Data mining for security at Google

Max Poletto
Google security team
Stanford CS259D
28 Oct 2014

Background
Monitoring
Analysis
Discussion

Why security at Google?

- Hundreds of millions of users trust Google with their data
- Billions of users trust Google search
- Massive computing footprint
- All manner of adversaries, from “script kiddies” to nation states
- All manner of attacks
  - From DDoS to politically-motivated targeting
  - Big range of frequency, sophistication, severity

⇒ Vast range of security problems

Large security team

- Product consulting: design to launch to bug bounty
- Infrastructure: auth*, systems hardening, logs
- Operations: vulnerability management, detection, response
- Threat intelligence: malware, indicators
- Privacy: special focus on unauthorized access to end-user data

Data mining used pervasively

Some examples:

- Account hijacking detection
- Click fraud detection
- DoS detection
- Infrastructure compromise detection

Data mining for monitoring and analysis

My team, secmon-tools, focuses on this

- Monitoring
  - Automated, continuous, feeds data to analysts
  - Things to look for: intrusion, exfiltration, privacy violation, ...
- Analysis
  - Not necessarily continuous
  - Often initiated by humans
  - Applications: threat intelligence, incident investigation, cleanup, ...

Important: “actors” are Google employees, not end users

Caution: security is a process

Any technology (data mining, etc.) is only a tool, not a solution

- User education (social engineering is surprisingly successful)
- System hardening (auth, secure engineering, timely patches, ...)
- Operational procedures
  - Adapting to growth (new hires / platforms / acquisitions)
  - Maintaining alertness (in the absence of major incidents)
  - Gathering intelligence
  - Escalation and response playbooks

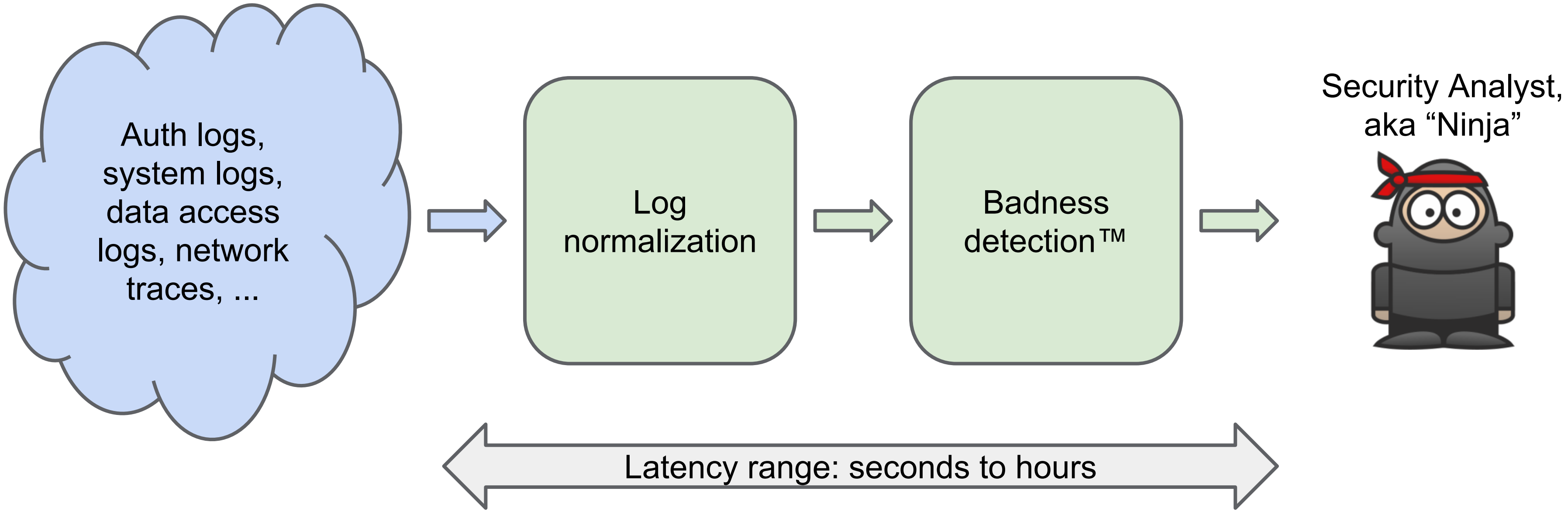
Background

Monitoring

Analysis

Discussion

High-level view of a monitoring pipeline



Some guiding principles

- False negatives are very expensive
  - Could cause arbitrary damage to our users
- False positives are expensive too
  - Analyst time is valuable
- Alerts should make sense to a human
  - The analyst (security expert) is key
  - False positives + inexplicable results → signal fatigue

Log normalization is underappreciated

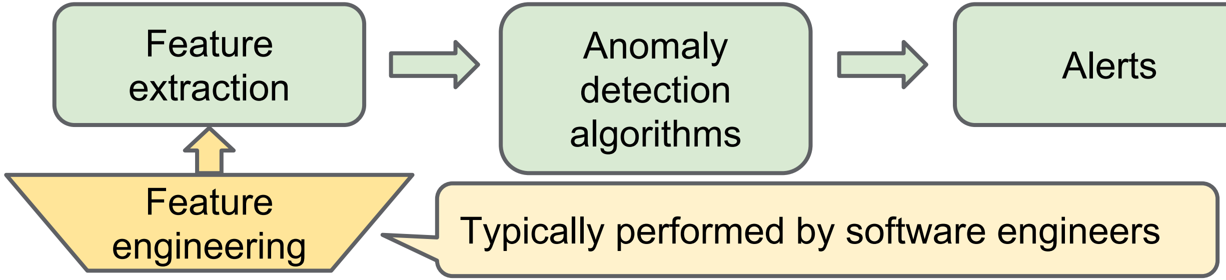
Analysis capability is limited by quality of underlying data

- Timestamps with missing or incorrect timezone
- Different names for the same thing: “GOOGLE\\maxp” vs “maxp.corp.google.com”
- One event spread across multiple log lines: e.g., sshd and PAM entries during an ssh login

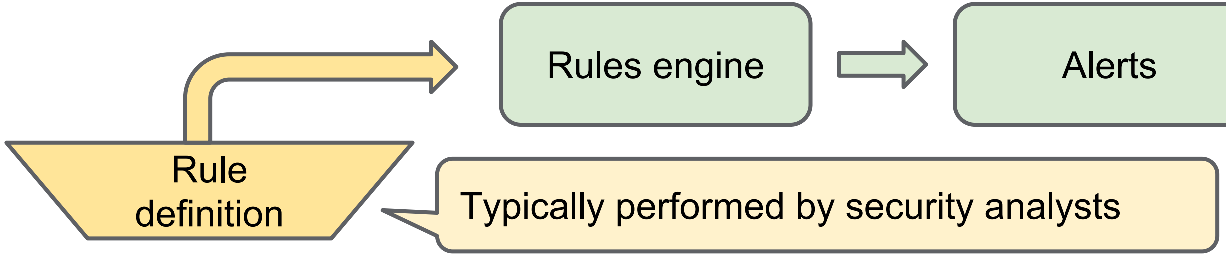
Sounds trivial, but takes a lot of engineering to get right and maintain

Two forms of “badness detection”™

1. Statistical (e.g., machine learning) ✘ poor results for us



2. Rule-based (e.g., expert system) ✔ good results for us



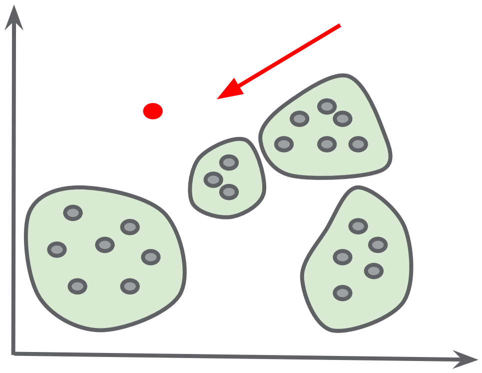
Statistical anomaly detection

Not easy to model attacks

- Huge attack space
- Few training examples

Intuition: model normal behavior, find outliers

- Pro: many training examples
- Pro: theoretically, ability to detect new, unanticipated attacks
- Cons: noisy and hard to interpret



Example: detecting anomalous actor behavior

Some reasons to care

- Employee account hijacked by malware?
- Intentional malicious activity?

