Incorporating Gradients to Rules: Towards Lightweight, Adaptive Provenance-based Intrusion Detection

Lingzhi Wang, Xiangmin Shen, Weijian Li , Zhenyuan Li ,
R. Sekar , Han Liu , and Yan Chen
Northwestern University, Stony Brook University, Zhejiang University

NORTHWESTERN
UNIVERSITY

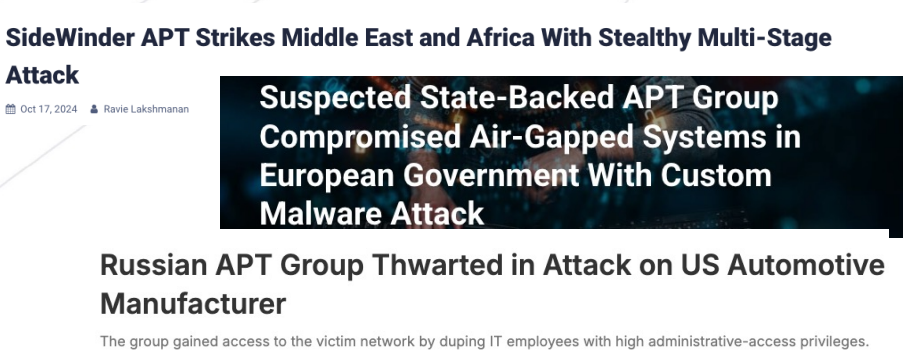
Stony Brook
University

浙江大學

Advanced Persistent Threats (APT)

APT attacks bring challenges to traditional intrusion detection systems.

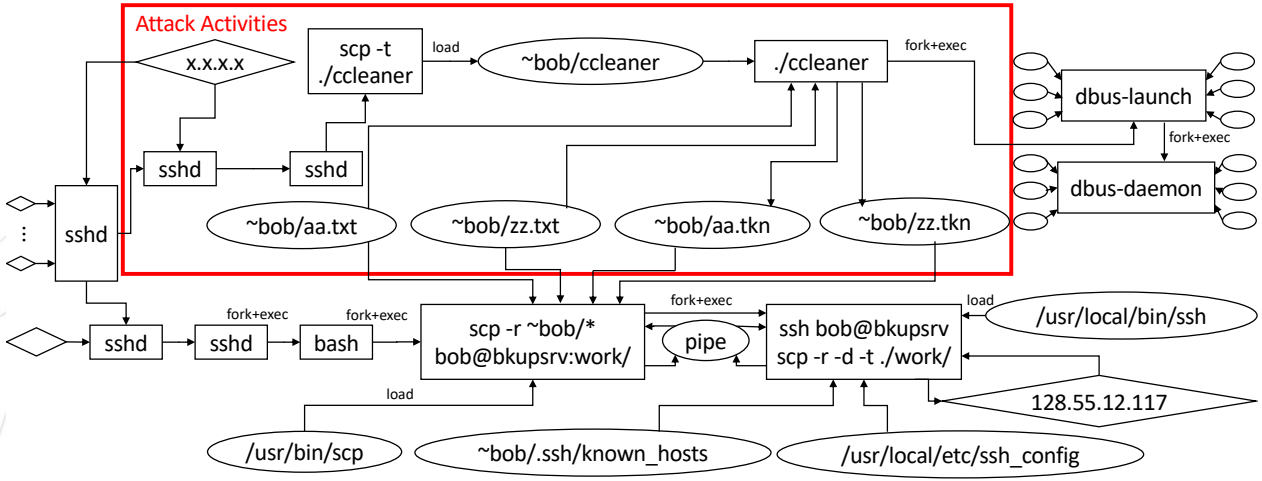
- Advanced: Use of sophisticated techniques (e.g., zero-day exploits, custom malware).
- Persistent: Long-term access to networks, often undetected for several days/months
- Targeted: Focused on specific organizations or individuals



Provenance-based Intrusion Detection (PIDS)

