Building a provenance-based intrusion detection system

Thomas Pasquier, University of Bristol
AMSS, 08/12/2020

Talk loosely based on following publications

- Han et al. **"SIGL: Securing Software Installations Through Deep Graph Learning",** USENIX Security Symposium 2021
- Han et al. **"UNICORN: Revisiting Host-Based Intrusion Detection in the Age of Data Provenance"**, NDSS 2020
- Pasquier et al. **"Runtime Analysis of Whole-System Provenance"**, ACM CCS 2018
- Pasquier et al. **"Practical Whole-System Provenance Capture"**, ACM SoCC 2017

https://tfjmp.org

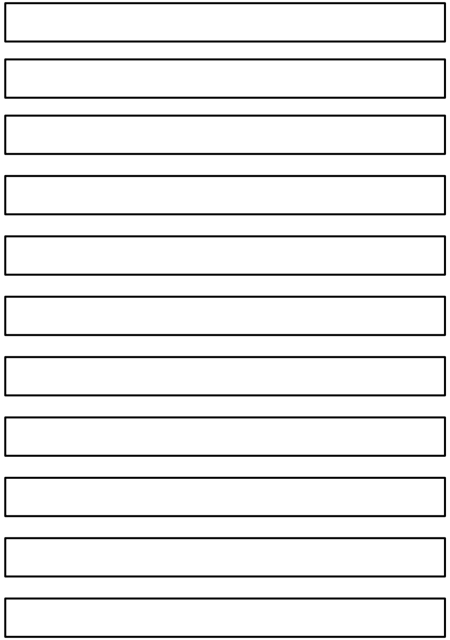
Plan: Building a provenance-based intrusion detection system

- Motivation
- What is provenance?
- How is useful?
- How to perform detection?
- How to evaluate
- Some insights from this work