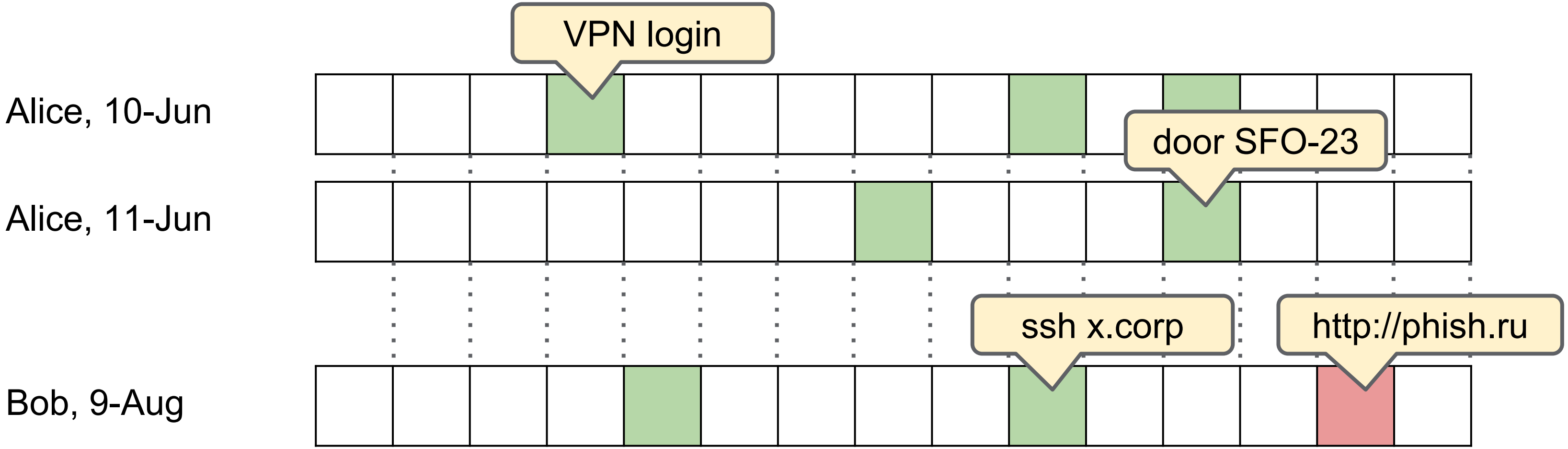
Goal: model actor behavior, find anomalies

What do we need to do

- Identify useful features
- Model normalcy
- Find outliers

Feature extraction: modeling actors

- Partition logs by actor and time
- Represent (actor, time) pairs as vectors of binary variables



Modeling normalcy and finding outliers

Need to find low-probability features or combinations of features
Many approaches possible. Some examples:

- Boltzmann machines, weighted histograms
  ⇒ probability model for features or pairs of features
- Nearest neighbors
  ⇒ similarity metric between actors
- “Strange pairs”
  ⇒ features that rarely appear together

Modeling normalcy: nearest neighbors

Intuition: find users that are not very similar to any other users

- Look at fraction of shared features
- Compare to users in same group / department / etc.

Variant: compare a user to her past

- “Neighbors” are feature vectors in user’s past
- Identify changes in behavior

Modeling normalcy: strange pairs

Intuition: identify users with pairs of features that occur frequently individually but rarely together

E.g., “accessed source code” and “works in HR”.

- Assume independence of features. Expect common features to occur frequently together
- Find ones that don’t
- User’s anomaly score is the sum of the “strangeness” of all her variable pairs

How well does all this modeling work?

Not well enough in our case, it turns out.

Top ~1% of users by anomaly score includes all “bad actors”
But ~50K Google employees → top 1% ≈ 500 users!

And, crucially, anomaly scores are difficult to explain to analysts
⇒ signal fatigue!

Important: not all anomalies are attacks

Former employee requests an authorization token

- Account revocation bug? Attack?
- Nope: username typo

Actor fails authentication 20K times

- Brute-force attack?
- Nope: actor changed password, forgot to update script

Email address in RPC to location service

- Privacy violation?
- Nope: address is “test@123.com”

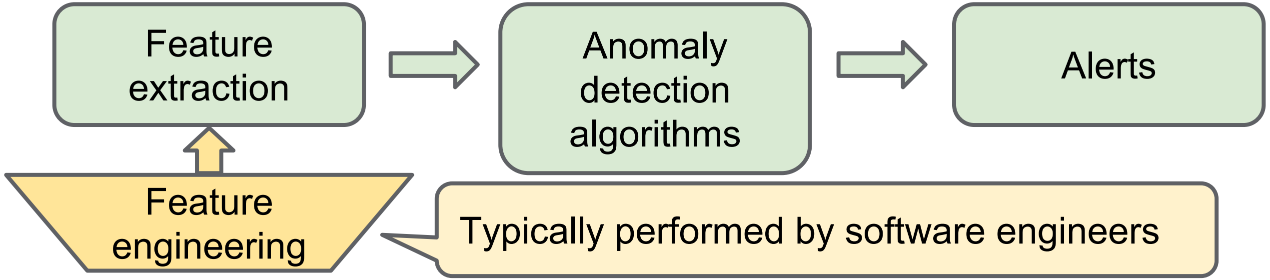
Why is statistical anomaly detection for security hard?

	Learning	Cost of error FP / FN	Goal	Attacker
Anomaly detection	Unsupervised	Medium / High	Classify & ***explain***	Adaptive
Spam detection	Supervised	High / Low	Classify	Adaptive
Product recommendation	Supervised	Low / Low	Classify	N/A

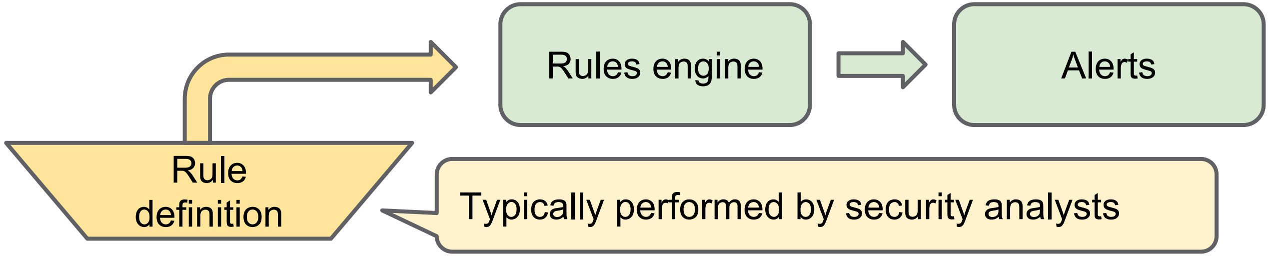
For more on this topic, consider "On Using Machine Learning For Network Intrusion Detection", Sommer and Paxson, Oakland 2010.

Two forms of “badness detection”™

1. Statistical (e.g., machine learning)



2. Rule-based (e.g., expert system)



An alternative: rule-based detection

Manually created rules

→ Characterize attacks or deviations, not normalcy

- Locality: each rule covers a small set of logs and features
- Explainability: direct connection between rules and alerts
- Specificity: make better use of analysts' expertise

Example rule ideas

- "Alert if a host appears to be ssh probing"
- "Alert if a host connects to a suspicious IP shortly after downloading a PDF"

Encode security experts' knowledge into many such rules

Rule quality

Can have high S/N ratio, but brittle if written carelessly

Best rules cannot be trivially disabled by changing a parameter
Context is valuable: conjunction of terms, temporal logic, etc.