Provenance Graph: A graph recording system entities and their interactions

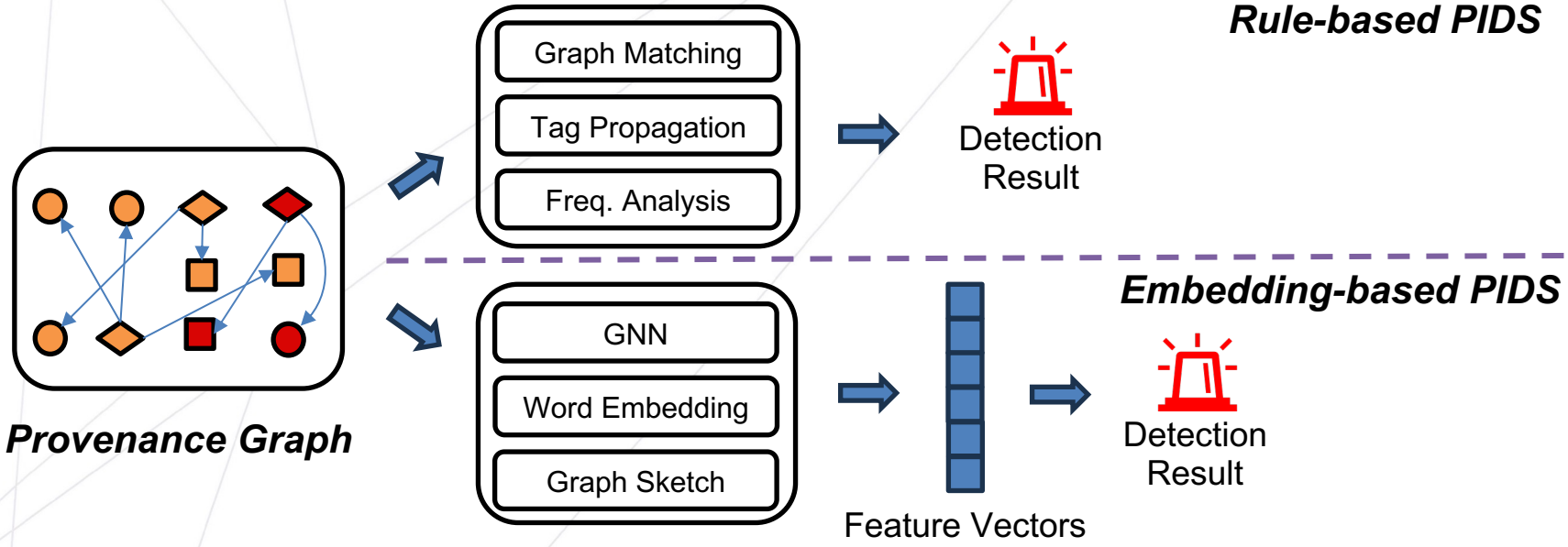
- **Nodes:** System entities (processes, files, sockets, …)
- **Edges:** Interactions between entities (read, write, fork, execve, load, …)
- PIDS provides **more comprehensive contexts, correlations, and scopes** against APT attacks



Rule-based PIDS vs Embedding-based PIDS

Existing PIDS can be (roughly) divided into two categories:

- **Rule-based PIDS:** Use simple, static rules to model graph patterns
- **Embedding-based PIDS:** Use embedding functions to vectorize graph features



Rule-based PIDS vs Embedding-based PIDS

Existing PIDS can be (roughly) divided into two categories:

- **Rule-based PIDS:** Use simple, static rules to model graph patterns
- **Embedding-based PIDS :** Use embedding functions to vectorize graph features

Rule-based PIDS

Fast & Efficient
Lightweight
Explicable Detection Process

Low Accuracy
Rigid & Inflexible
Unadaptable to New Data

Embedding-based PIDS

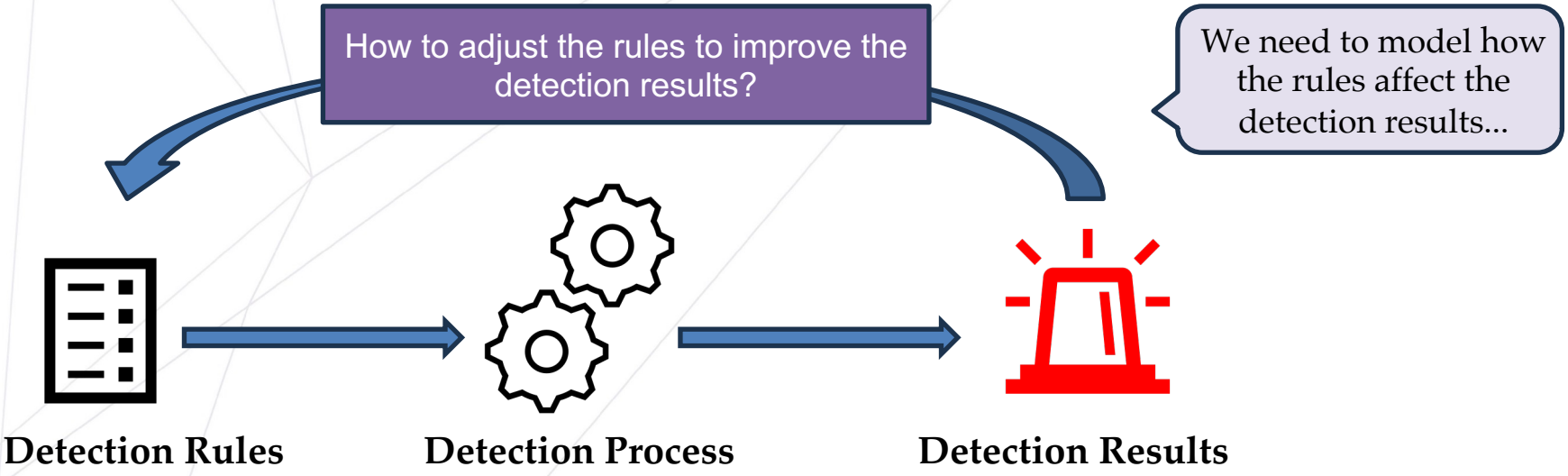
Powerful Feature Extraction
Learning Capability
Adaptable to New Data