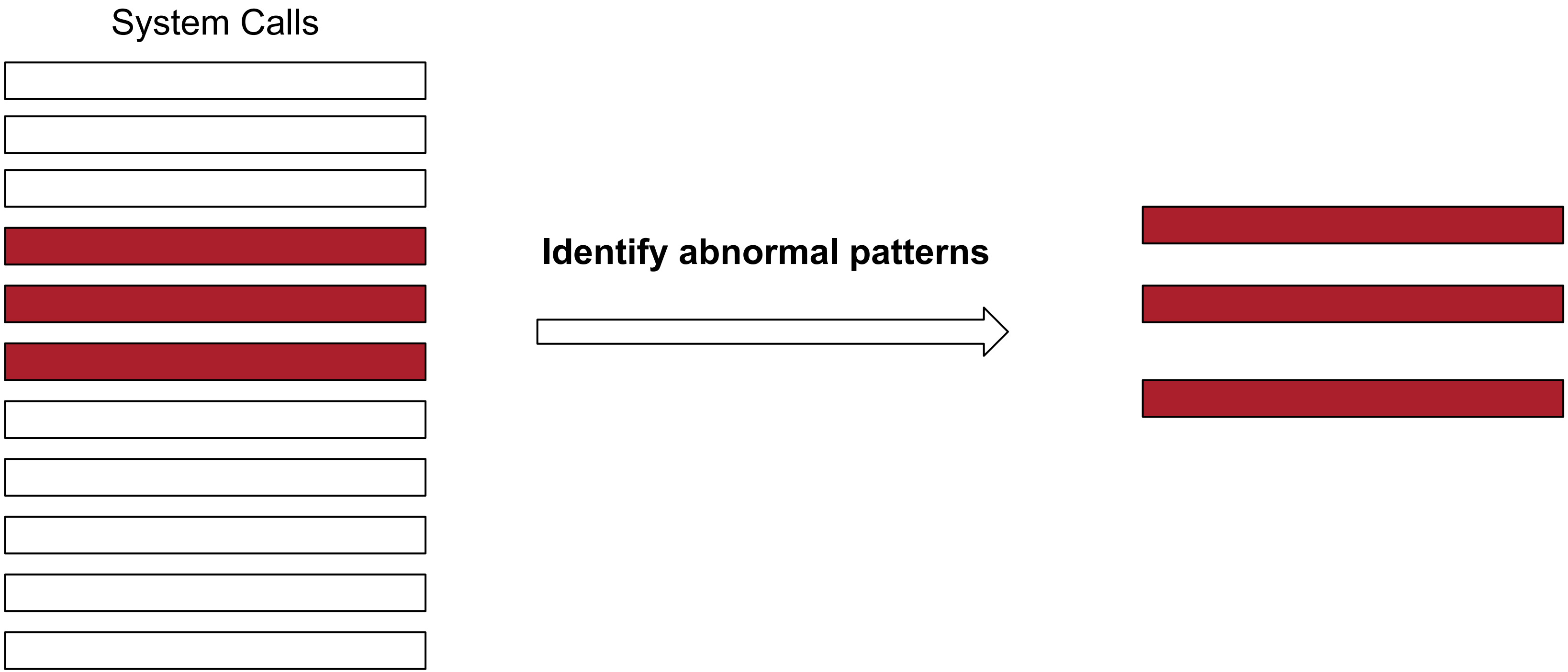
Motivation

Motivation: System call based intrusion detection

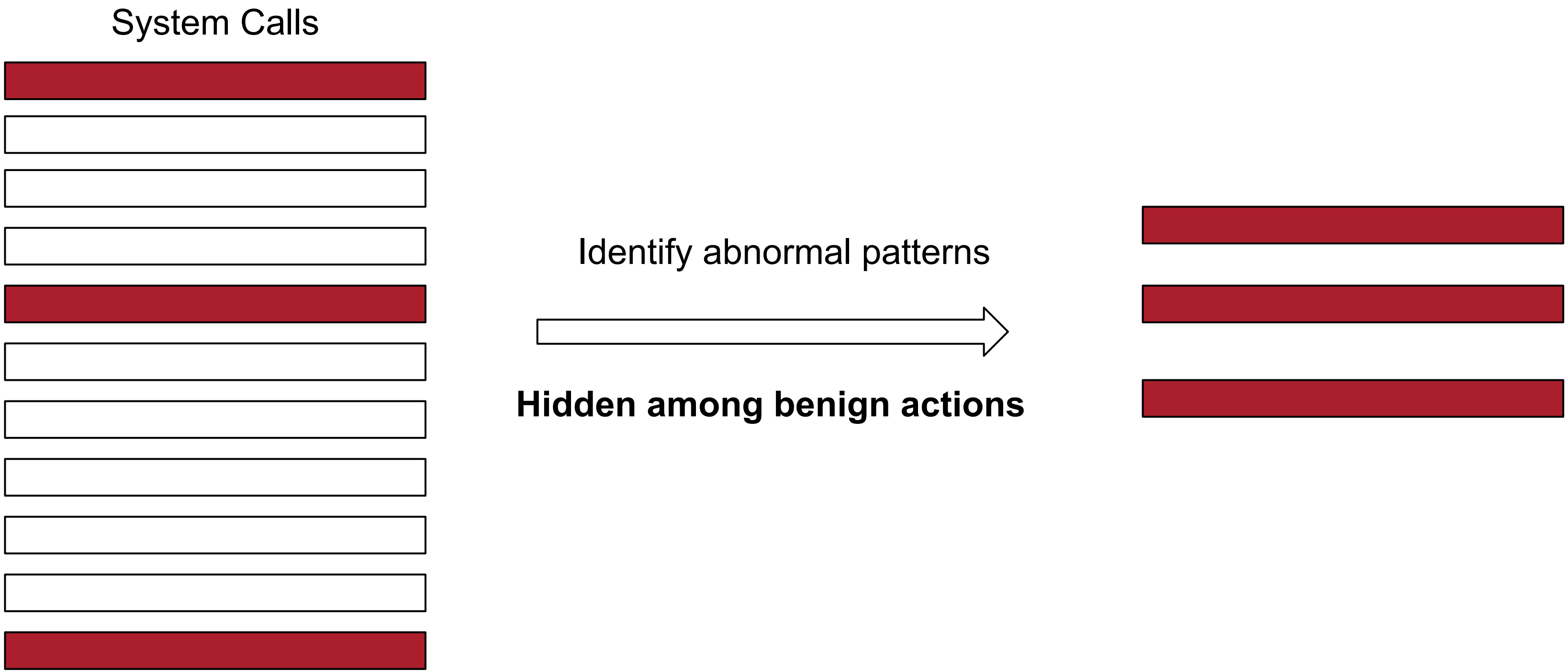
System Calls



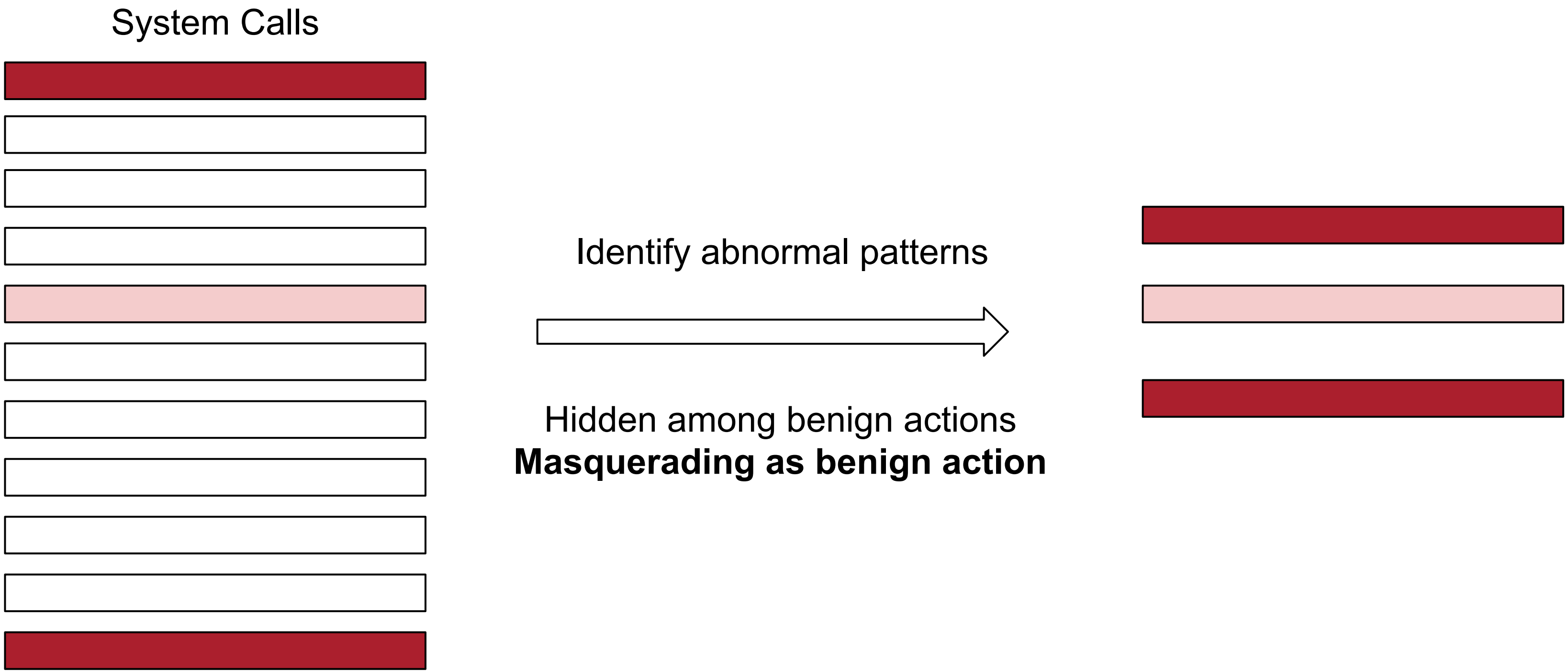
Motivation: System call based intrusion detection



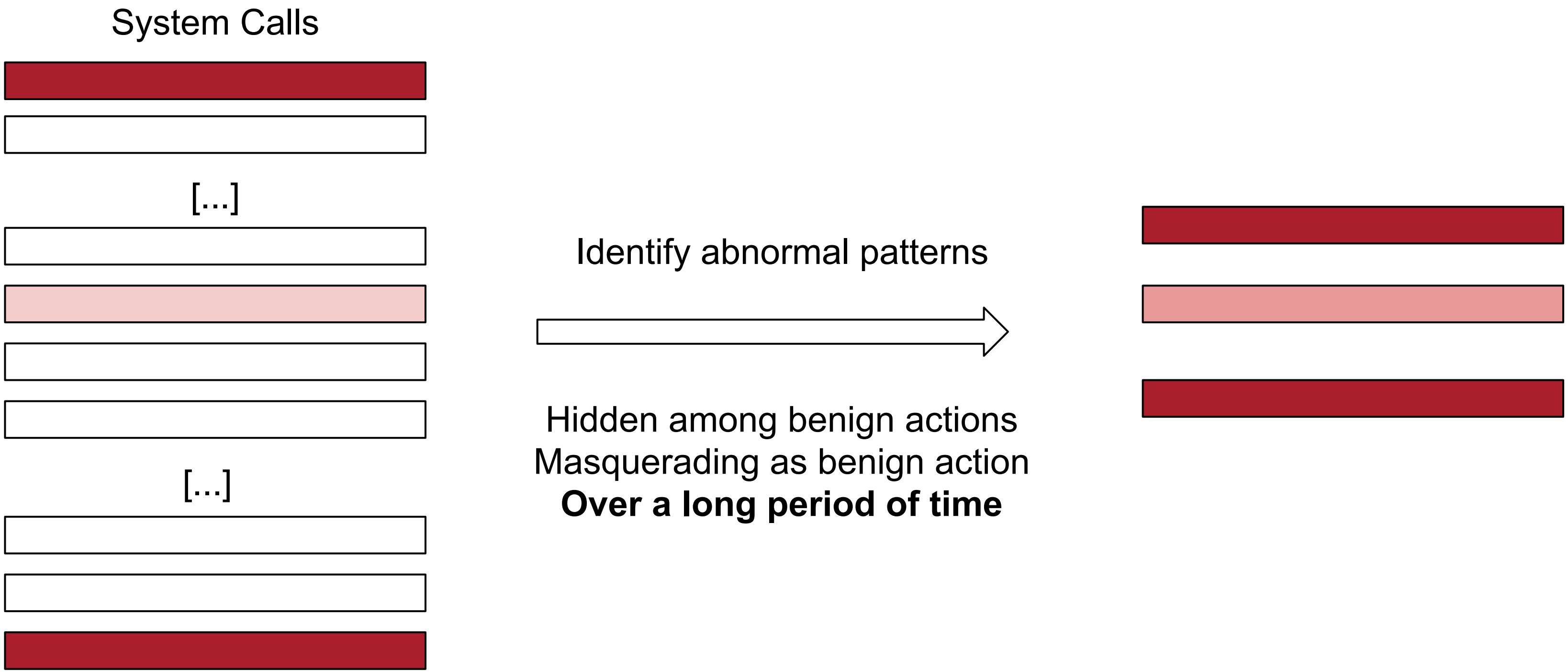
Motivation: System call based intrusion detection



Motivation: System call based intrusion detection



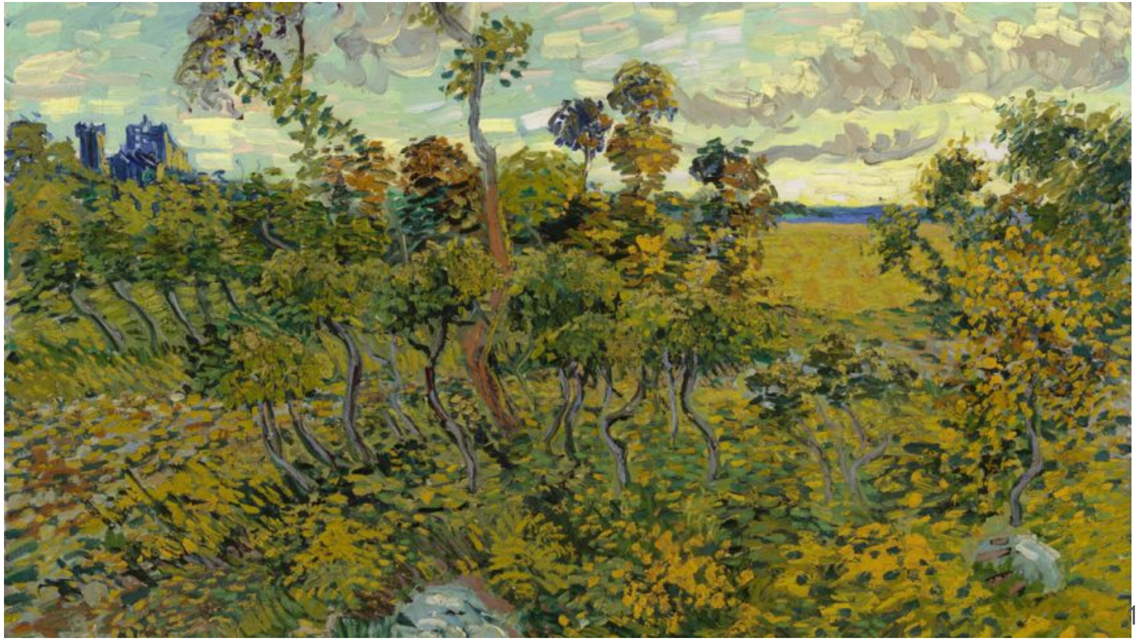
Motivation: System call based intrusion detection



What is provenance?