Consider previous examples

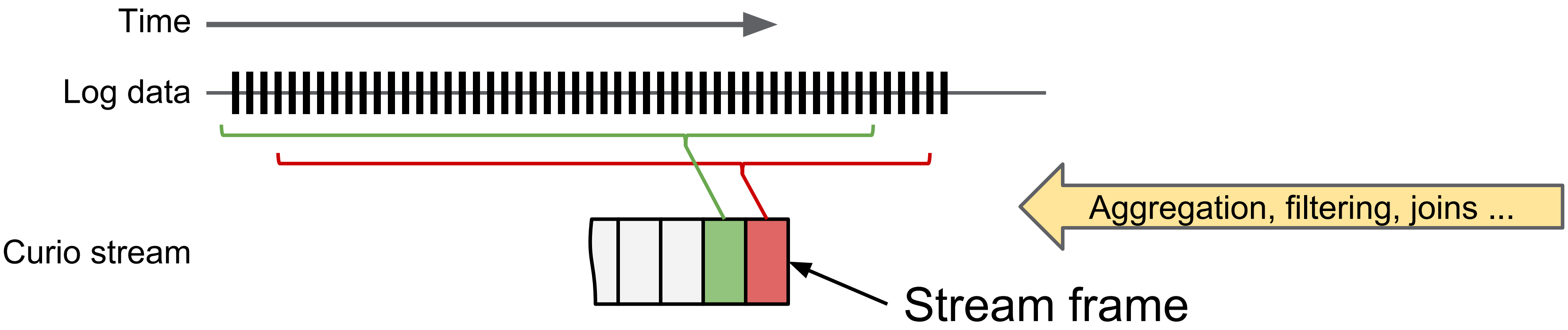
- Former employee login: measure Levenshtein distance
- Brute force failures: consider network connection history
- Email address privacy: use dynamic whitelist

How to implement the rules?

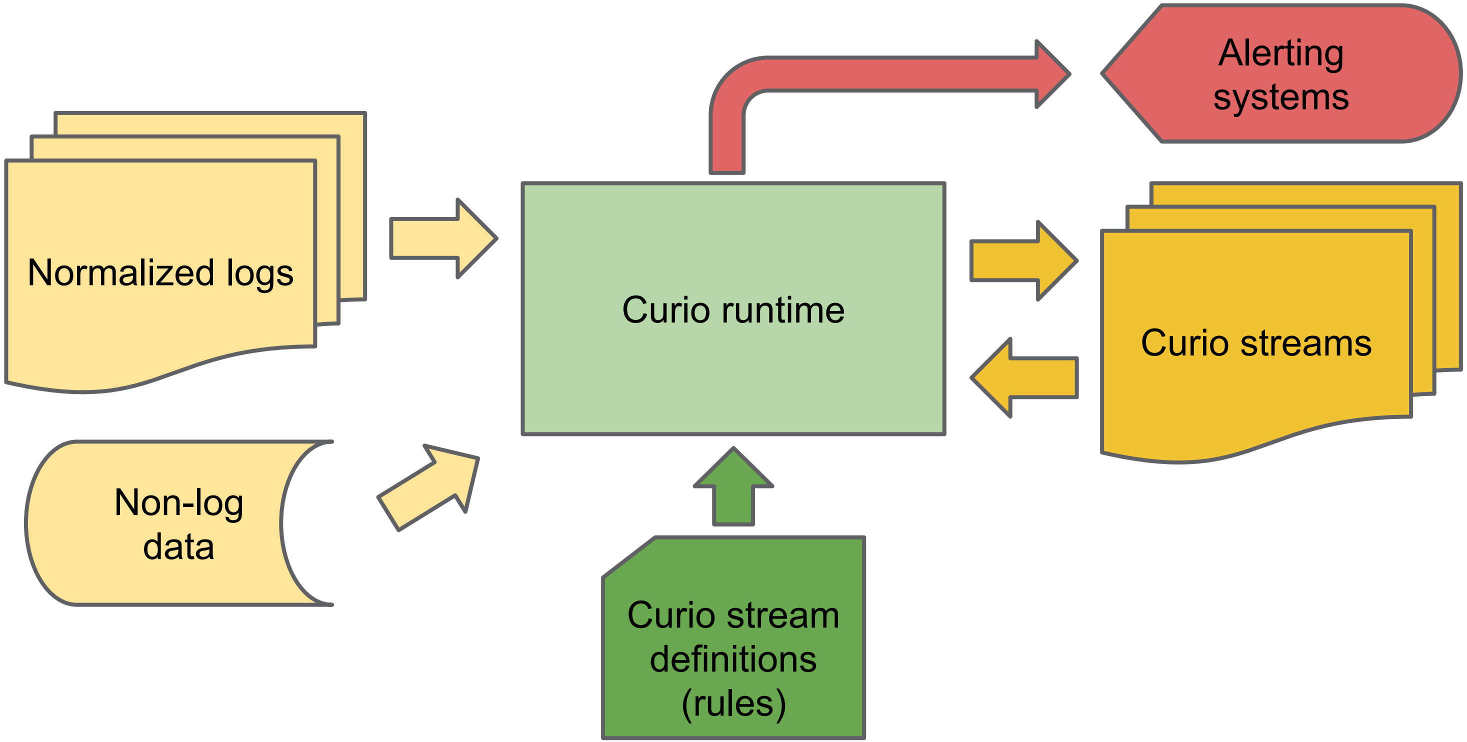
- Ad-hoc code (e.g., Python, C++)
  - Pros: it can do anything
  - Cons: complex, hard to maintain
- SQL database
  - Pros: easier, more expressive
  - Cons: problems with temporal logic; poor match for log workloads
- Domain specific language for processing streaming logs
  - Pros: sliding time windows; temporal logic
  - Cons: implementation is not easy

Curio, a system for continuous data processing

- Built on top of Dremel (Melnik et al., VLDB 2010)
- Aggregates *streams* into *frames*, collections of records corresponding to a time interval
- Enables temporal analysis and correlation



Curio architecture



Nice Curio features

Scalability

- Shards to very large queries

Resilience

- Handles job failures seamlessly
- Adapts to source log delays

Integration with reporting systems

- Sends alerts to the right places automatically

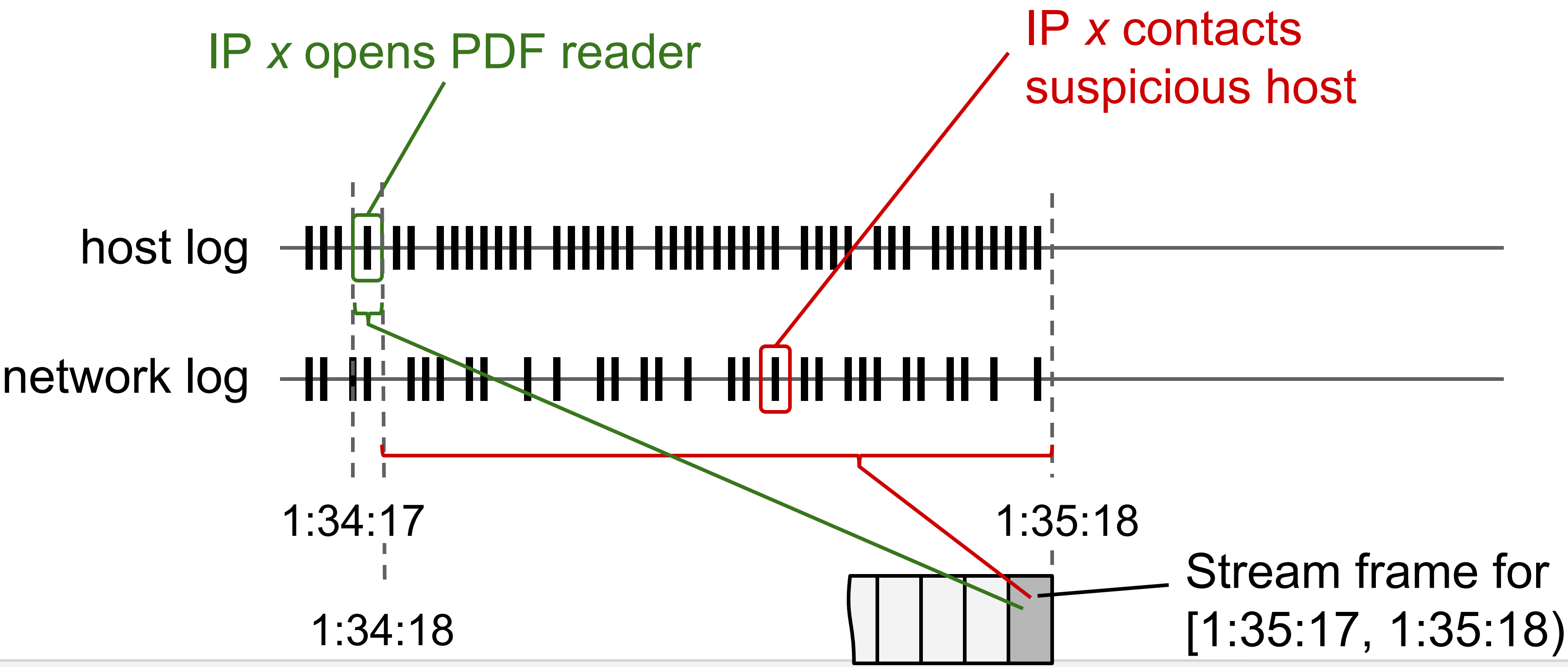
Quick case study: detect PDF spawning malware