High System Overhead
Long Detection Latency
Inexplicable Detection Process

Motivation

Is it possible to maintain the lightweight and efficiency of the rule-based detection while enabling dynamic and automated rule learning?

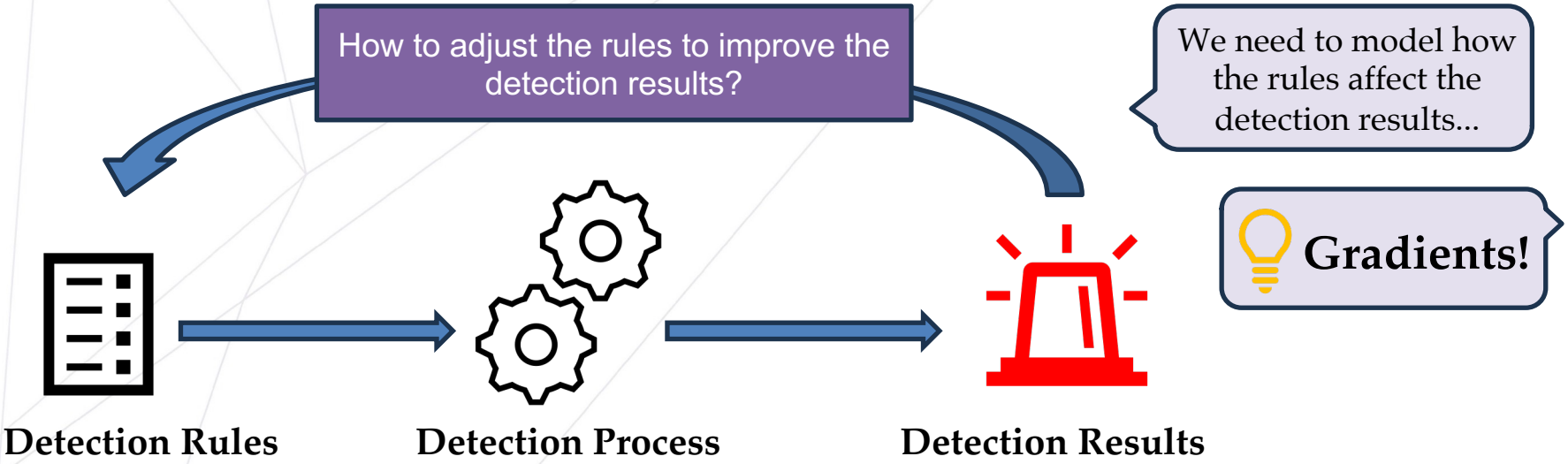
Given a set of rules, how to dynamically adjust them to get a better detection result?



Motivation

Is it possible to maintain the lightweight and efficiency of the rule-based detection while enabling dynamic and automated rule learning?

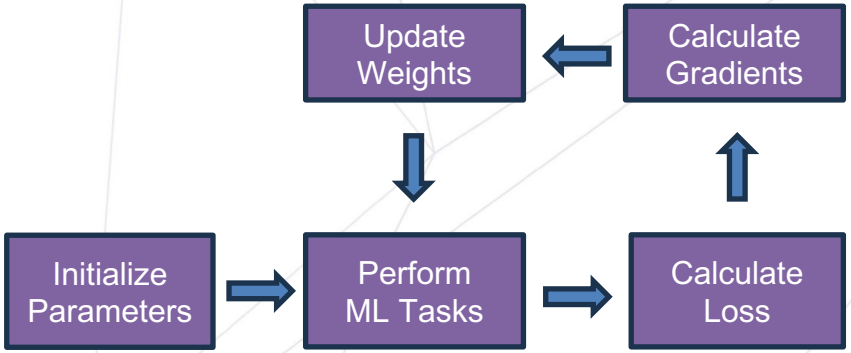
Given a set of rules, how to dynamically adjust them to get a better detection result?