What is provenance?

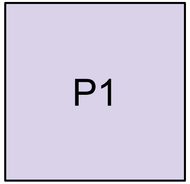
- From the French “provenir” meaning “coming from”
- **Formal set of documents** describing the origin of an art piece
- **Sequence** of
  - Formal ownership
  - Custody
  - Places of storage
- Used for authentication



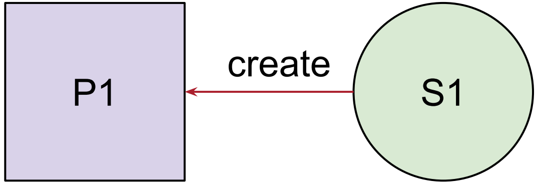
What is data-provenance?

- Represent interactions between objects of different types
  - Data-items (**entities**)
  - Processing (**activities**)
  - Individuals and Organisations (**agents**)
- Represented as a **directed acyclic graph** (think information flows)
- Edges represent interactions between objects as dependencies
- It is a representation of **history**
  - Immutable (unless it’s 1984)
  - No dependency to the future

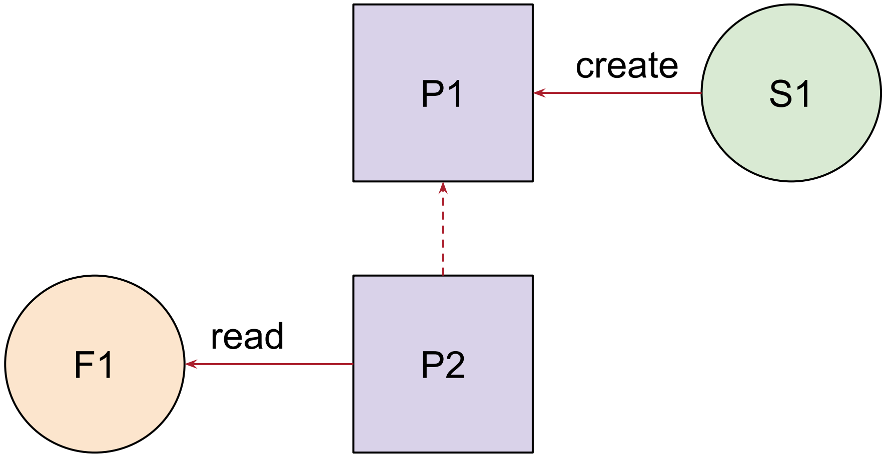
Example provenance (simplified)



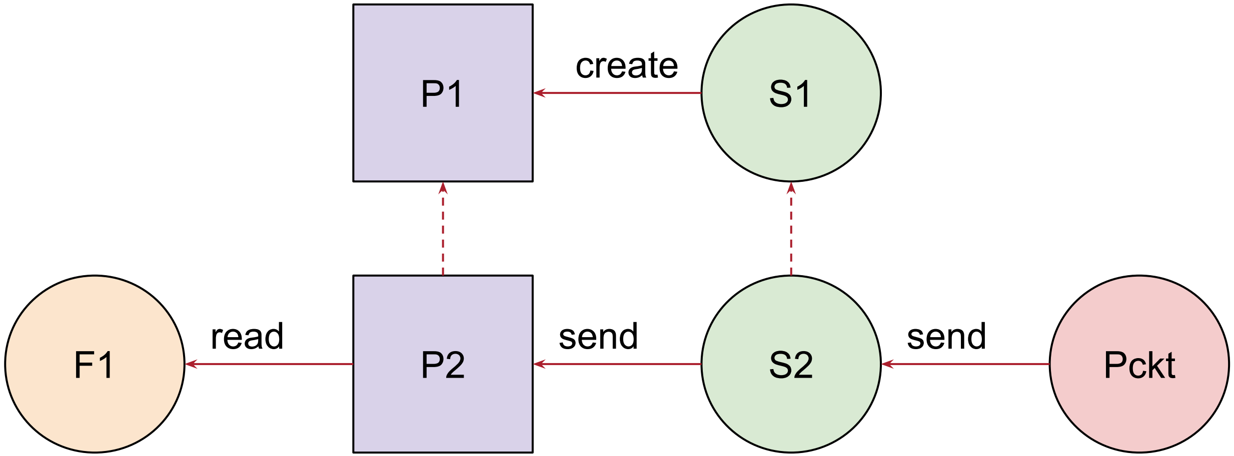
Example provenance (simplified)



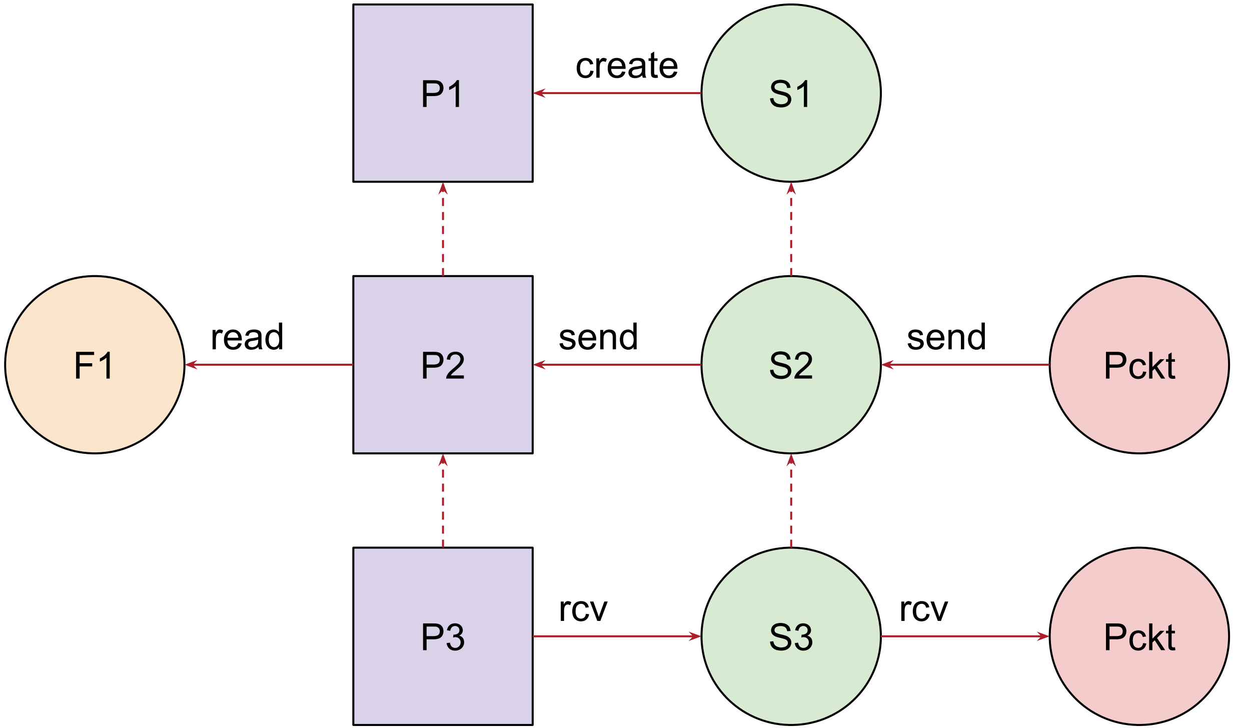
Example provenance (simplified)