Alert if host contacts “suspicious” IP within 1 minute of opening PDF

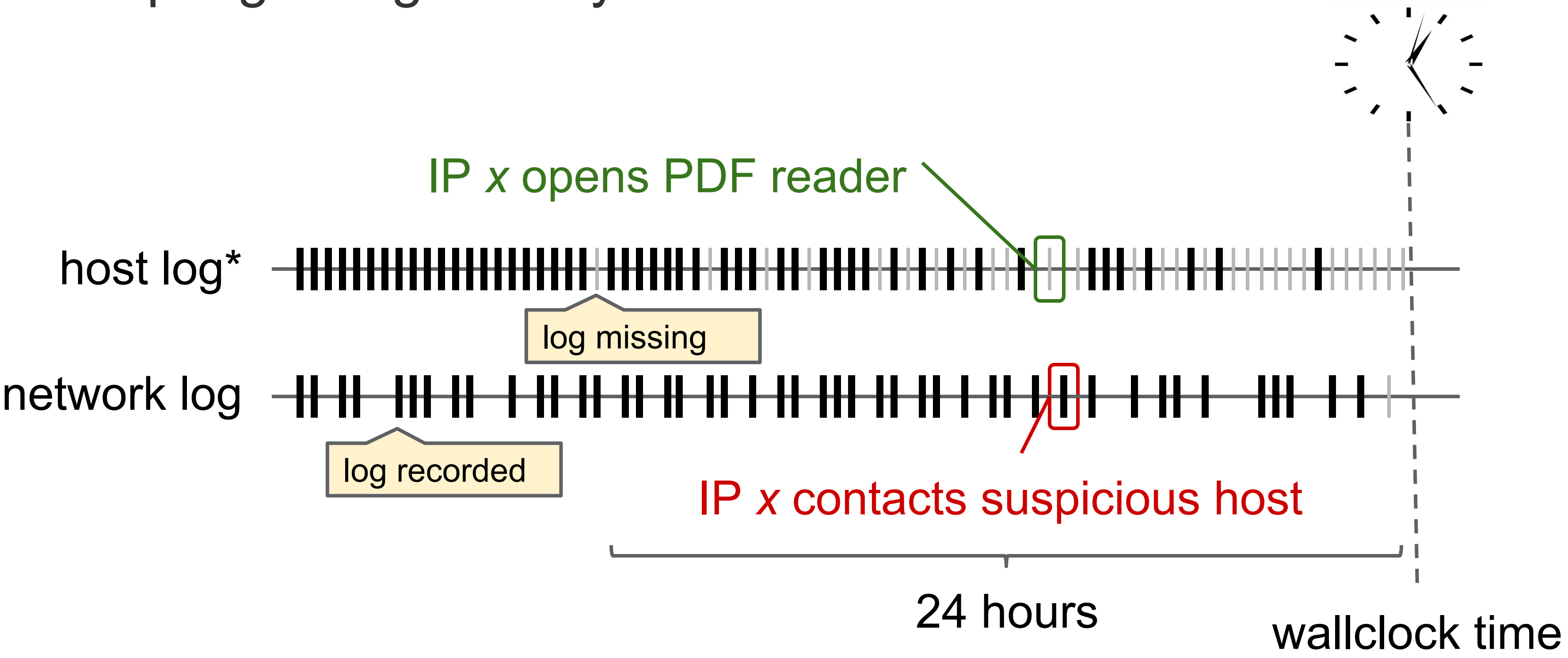
Ground truth:

- host execution logs
- network logs

Quick case study: detect PDF spawning malware



Adapting to log latency



*Missing entries because *x* unreachable

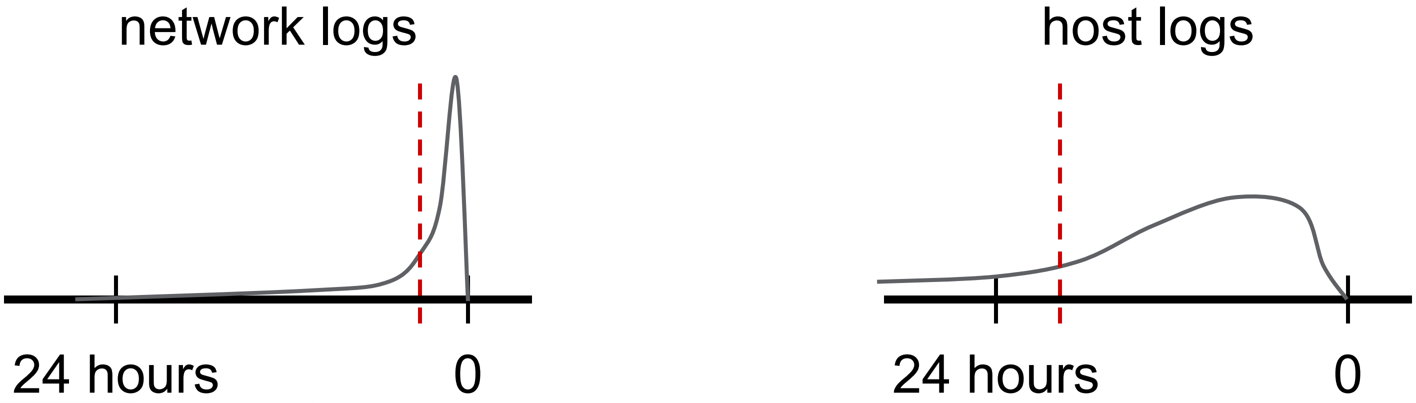
Adapting to log latency

Maintain a histogram of:

delay = (event in logs) - (event time)

Use it to decide when to advance each stream

Results available as soon as data is “reasonably” complete



How well does this all work?

Cautiously optimistic

- Many streams and signals
- Knowledge encoded from scores of analysts
- Seems effective, but beware unknown unknowns

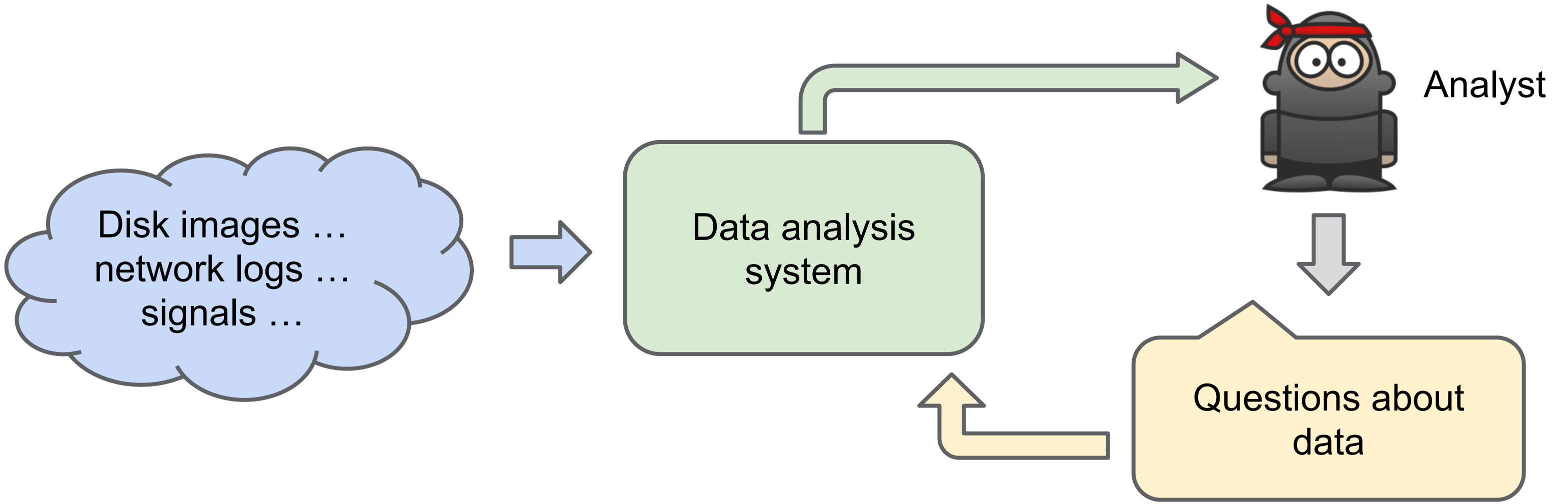
Quality measures are crucial

- Well-defined process for launching new signals
- End-to-end tests to detect “log rot”
- Open-ended penetration tests