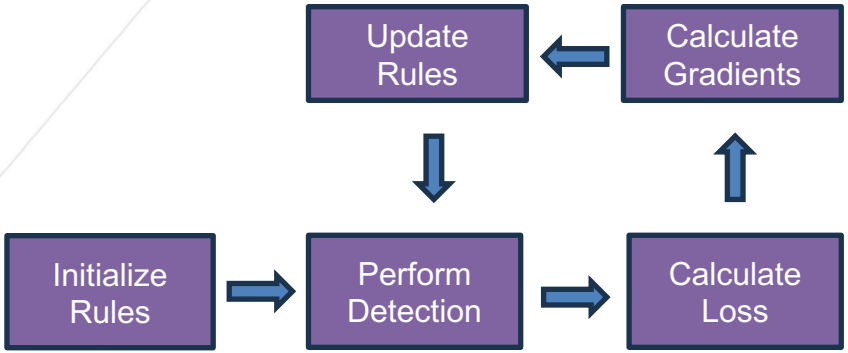
Motivation

Is it possible to maintain the lightweight and efficiency of the rule-based detection while enabling dynamic and automated rule learning?

Inspiration: Parameter learning with gradient-based methods.



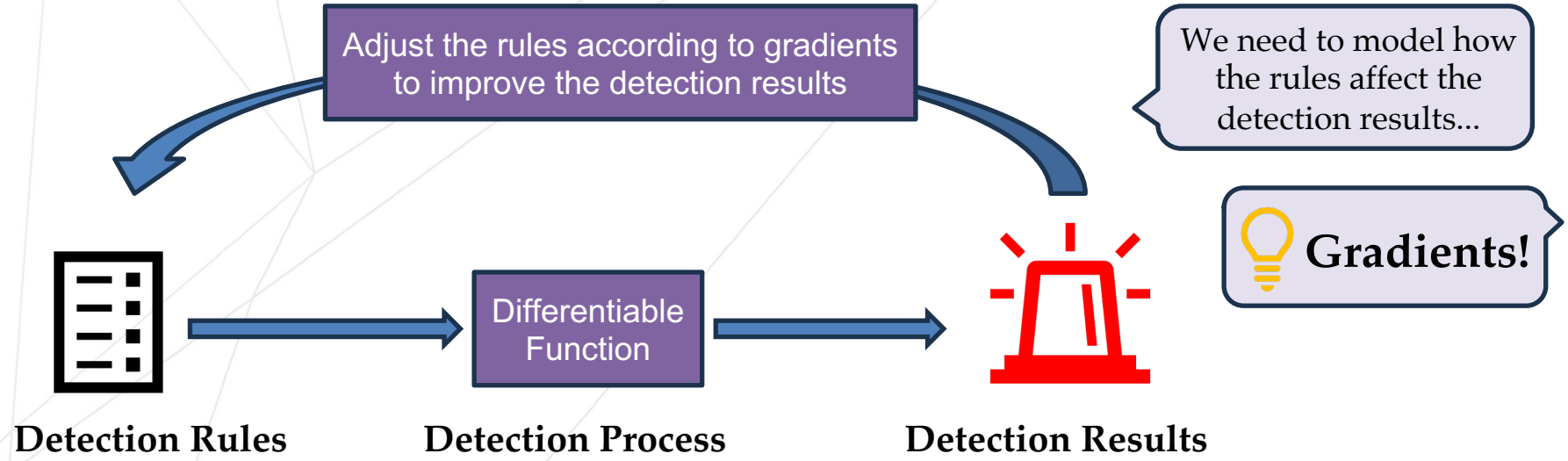
Gradient-based Parameter Learning



Gradient-based Rule Learning

One-Sentence Takeaway

We transform the non-differentiable rule-based detection process to a differentiable function, enabling us to fine-tune the detection rules based on the detection results using gradients.



Tag-Propagation: A major mechanism of rule-based detection

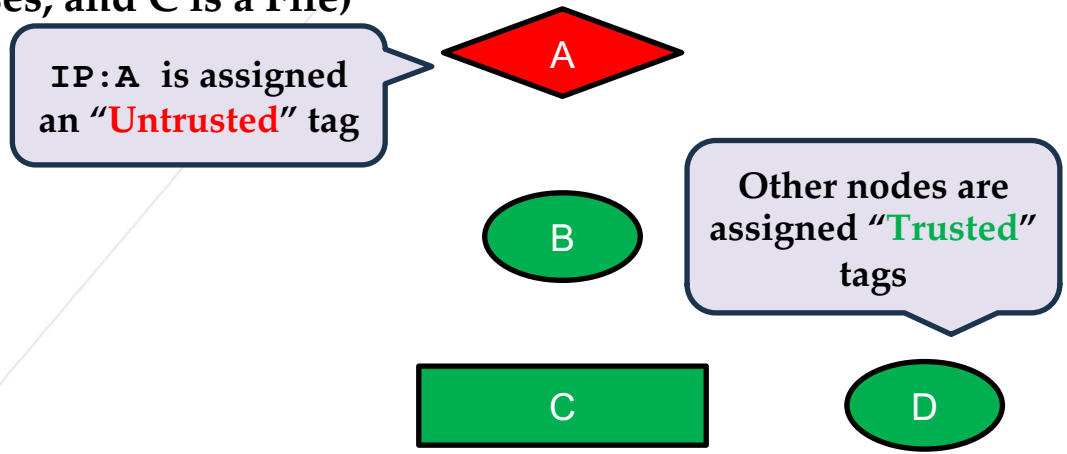
We build the differentiable detection system on tag propagation, one of the major mechanisms of rule-based detection.