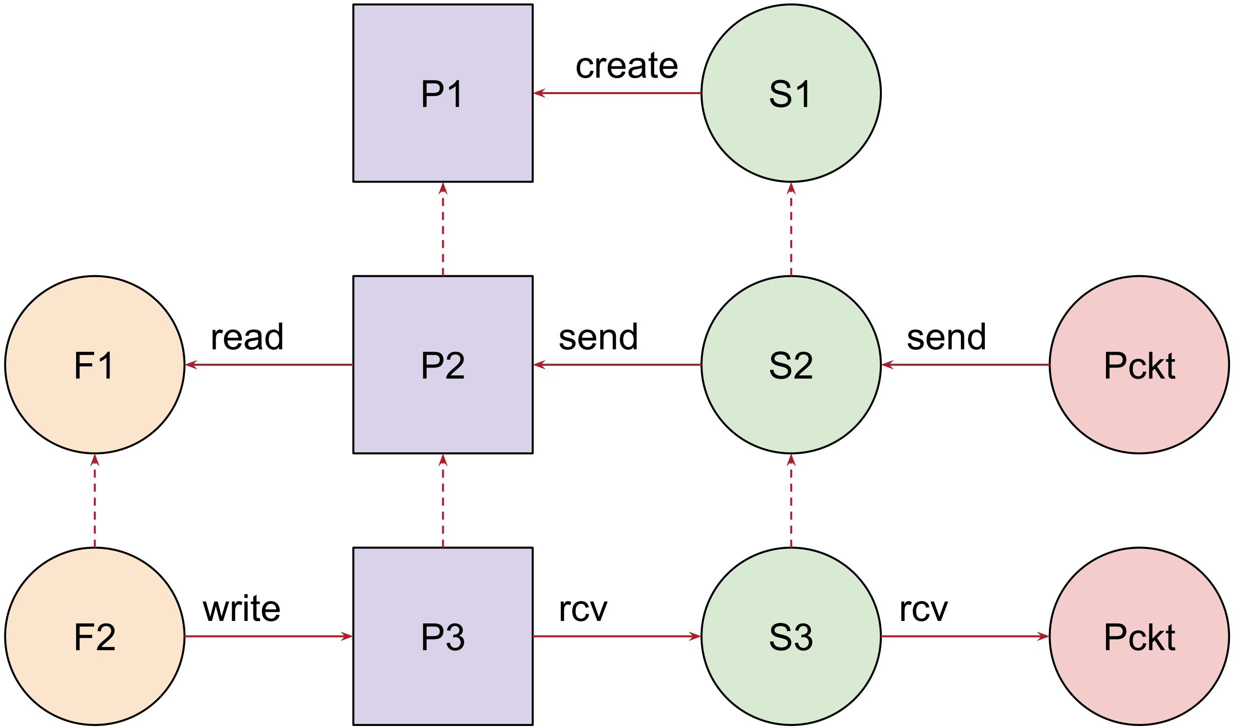
Example provenance (simplified)



Example provenance (simplified)

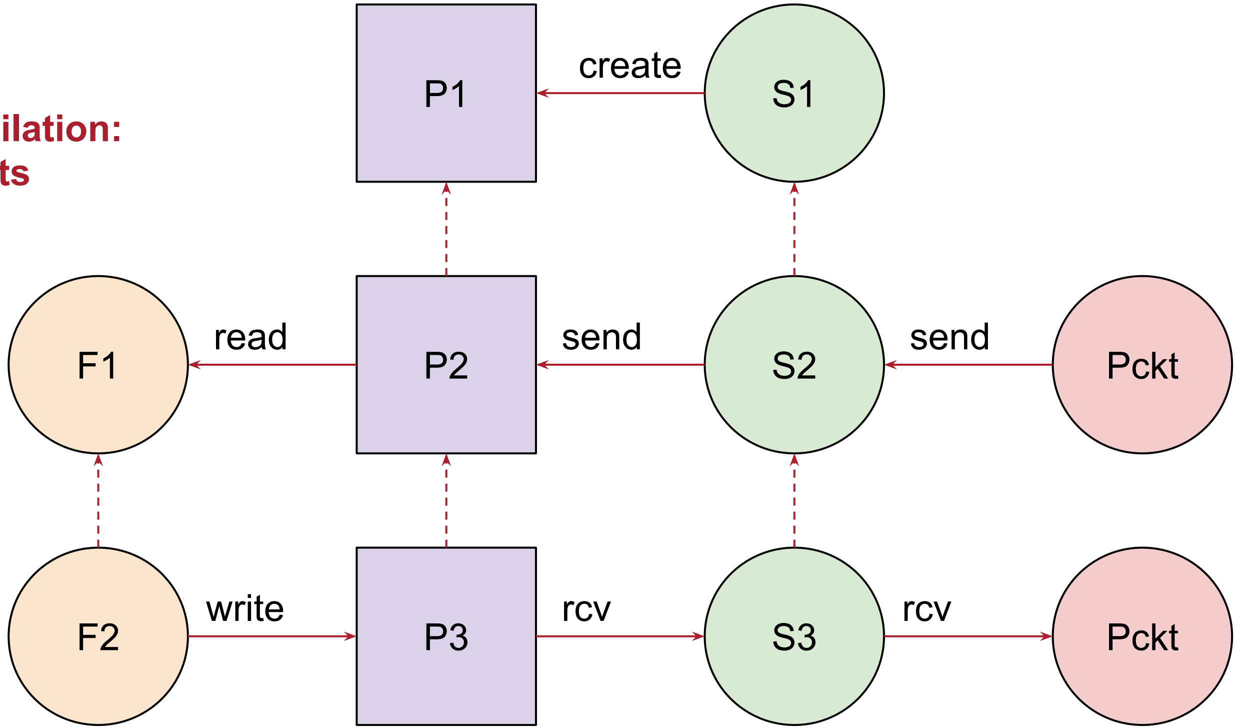


Example provenance (simplified)



Example provenance (simplified)

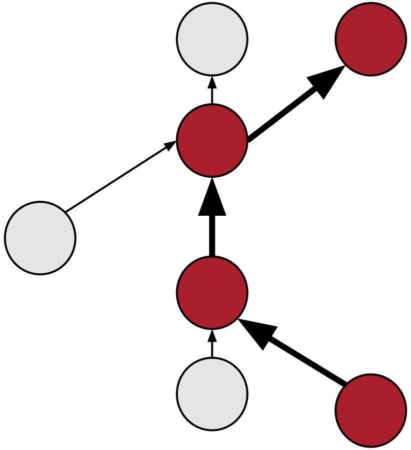
Linux kernel compilation:
~2M graph elements



How is this useful?

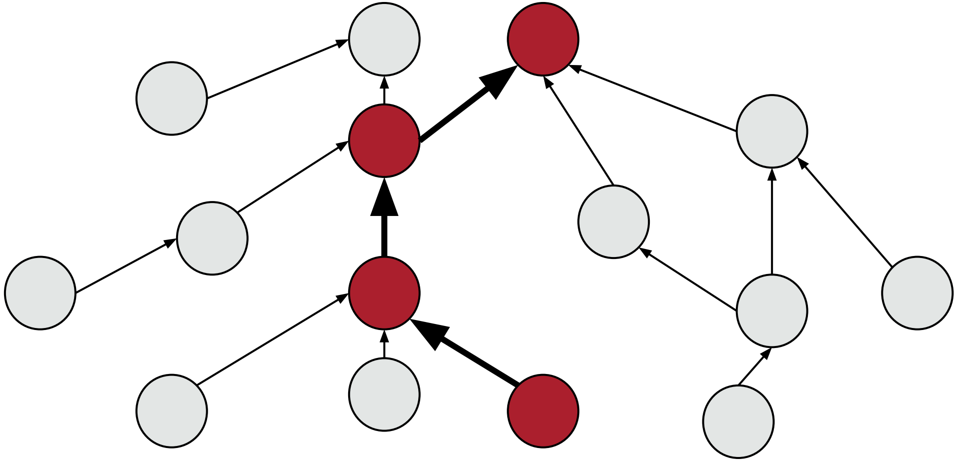
Provenance-based intrusion detection

- **Intuition**: provenance graph **exposes causality relationships** between events



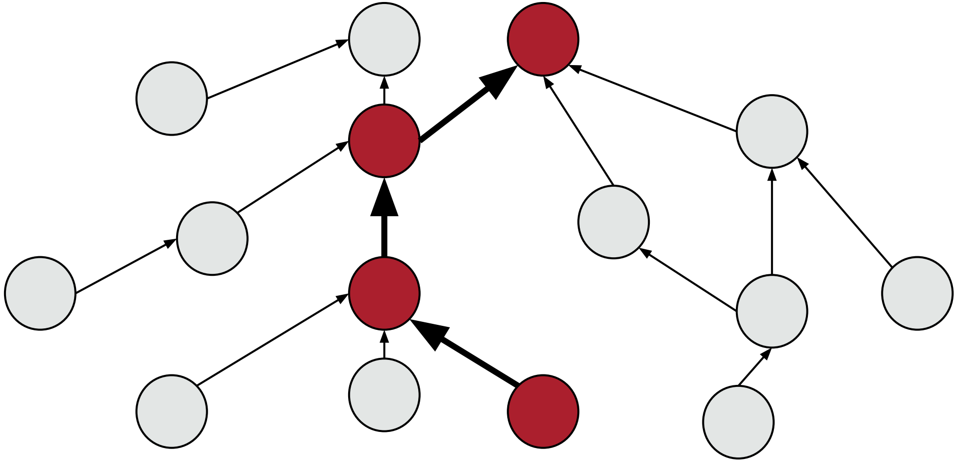
Provenance-based intrusion detection

- **Intuition**: provenance graph **exposes causality relationships** between events