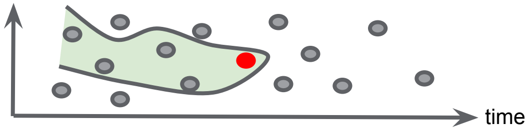
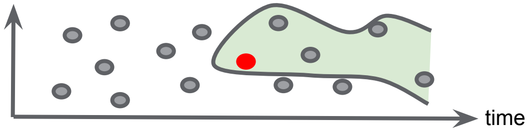
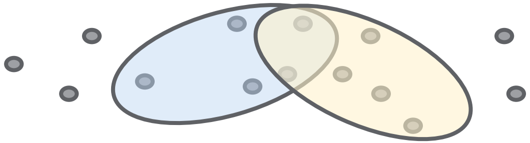
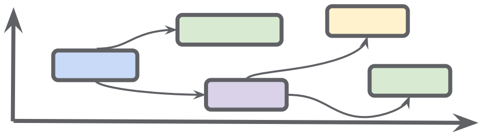
Background
Detection
Analysis
Discussion

Data mining for security analysis

It's not all about automated detection. Skilled analysts are a valuable resource: give them the tools to use their time effectively



Questions an analyst might ask

Causation	How did the attacker get root?	
Consequence	What was the effect of running the script?	
Correlation	Which signals fired simultaneously?	
Summarization	What was the user doing last night?	

Broad spectrum of tools

- Looking for causes and effects
  - graph traversal
- Extracting meaning from noisy data
  - graph summarization
  - clustering
- Triaging malware
  - classification

Statistical (as well as graph-based) approaches are effective in this problem domain
Let’s look at three examples

Example 1: graph traversal for incident investigation

Some questions to answer:

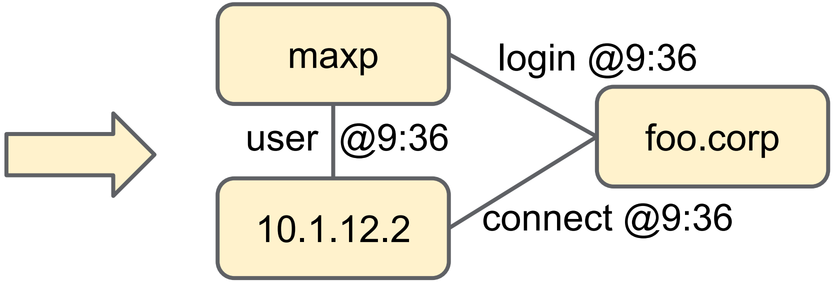
- Were any machines affected by watering hole X?
- User U downloaded malware. What should be cleaned up?

Given a graph representation of all relevant logs, can be framed as a large-scale graph search problem

Graph creation

Log lines induce graph components

Oct 20 9:36:30 foo.corp sshd[29661]: maxp login from 10.1.12.12 port 65298



Edges annotated with times and semantics

Many different log sources in one huge (peta-scale) graph

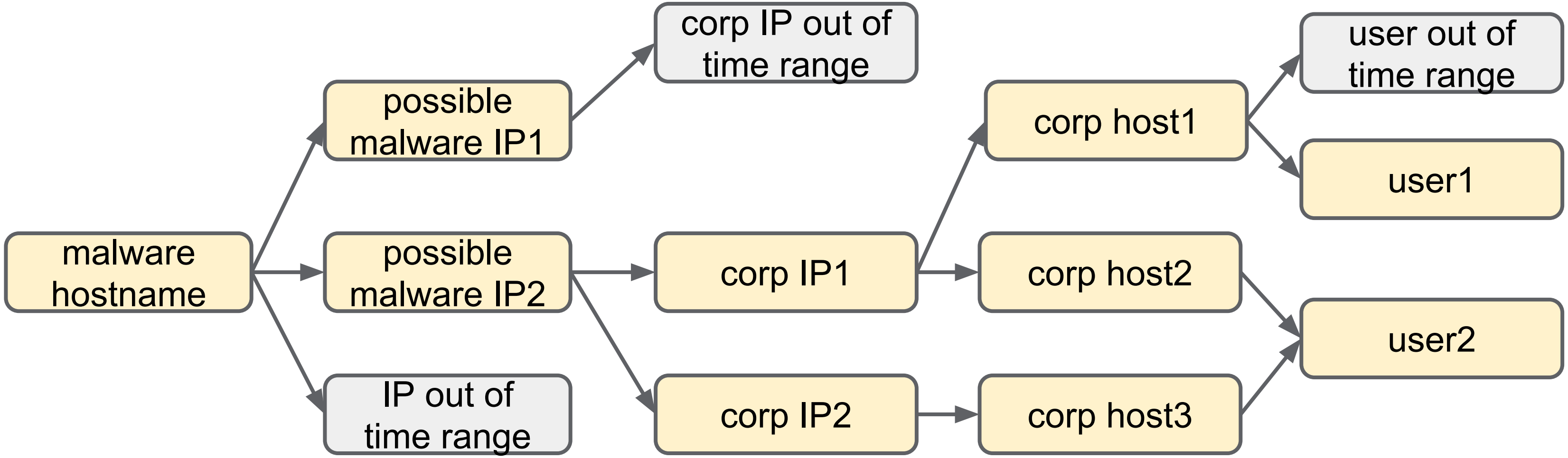
Data normalization again an issue: “maxp” vs “maxp@google”, etc.

Sample graph query

Given watering hole hostname X ...
→ IPs that it resolved to
→ internal IPs that talked to them
→ machines (assets) those internal IPs belonged to
→ users who used those machines
→ other machines those users have logged into

Hours of manual research replaced by a ~10-second query

Sample graph query: time constraints



Search for potentially compromised users during a time interval

Some implementation insights

Keep the graph as close as possible to ground truth

- Limit data pre-processing
- Global graph corrections are expensive

Most of the work is in query encoding and execution

- Guiding and constraining the search is a challenge
- Some edges may invalidate others (e.g., DHCP leases)
- Parallelism is your friend

Broad spectrum of tools

- Looking for causes and effects
  - graph traversal
- Extracting meaning from noisy data
  - graph summarization
  - clustering
- Triaging malware
  - classification

Example 2: log summarization via graph transformation

Many logs are so verbose that humans cannot make sense of them.

"Can't see the forest for the trees"

Example: Plaso (https://github.com/log2timeline/plaso)

- Open-source forensics tool
- Produces detailed timeline of all artifacts from disk image
- Useful when investigating a compromise.

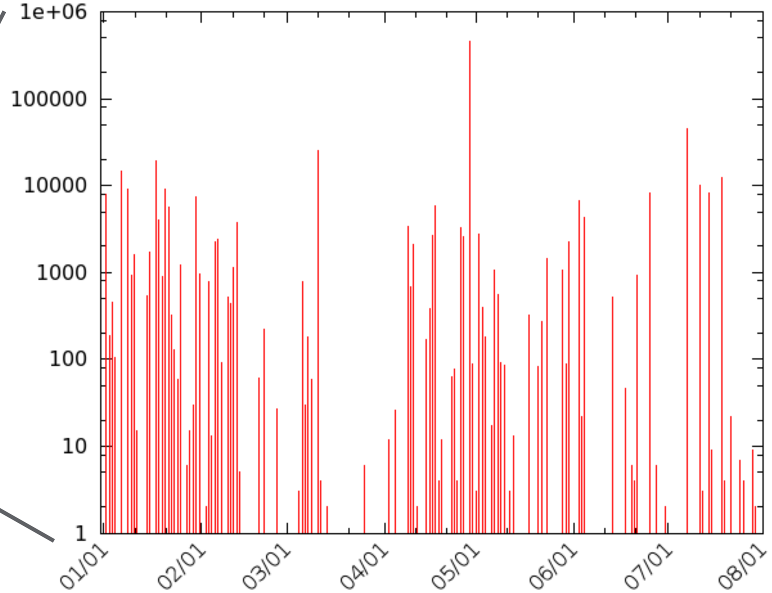
But...