Similar to *Taint Analysis*, tag-propagation system assign tags on each node in the provenance graph, propagates tags on the graph, and trigger alarms.

System Logs

Provenance Graph

(A is a IP socket, B and D are processes, and C is a File)



Tag-Propagation: A major mechanism of rule-based detection

We build the differentiable detection system on tag propagation, one of the major mechanisms of rule-based detection.

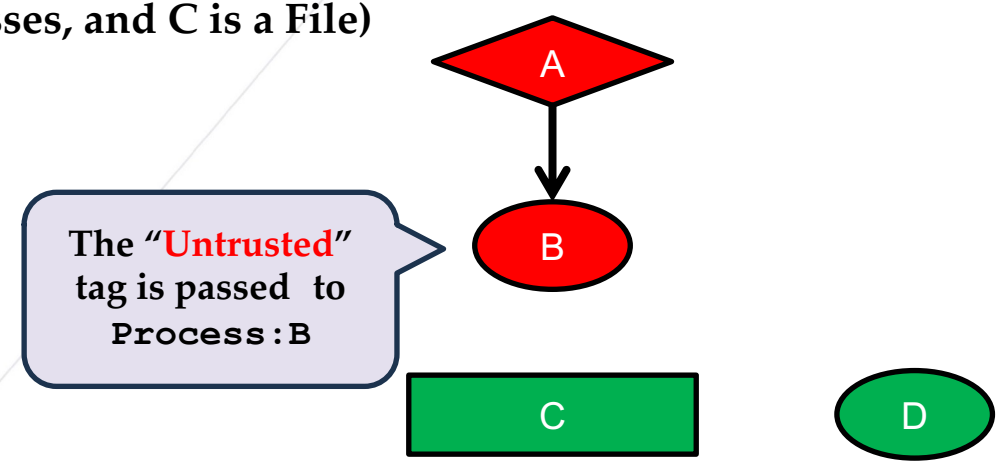
Similar to *Taint Analysis*, tag-propagation system assign tags on each node in the provenance graph, propagates tags on the graph, and trigger alarms.

System Logs

(A is a IP socket, B and D are processes, and C is a File)

```
1. Prcoess:B recv_from IP:A
```

Provenance Graph



Tag-Propagation: A major mechanism of rule-based detection

We build the differentiable detection system on tag propagation, one of the major mechanisms of rule-based detection.

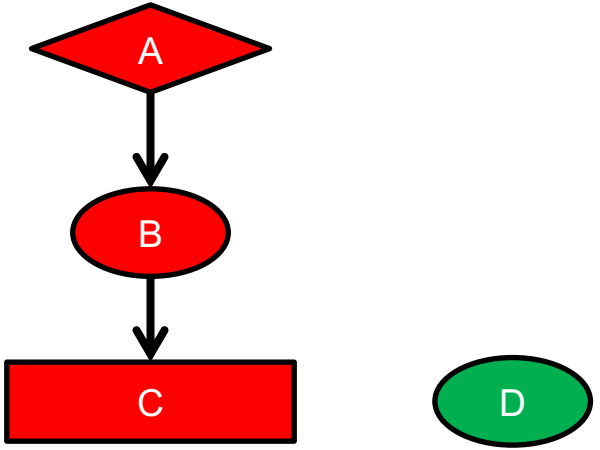
Similar to *Taint Analysis*, tag-propagation system assign tags on each node in the provenance graph, propagates tags on the graph, and trigger alarms.

System Logs

(A is a IP socket, B and D are processes, and C is a File)

```
1. Prcoess:B recv_from IP:A
2. Prcoess:B write File:C
```

Provenance Graph



Tag-Propagation: A major mechanism of rule-based detection

We build the differentiable detection system on tag propagation, one of the major mechanisms of rule-based detection.