Provenance-based intrusion detection

- Related events are connected even across long period of time

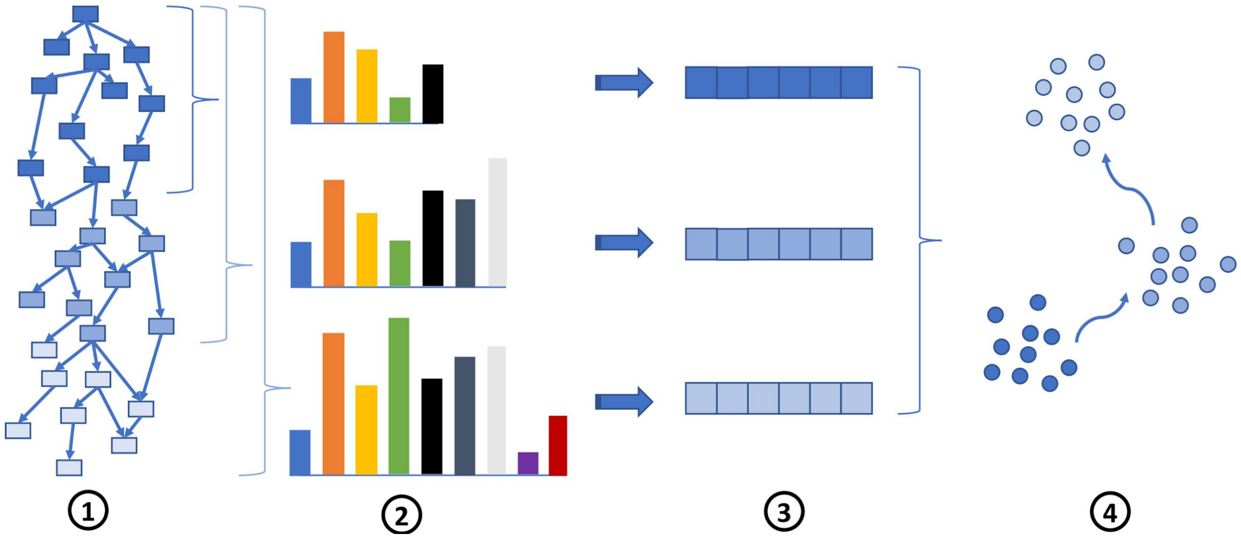


How to perform detection?

Assumptions (and limitations)

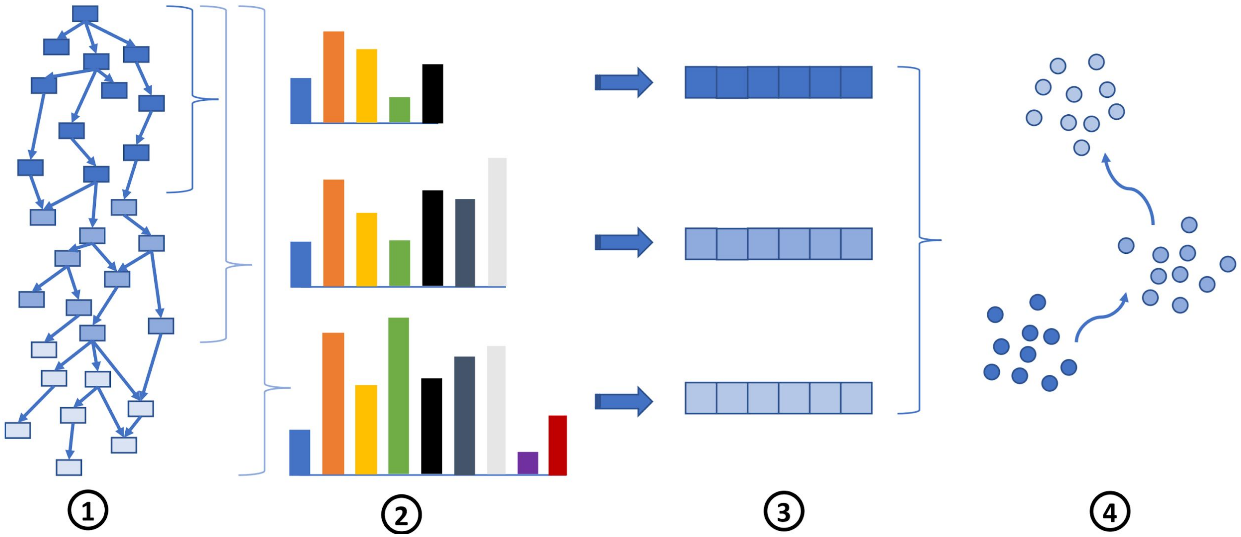
- **Runtime detection**
- We target environment with **minimal human intervention**
  - relatively consistent behaviour
  - e.g. web servers, CI pipelines etc...
- Build a **model of system behaviour** (unsupervised training)
  - in a controlled environment
  - from a representative workload (this is hard!)
- **Detect deviation** from the model
- Several approaches being explored…

Example: UNICORN



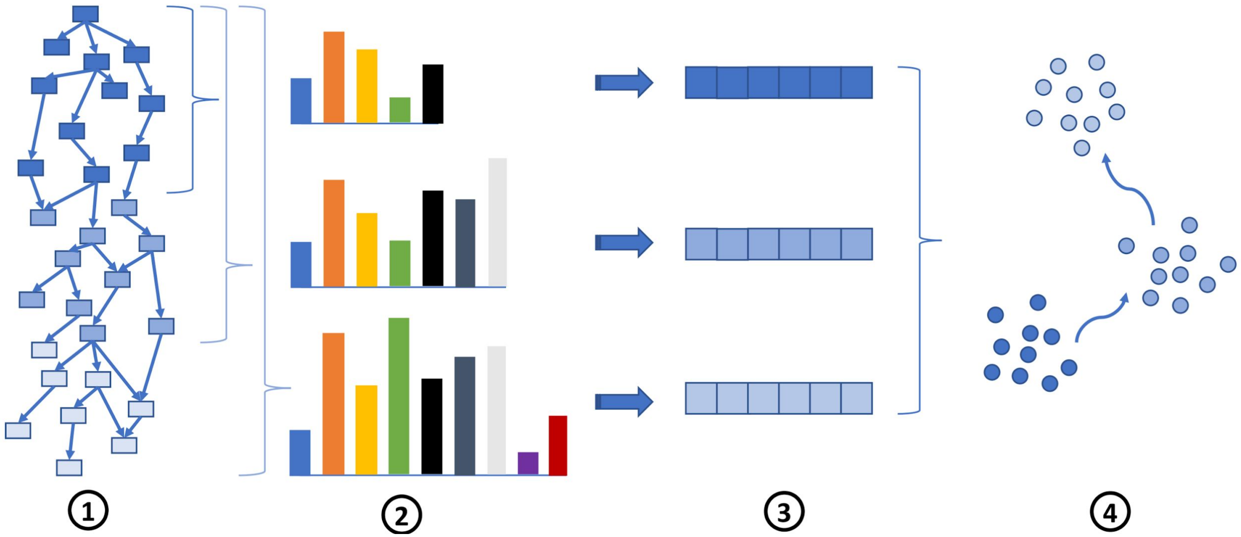
- Han et al. **“UNICORN: Runtime Provenance-Based Detector for Advanced Persistent Threats”**, NDSS 2020