Plaso logs look like this...

Timestamp	Desc	Message/Source
2013-07-15T18:29:16.382852	Page Visited	https://www.google.com/search?q=kristinn+gudjonsson&oq=kristinn+gudjonsson&aqs=chrome.0.57.2638j0&sourceid=chrome&ie=UTF-8 (kristinn gudjonsson - Google Search) [count: 0] Host: www.google.com (URL not typed directly - no typed count) /Users/demouser/Library/Application Support/Google/Chrome/Default/History
2013-07-15T18:29:42.966055	Creation Time	MessageID: 1428 Level: NOTICE (5) User ID: 501 Group ID: 20 Read User: ALL Read Group: 80 Host: Macintosh.local Sender: UserEventAgent Facility: messagetracer Message: com.apple.message.domain [com.apple.usage.app_activetime: com.apple.message.signature][loginwindow: com.apple.message.signature2][com.apple.loginwindow ||| 8.2 (8.2): com.apple.message.value][38: com.apple.message.value2][NO: com.apple.message.summarize],asl_log, /private/var/log/DiagnosticMessages/2013.07.15.asl,Document Printed,1,369849

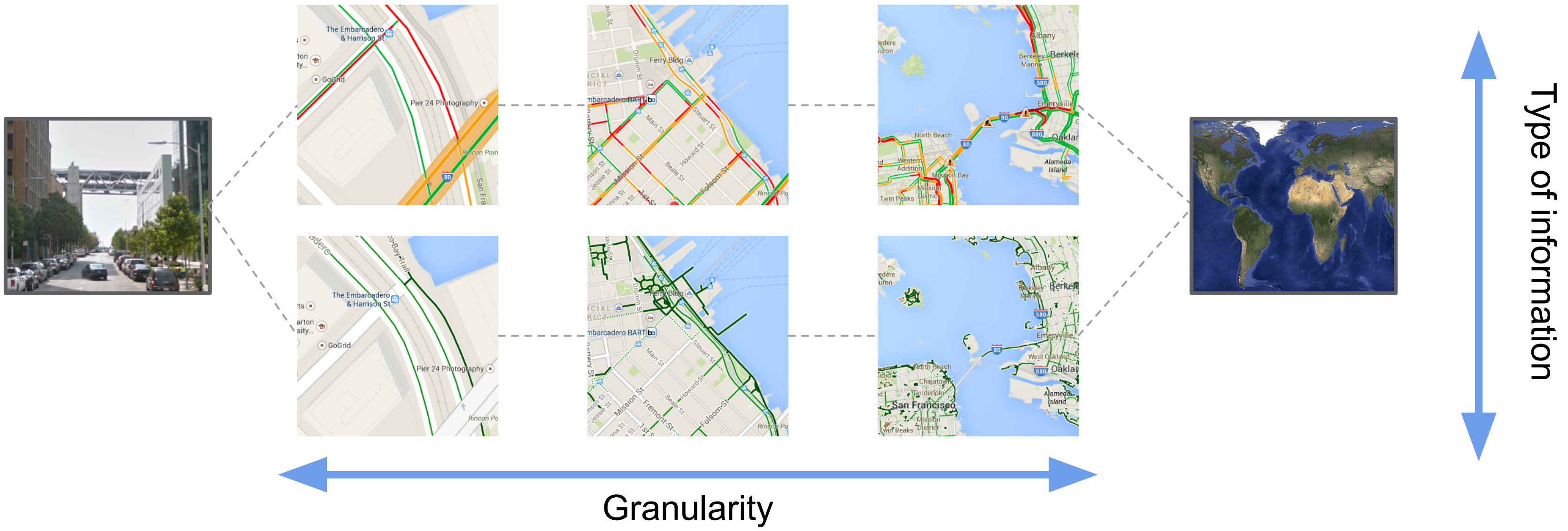
… and there are many of them!

Compressed Plaso dump	67MB
Number of Events	
Total	1,118,757
01 Jan to 30 July 2013	710,812
15 July, 6:27PM to 6:59PM	8,140



Ideally, multiple perspectives on log data

- Granularity
- Semantics (e.g., time order vs ownership)



Log summarization

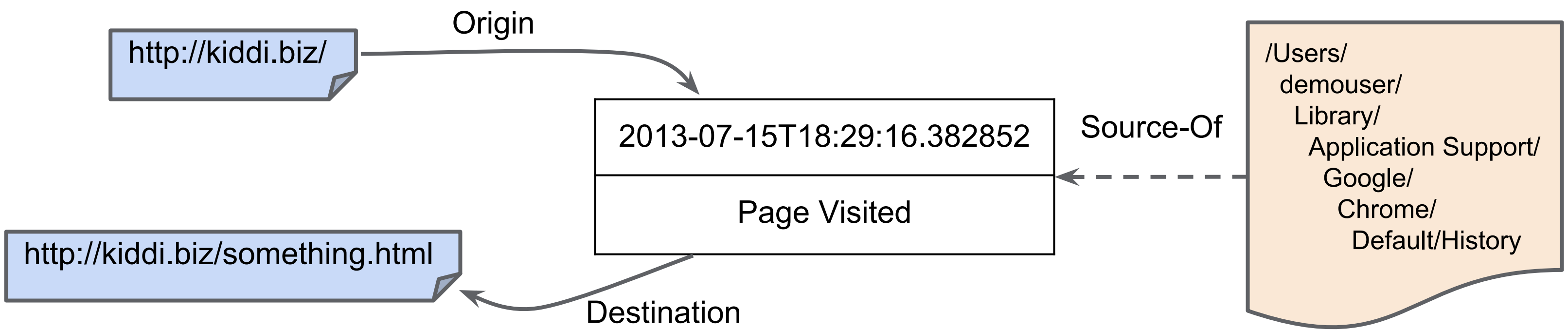
As with graph traversal, convert logs to a graph

But then don’t stop at ground truth

Transform (minimize) the graph to extract meaning

Relationships define edges

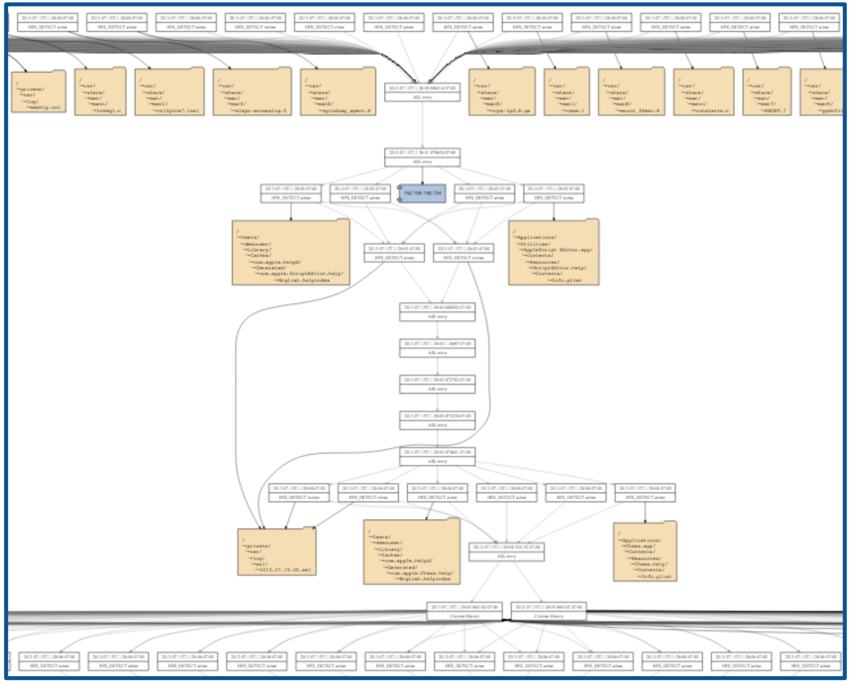
Timestamp	Desc	Message/Source
2013-07-15T18:29:16.382852	Page Visited	http://kiddi.biz/something.html (Some Randomly Generated Web Site) [count: 0] Host: kiddi.biz Visit from: http://kiddi.biz/ (Kristinn) (URL not typed directly - no typed count) /Users/demouser/Library/Application Support/Google/Chrome/Default/History



