound
- Not all algorithms created equal: NTFS actually got slower when we turned on encryption