Example: UNICORN



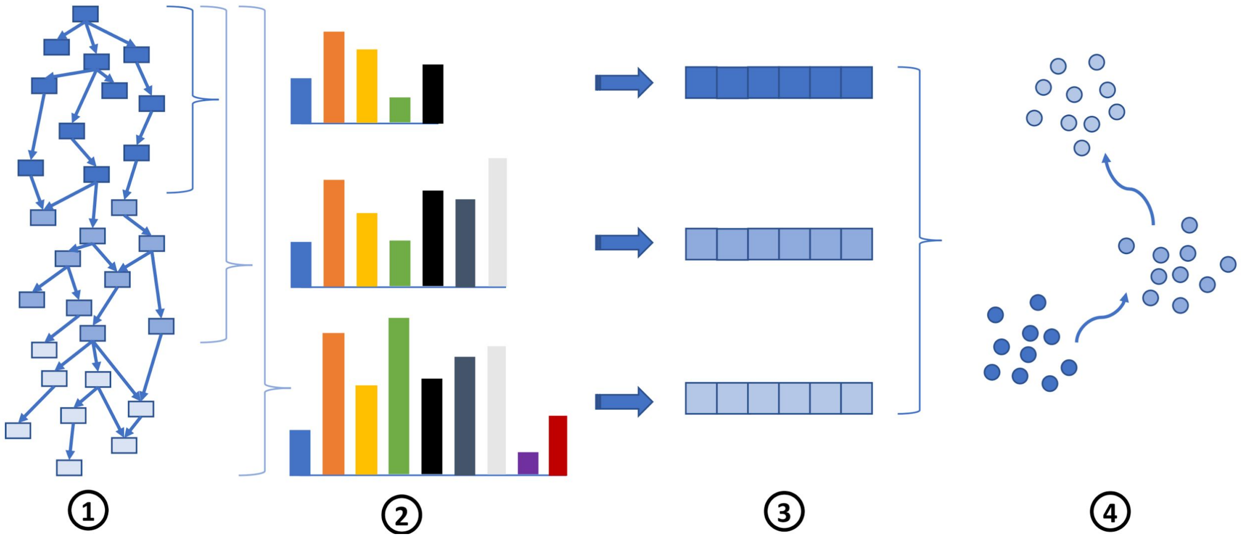
1) Graph streamed in, converted to histogram, labelled using (modified) **struct2vec**

Example: UNICORN



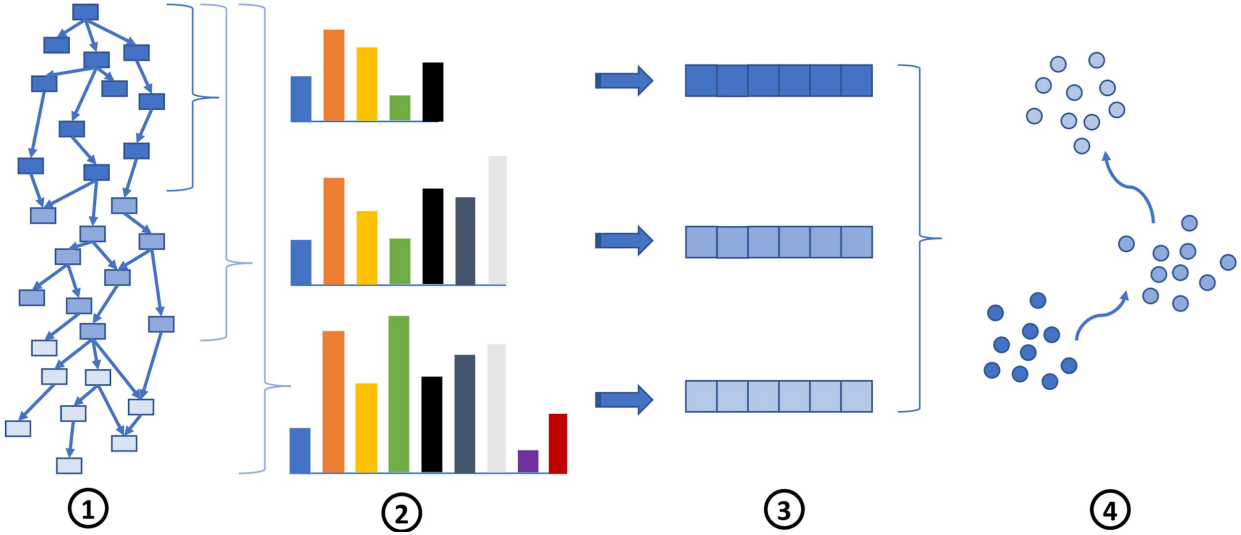
2) At regular interval, histogram converted to a fixed size vector using **similarity preserving graph sketching**

Example: UNICORN



3) Feature vectors are **clustered**

Example: UNICORN



4) Cluster forms “**meta-state**”, transitions are modelled

In deployment, anomaly detected via clustering and “meta-state” model

Relatively simple

- Labelled directed acyclic graph
  - node/edge types
  - security context (when available)
- Modification and combination of existing algorithms
  - struct2vec
  - similarity preserving hashing
  - clustering
- Right combination + domain knowledge

How to evaluate?

Evaluation with DARPA datasets

Experiment	Dataset	# of Graphs	Avg. $|V|$	Avg. $|E|$	Raw Data Size (GiB)
DARPA	Benign	66	59,983	4,811,836	271
CADETS	Attack	8	386,548	5,160,963	38
DARPA	Benign	43	2,309	4,199,309	441
ClearScope	Attack	51	11,769	4,273,003	432
DARPA	Benign	2	19,461	1,913,202	4
THEIA	Attack	25	275,822	4,073,621	85

TABLE IV: Characteristics of graph datasets used in the DARPA experiments.

Experiment	Precision	Recall	Accuracy	F-Score
DARPA CADETS	0.98	1.0	0.99	0.99
DARPA ClearScope	0.98	1.0	0.98	0.99
DARPA THEIA	1.0	1.0	1.0	1.0

TABLE V: Experimental results of the DARPA datasets.

Evaluation with DARPA datasets

Experiment	Dataset	# of Graphs	Avg. |V|	Avg. |E|	Raw Data Size (GiB)
DARPA	Benign	66	59,983	4,811,836	271
CADETS	Attack	8	386,548	5,160,963	38
DARPA	Benign	43	2,309	4,199,309	441
ClearScope	Attack	51	11,769	4,273,003	432
DARPA	Benign	2	19,461	1,913,202	4
THEIA	Attack	25	275,822	4,073,621	85

TABLE IV: Characteristics of graph datasets used in the DARPA experiments.

Experiment	Precision	Recall	Accuracy	F-Score
DARPA CADETS	0.98	1.0	0.99	0.99
DARPA ClearScope	0.98	1.0	0.98	0.99
DARPA THEIA	1.0	1.0	1.0	1.0

TABLE V: Experimental results of the DARPA datasets.

SUCH GOOD RESULTS ARE NOT NORMAL

Building our own dataset

Experiment	Dataset	# of Graphs	Avg. |V|	Avg. |E|	Raw Data Size (GiB)
SC-1	Benign	125	265,424	975,226	64
	Attack	25	257,156	957,968	12
SC-2	Benign	125	238,338	911,153	59
	Attack	25	243,658	949,887	12

TABLE VI: Characteristics of the datasets used in the supply-chain APT attack experiments.

Experiment	Precision	Recall	Accuracy	F-Score
SC-1	0.85	0.96	0.90	0.90
SC-2	0.75	0.80	0.77	0.78

TABLE VIII: Experimental results of the supply-chain APT attack scenarios.

- Attack designed to look similar to background activity

Building our own dataset

Experiment	Dataset	# of Graphs	Avg. |V|	Avg. |E|	Raw Data Size (GiB)
SC-1	Benign	125	265,424	975,226	64
	Attack	25	257,156	957,968	12
SC-2	Benign	125	238,338	911,153	59
	Attack	25	243,658	949,887	12

TABLE VI: Characteristics of the datasets used in the supply-chain APT attack experiments.

Experiment	Precision	Recall	Accuracy	F-Score
SC-1	0.85	0.96	0.90	0.90
SC-2	0.75	0.80	0.77	0.78

TABLE VIII: Experimental results of the supply-chain APT attack scenarios.

- Attack designed to look similar to background activity
- Is that enough?

Some insights from this work

We can build practical provenance-based IDSs

- We can detect intrusion out of graph structure with little metadata
  - Vertex type (thread, file, socket etc…)
  - Edge type (read, write, connect etc…)
- Processing speed
  - Current prototype
  - Data generation speed < processing speed!

Proper evaluation is hard!

- Dataset are hard to generate
    - What is a good quality dataset?
- Hard to compare across papers, a lot is not available
    - Experiments (i.e. attacks)
    - Capture Mechanisms
    - Analysis pipelines
- Leads to unsatisfactory evaluation
    - I may be able to compare to similar techniques (may reuse dataset)
    - … very hard for unrelated one (i.e. ingest different data type)
- Adversarial ML?