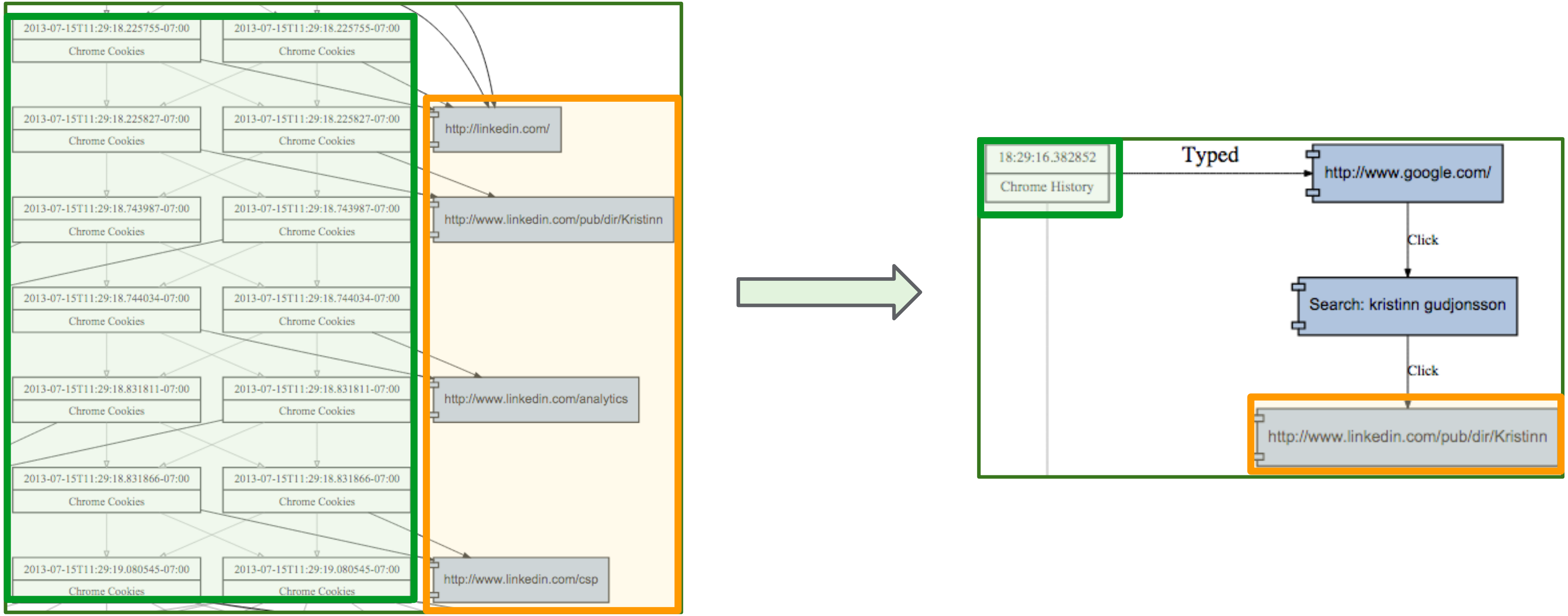
Temporal relationships also define edges

Graph for 1 minute fragment	
Log events	2,544
Timestamps	147
Nodes	4,825
Edges	5,753

Large, unreadable graph, but temporal structure is obvious



Graph minimization merges and re-labels nodes

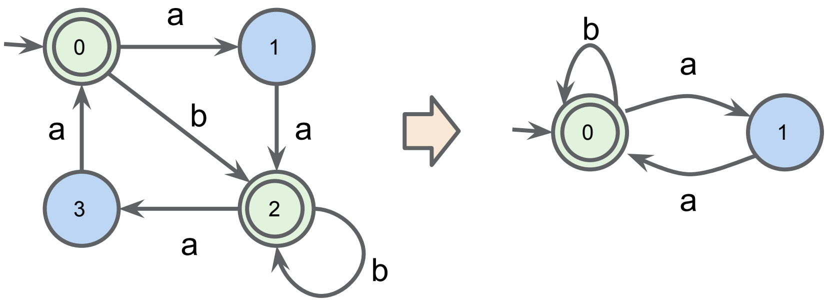


Graph minimization mechanics

Given a graph and a condition for equivalence, find smallest graph that preserves structure of G and merges equivalent nodes
⇒ "relative coarsest partition problem"

Area of research in automata theory and model checking, starting with Hopcroft's automata minimization algorithm, 1971

(aa|b)*:



Examples of equivalence conditions

Timestamps → intervals

- All ~identical operations within a time range are collapsed to one node

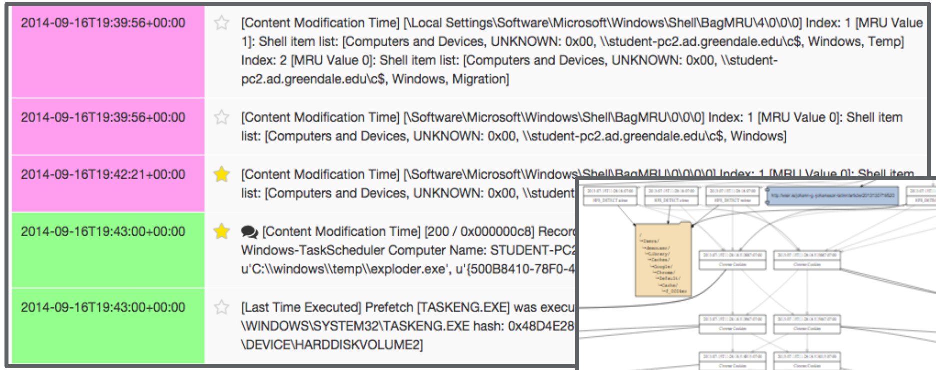
URLs → domain name

- All visits to pages on a domain are collapsed to one node

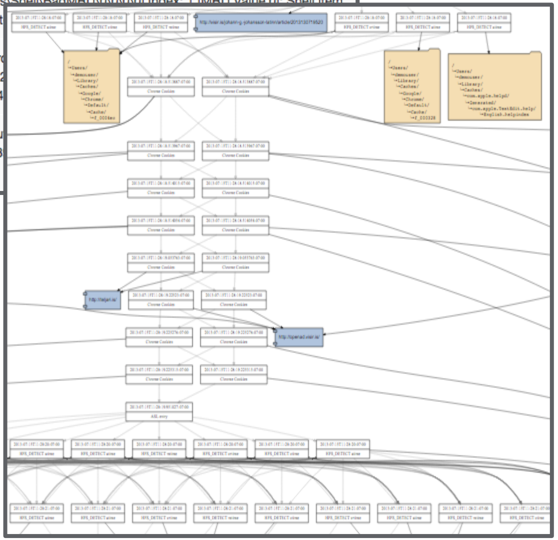
Subgraph → operation

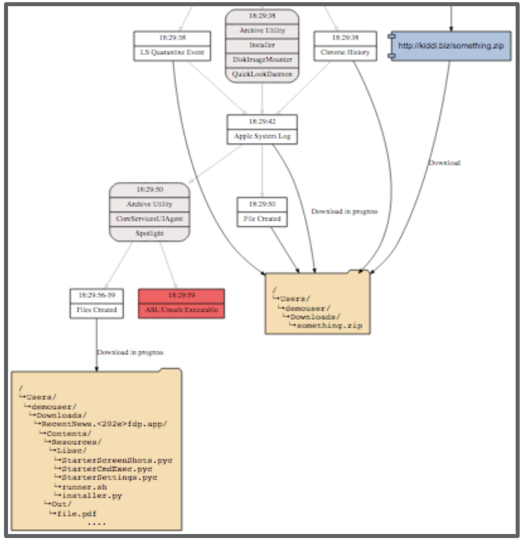
- Subgraphs corresponding to a high-level operation (file open, process exec) are collapsed to one node