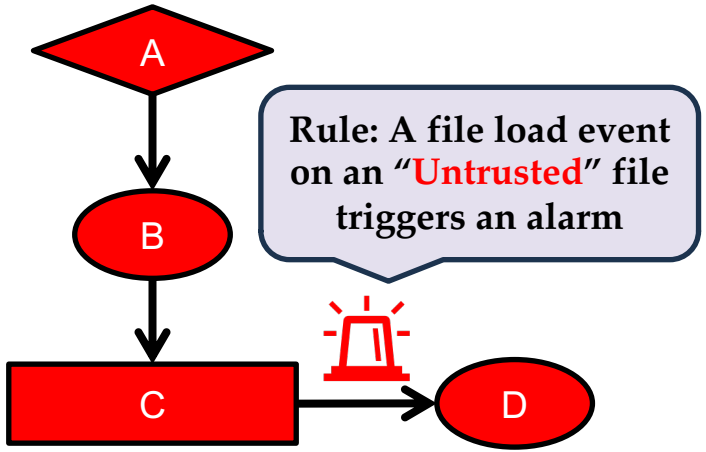
Similar to *Taint Analysis*, tag-propagation system assign tags on each node in the provenance graph, propagates tags on the graph, and trigger alarms.

System Logs

(A is a IP socket, B and D are processes, and C is a File)

```
1. Prcoess:B recv_from IP:A
2. Prcoess:B write File:C
3. Process:D load File:C
```

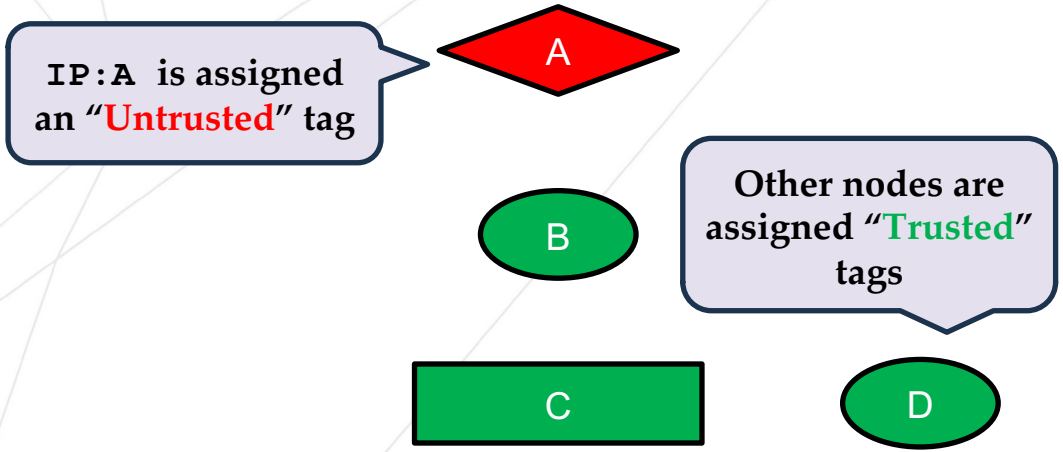
Provenance Graph



Challenges of Tag-propagation in PIDS

Tag Initialization Rules:

- How to assign initial tags for nodes without parents (like **`IP:A`**)?
- Too many "**Untrusted**" tags → Excessive false alarms
- Too many "**Trusted**" tags → Missing attacks



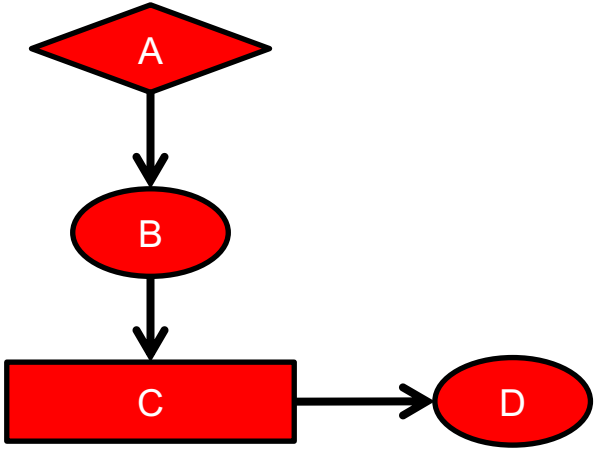
Challenges of Tag-propagation in PIDS

Tag Initialization Rules:

- How to assign initial tags for nodes without parents (like `IP:A`)?
- Too many "**Untrusted**" tags → Excessive false alarms
- Too many "**Trusted**" tags → Missing attacks

Tag Propagation Rules:

- How to propagate tags among nodes?
- Tag explosion issue → Excessive false alarms



Challenges of Tag-propagation in PIDS

Tag Initialization Rules:

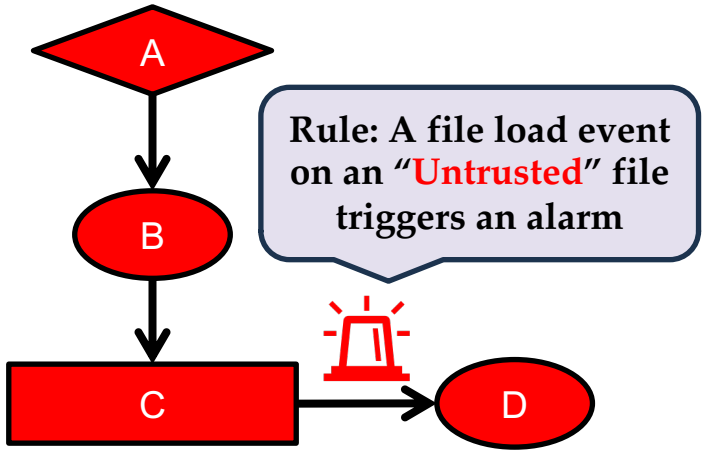
- How to assign initial tags for nodes without parents (like `IP:A`)?
- Too many "**Untrusted**" tags → Excessive false alarms
- Too many "**Trusted**" tags → Missing attacks

Tag Propagation Rules:

- How to propagate tags among nodes?
- Tag explosion issue → Excessive false alarms

Alarm Generation Rules:

- When should an alarm be triggered?
- Unable to control the sensitivity for specific events.



Challenges of Tag-propagation in PIDS

Tag Initialization Rules:

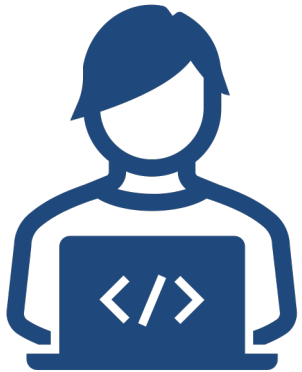
- How to assign initial tags for nodes without parents (like `IP:A`)?
- Too many "**Untrusted**" tags → Excessive false alarms
- Too many "**Trusted**" tags → Missing attacks

Tag Propagation Rules:

- How to propagate tags among nodes?
- Tag explosion issue → Excessive false alarms

Alarm Generation Rules:

- When should an alarm be triggered?
- Unable to control the sensitivity for specific events.



`All Rules set by human experts!`

Challenges of Tag-propagation in PIDS

Tag Initialization Rules:

- How to assign initial tags for nodes without parents (i.
- Too many "**Untrusted**"
-

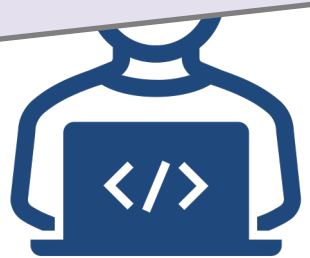
T

-
- alarms

Al **eneration Rules:**

- When should an alarm be triggered?
- Unable to control the sensitivity for specific events.

Let's incorporate gradients!

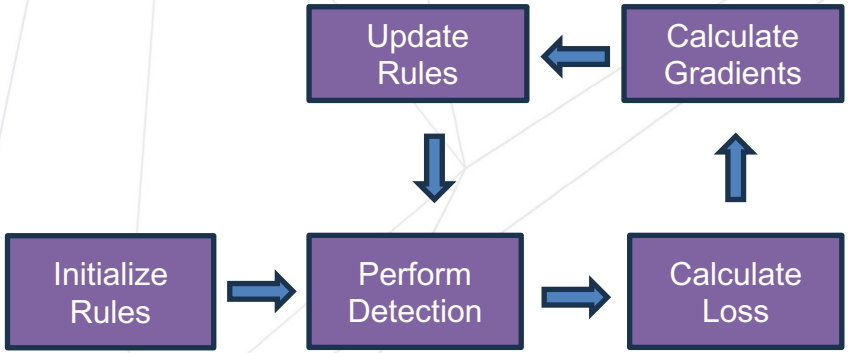


All Rules set by human experts!

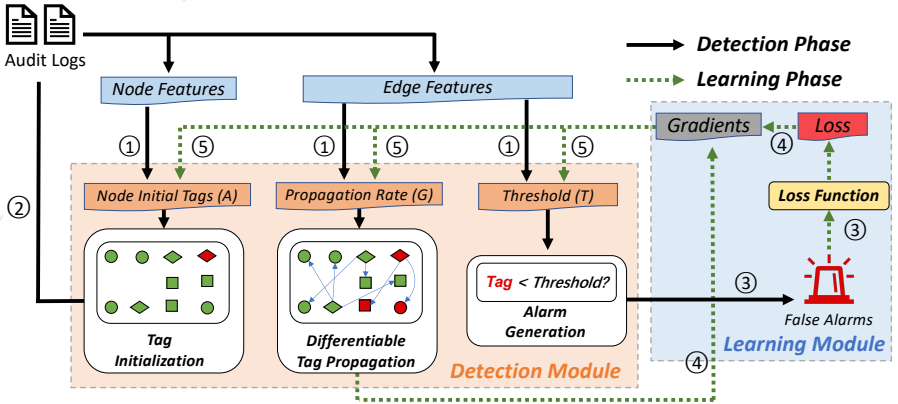
Overall Framework

CATAIN: A lightweight, adaptive provenance-based intrusion detection system.