Identifying threats: explainability is a problem

- There is a problem within the last batch of X graph elements
  - 2,000 in previous figures
- Good luck finding out what went wrong
- Provenance forensic is an active field of research
  - Promising work out of the DARPA programme
- … but could we do better during detection?
- Han et al. **“SIGL: Securing Software Installations Through Deep Graph Learning”,** USENIX Security Symposium 2021
  - Graph LSTM Autoencoder
  - Pinpoint issue in the graph, but...
  - ...less complex problem
  - ...smaller graphs
  - Can we take such approach further?

Looking for ...

- PhD Students
- Postdocs: job advert to come (soon) on intrusion detection at the edge
- Collaborators

Get in touch: **https://tfjmp.org**

Thank you! Questions?

tfjmp.org
camflow.org

Comparison state of the art

Experiment	Dataset	# of Graphs	Avg. $\|V\|$	Avg. $\|E\|$	Preprocessed Data Size (GiB)
StreamSpot	YouTube	100	8,292	113,229	0.3
	Gmail	100	6,827	37,382	0.1
	Download	100	8,831	310,814	1
	VGame	100	8,637	112,958	0.4
	CNN	100	8,990	294,903	0.9
	Attack	100	8,891	28,423	0.1

TABLE I: Characteristics of the StreamSpot dataset. The dataset is publicly available only in a preprocessed format.

Experiment	Precision	Recall	Accuracy	F-Score
StreamSpot (baseline)	0.74	N/A	0.66	N/A
$R = 1$	0.51	1.0	0.60	0.68
$R = 3$	0.98	0.93	0.96	0.94

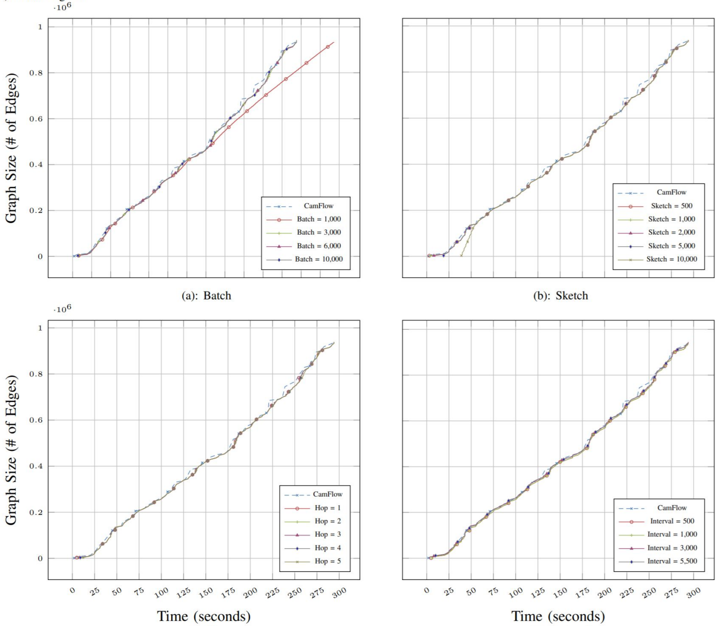
TABLE II: Comparison to StreamSpot on the StreamSpot dataset. We estimate StreamSpot's average accuracy and precision from the figure included in the paper [83], which does not report exact values. They did not report recall or F-score.

Manzoor et al. "**Fast memory-efficient anomaly detection in streaming heterogeneous graphs**"
ACM KDD, 2016.

R -> neighborhood size for struct2vec algorithm

Runtime performance

Fig. 4: Total number of processed edges over time (in seconds) in the SC-1 experimental workload with varying batch sizes (Fig. 4(a)), sketch sizes (Fig. 4(b)), hop counts (Fig. 4(c)), and intervals of sketch generation (Fig. 4(d)). Dashed blue line represents the speed of graph edges streamed into UNICORN for analysis. Triangle maroon baseline has the same configurations as those used in our experiments and indicates the values of the controlled parameters (that remain constant) in each figure.

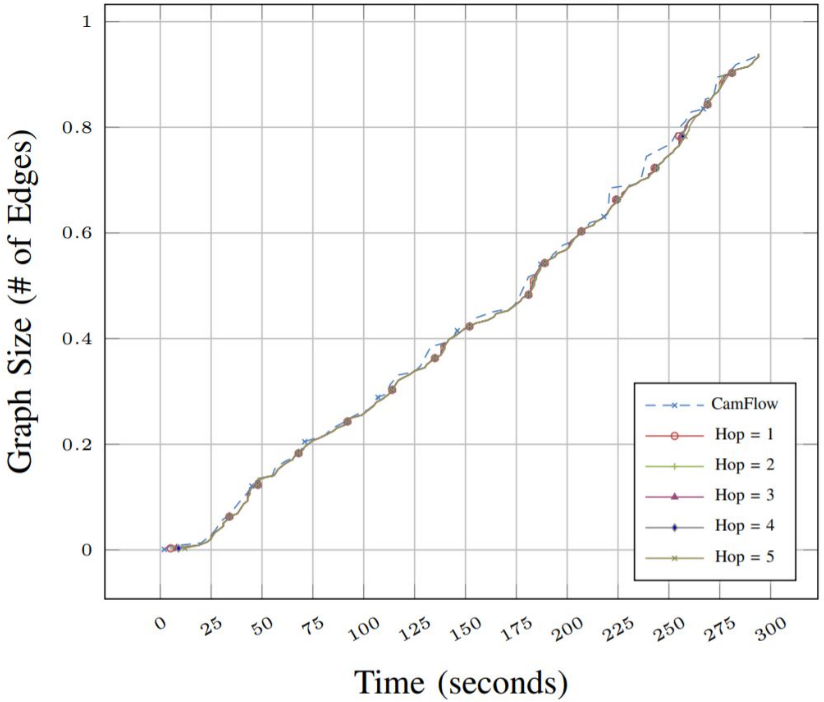


(a): Batch (b): Sketch (c): Hop (d): Interval

F. CPU & Memory Utilization

Configuration Parameter	Parameter Value	Max Memory Usage (MB)
	R = 1	562

Runtime performance

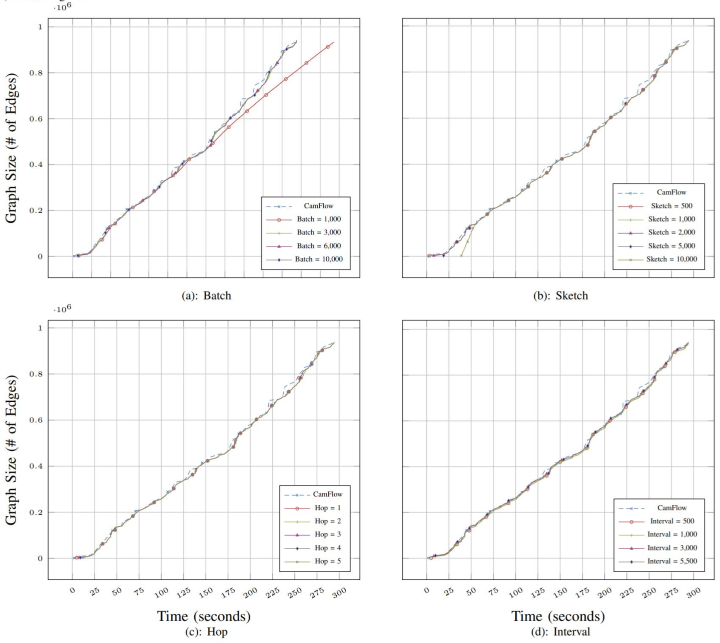


(c): Hop

Runtime performance

Memory usage: ~500MB
CPU usage 15% on 1 core

Fig. 4: Total number of processed edges over time (in seconds) in the SC-1 experimental workload with varying batch sizes (Fig. 4(a)), sketch sizes (Fig. 4(b)), hop counts (Fig. 4(c)), and intervals of sketch generation (Fig. 4(d)). Dashed blue line represents the speed of graph edges streamed into UNICORN for analysis. Triangle maroon baseline has the same configurations as those used in our experiments and indicates the values of the controlled parameters (that remain constant) in each figure.