Log summarization outcome



2014-09-16T19:39:56+00:00	[Content Modification Time] [\Local Settings\Software\Microsoft\Windows\Shell\BagMRU\4\0\0\0] Index: 1 [MRU Value 1]: Shell item list: [Computers and Devices, UNKNOWN: 0x00, \\student-pc2.ad.greendale.edu\c$, Windows, Temp] Index: 2 [MRU Value 0]: Shell item list: [Computers and Devices, UNKNOWN: 0x00, \\student-pc2.ad.greendale.edu\c$, Windows, Migration]
2014-09-16T19:39:56+00:00	[Content Modification Time] [\Software\Microsoft\Windows\Shell\BagMRU\0\0\0] Index: 1 [MRU Value 0]: Shell item list: [Computers and Devices, UNKNOWN: 0x00, \\student-pc2.ad.greendale.edu\c$, Windows]
2014-09-16T19:42:21+00:00	[Content Modification Time] [\Software\Microsoft\Windows list: [Computers and Devices, UNKNOWN: 0x00, \\student
2014-09-16T19:43:00+00:00	[Content Modification Time] [200 / 0x000000c8] Record Windows-TaskScheduler Computer Name: STUDENT-PC2 u'C:\\windows\\temp\\exploder.exe', u'{500B8410-78F0-4
2014-09-16T19:43:00+00:00	[Last Time Executed] Prefetch [TASKENG.EXE] was execu \WINDOWS\SYSTEM32\TASKENG.EXE hash: 0x48D4E28 \DEVICE\HARDDISKVOLUME2]





Broad spectrum of tools

- Looking for causes and effects
  - graph traversal
- Extracting meaning from noisy data
  - graph summarization
  - clustering
- Triaging malware
  - classification

Example 3: malware classification

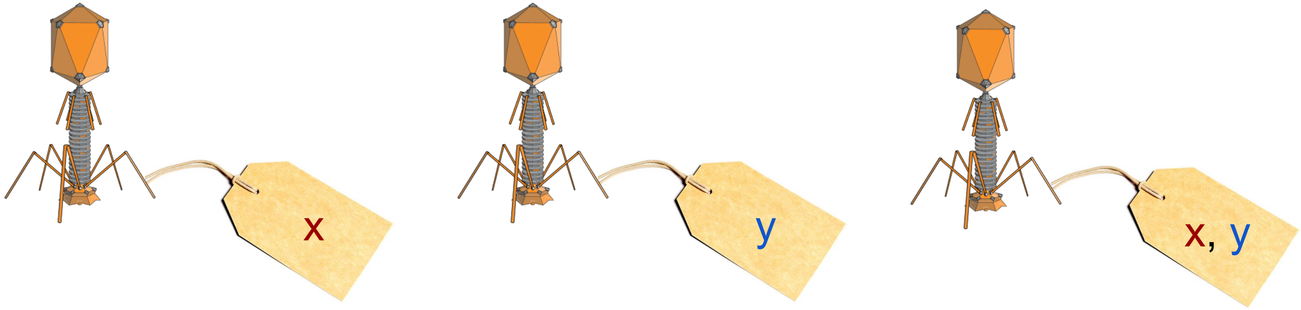
Given a binary, is it malware? If so, what kind?

Non-exclusive taxonomy: “labels”, not “folders”

Why is this useful?

- Incident triage – is this malware we should care about?
- Robust hunts and scans in the presence of polymorphism

Malware samples, indicators, and families



Each *sample* is an executable. It has *indicators* (features) from static and dynamic analysis (e.g., basic block structure, registry changes, ...)

Malware in training corpus also has one or more labels (from manual labeling, A/V signatures, etc.) denoting its *families*

Requirements

- Make use of labeled data → supervised learning
- Classify samples by family → N-ary classification
- Non-exclusive taxonomy → samples with multiple labels
- Compare samples → meaningful metric
- Summarize important indicators → weighting of features
- Scale: thousands of families, millions of indicators

Choosing a learner

Some options and their pitfalls

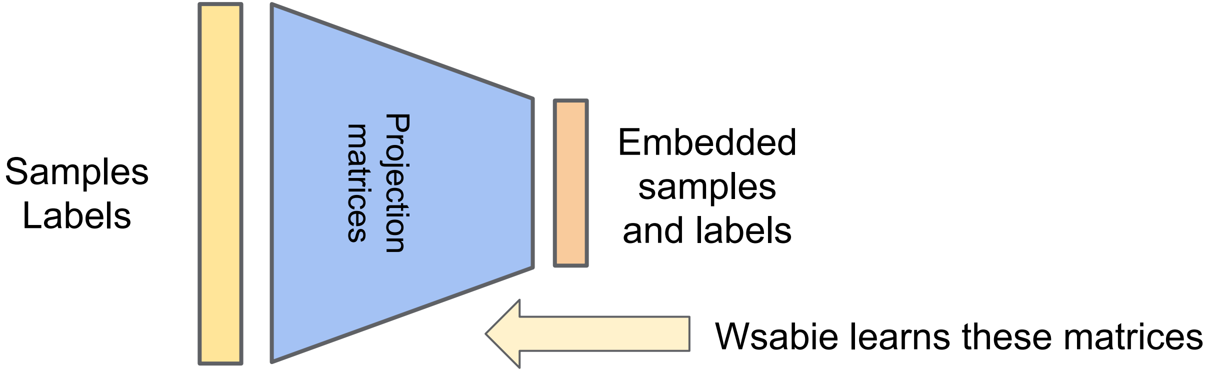
- Manual signatures: don’t scale
- k-Means: unsupervised, loses valuable label information
- Logistic regression: no similarity metric between samples

Final choice: Wsabie [Weston et al., IJCAI 2011]

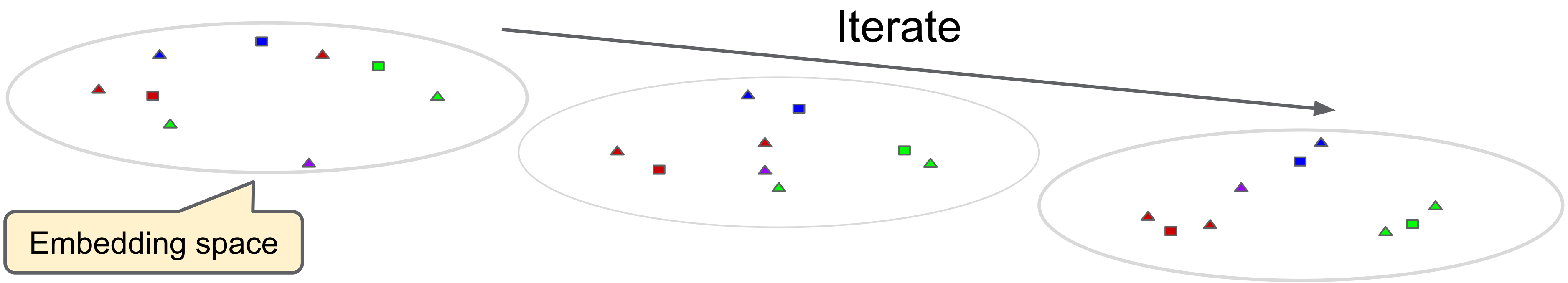
- “Web-scale annotation by input embedding”
- Learns an embedding model

Modeling the data

- Each sample X is a sparse N-dimensional vector (N ≈ millions)
- Each label is an integer in [1, k] (k ≈ thousands)
- Wsabie learns a projection into a low-dimensional embedding space
    - Makes the problem computationally feasible
    - Provides meaningful metric inside embedding space



Learning the model



Enforce constraint:

$$\mathrm{Sim}(x, y+) > \mathrm{Sim}(x, y-) + \text{margin}$$

Use gradient descent to minimize loss function:

$$\text{Loss} = |\text{margin} - \mathrm{Sim}(x, y+) + \mathrm{Sim}(x, y-)|$$

Normalized dot product is a meaningful similarity function

Using the model

Once the projection matrices are learned, we can do useful things

- Compare two samples? Project into embedding space, measure distance
- Closest family to a sample? Project sample and all families, find smallest distance
- Approximate nearest sample? Filter samples by closest family

- Background
- Detection
- Analysis
- Discussion

Summary

Many applications of data analysis to security

Without an automated signal pipeline, analysts run blind, but

- Building a pipeline with high S/N ratio is hard
- Unknown unknowns remain a concern

Interactive analysis tools are just as important as a signal pipeline

- Security monitoring as a search problem

Thank you!