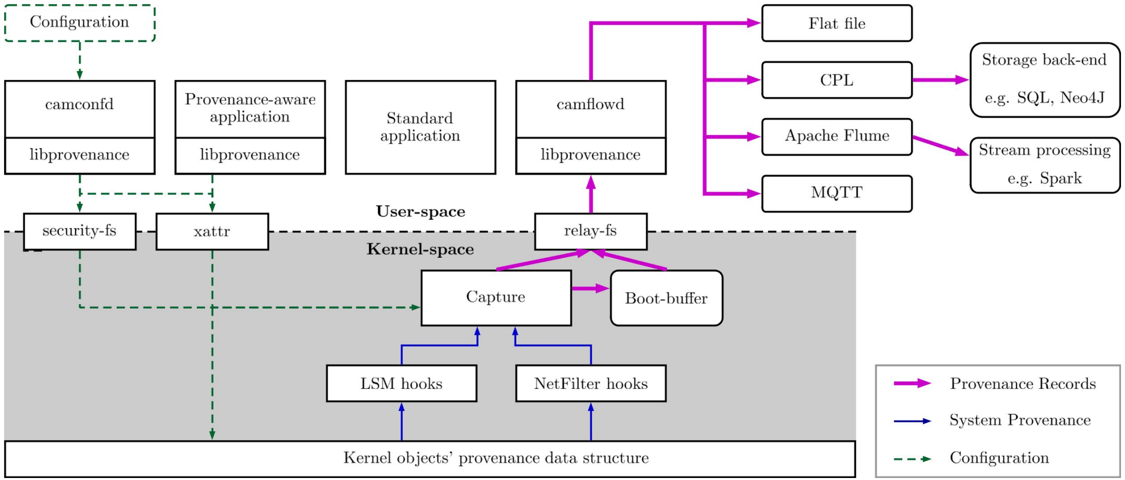


F. CPU & Memory Utilization

Configuration Parameter	Parameter Value	Max Memory Usage (MB)
	R = 1	562

CamFlow capture mechanism

- Leverage existing kernel features whenever possible
- Avoid alteration of existing code
- We therefore build upon:
    - **Linux Security Module**
    - to capture system events
    - **NetFilter**
    - to capture network events
    - **RelayFS**
    - to transfer provenance to user space
    - **SecurityFS**
    - to provide a userspace interface for settings



Extent of modification

Modifications to the Linux Kernel code

System	Headers	C File	Total	LoC
PASS (v2.6.27) pub. 2006	18	69	87	5100
LPM (v2.6.32) pub. 2015	13	61	74	2294
CamFlow (v5.4.15) circa 2020	3	0	3	4220

Capture overhead

Micro-benchmark

Sys Call	Whole	Selective
stat	100%	28%
open/close	80%	18%
fork	6%	2%
exec	3%	<1%

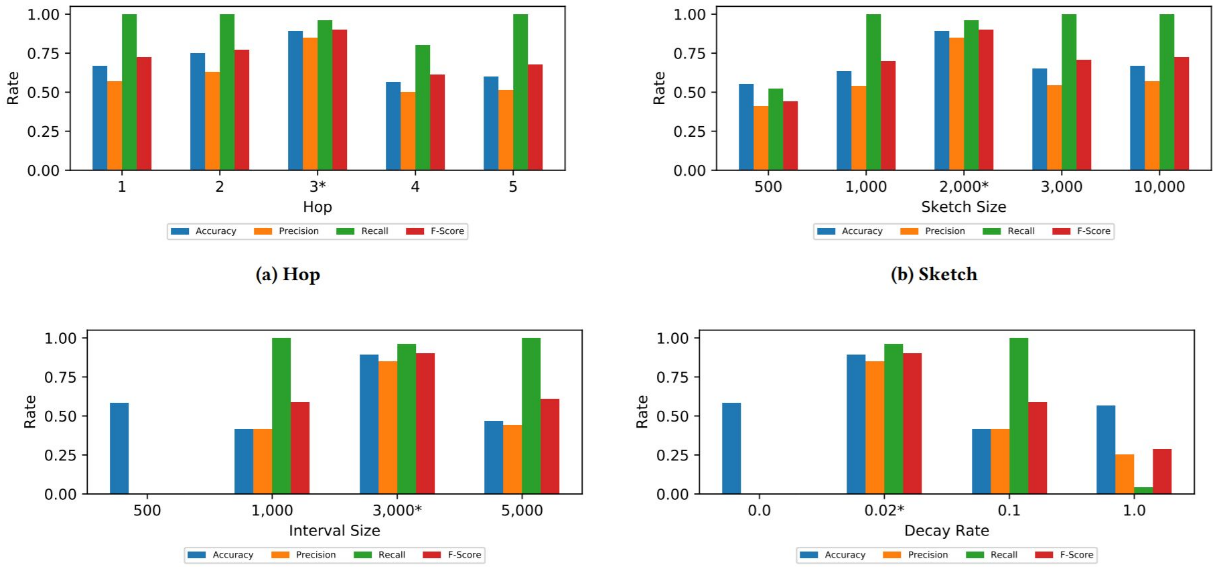
Macro-benchmark

Prog.	Whole	Selective
unpack	2%	<1%
build	2%	0%
postmark	11%	6%

Selective: cost of allocating/freeing provenance “blob” + recording or not decision

Whole: **Selective** + cost of recording provenance information

IDS performance (more)



(a) Hop

(b) Sketch

(c) Interval

(d) Decay

Figure 4: Detection performance (precision, recall, accuracy, and F-score) with varying hop counts (Fig. 4a), sketch sizes (Fig. 4b), intervals of sketch generation (Fig. 4c), and decay factor (Fig. 4d). Baseline values (*) are used by the controlled parameters (that remain constant) in each figure.

IDS performance (more)

Configuration Parameter	Parameter Value	Max Memory Usage (MB)
Hop Count	R = 1	562
	R = 2	624
	R = 3	687
	R = 4	749
	R = 5	812
Sketch Size	|S| = 500	312
	|S| = 1,000	437
	|S| = 2,000	687
	|S| = 5,000	1,374
	|S| = 10,000	2,498

Table 5: Memory usage with varying hop counts and sketch sizes.

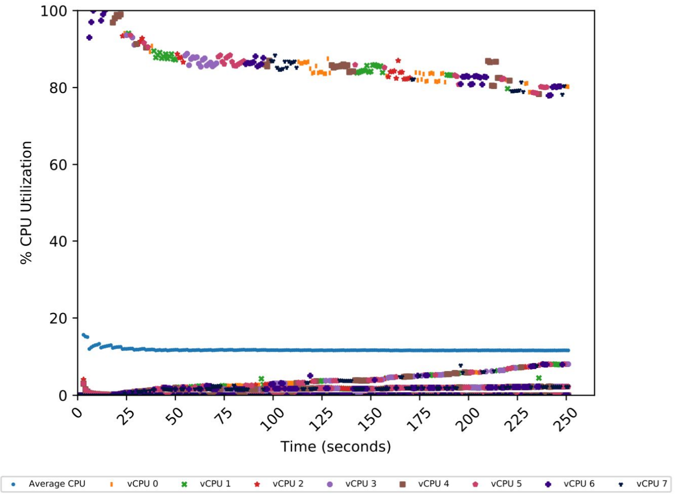


Figure 6: Per virtual CPU and average CPU utilization.

IDS performance (more)

CPU over long time period? 15% CPU time across cores

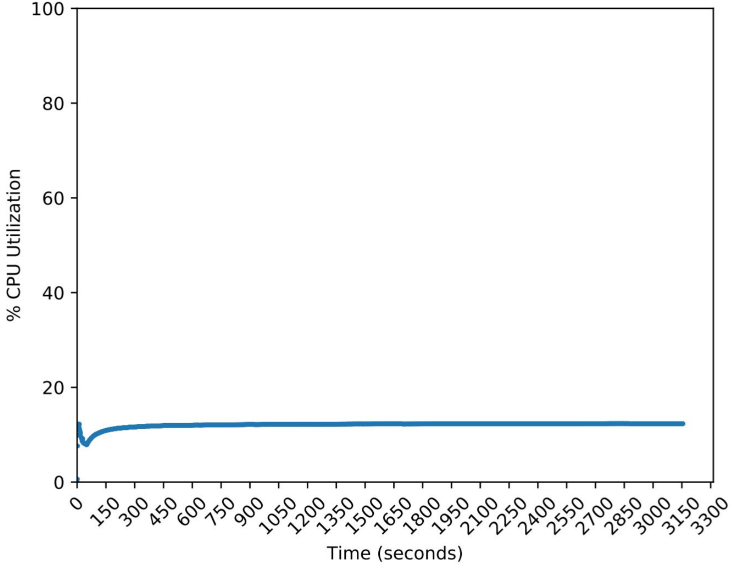


Figure 5: Average CPU utilization with the baseline configurations.

Advanced Persistent Threats