- Address the dilemma between detection accuracy and detection efficiency
- Incorporate gradient-descent idea into optimizing traditional rule-based IDS



Gradient-based Rule Learning



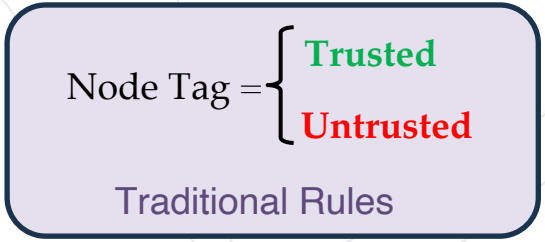
Overall Framework of CAPTAIN

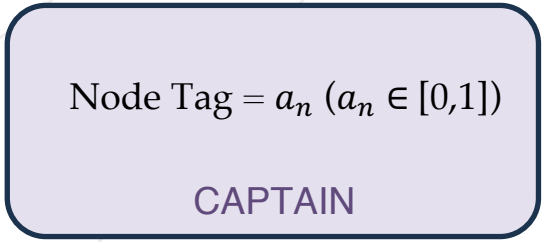
Key Question 1: How to convert rules to parameters?

Parameterize Rules

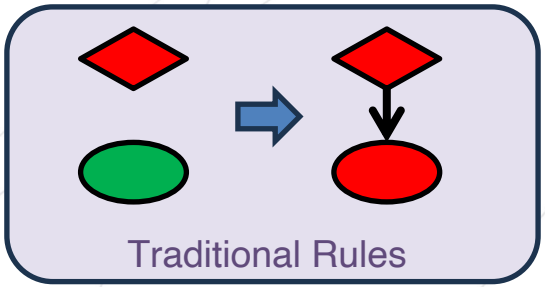
Convert three types of rules into numerical adaptive parameters (A, G, and T).

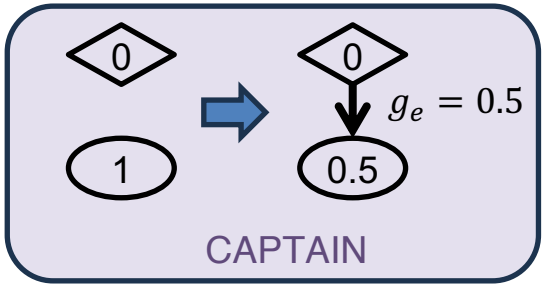
- Tag Initialization Rules: Use the **integrity score (a_n)** as the initial tag for each node





- Tag Propagation Rules: Use the **propagation rate (g_e)** for each event to partially propagate tags





Parameterize Rules

Convert three types of rules into numerical adaptive parameters (A, G, and T).

- Alarm Triggering Rules: Use a numerical **alarm threshold (t_e)** for each event e

```
if Event.Type = “load”
and File.Tag = ”Untrusted”:
Raise Alarm(“Load Untrusted Code!”)
```

Traditional Rules

$$\text{File.Tag} - t_e = \begin{cases} > 0 & \text{Benign} \\ < 0 & \text{Alarm} \end{cases}$$

CAPTAIN

Integrity scores (a_n), propagation rates (g_e), and **alarm thresholds (t_e)** provides fine-grained rules

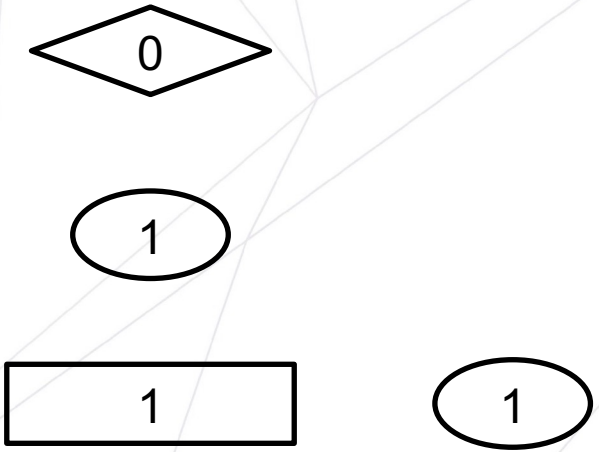
- Each node has its own a_n
- Each edge has its own g_e and t_e
- All parameters are continuous values between 0 and 1.

Key Question 2: How to calculate the gradients?

Calculate Gradients

Record & update tag gradients during tag propagation

- Initialize a_n, g_e, and t_e according to the default setting
- Perform tag propagation
- Update and record gradients when tag changes

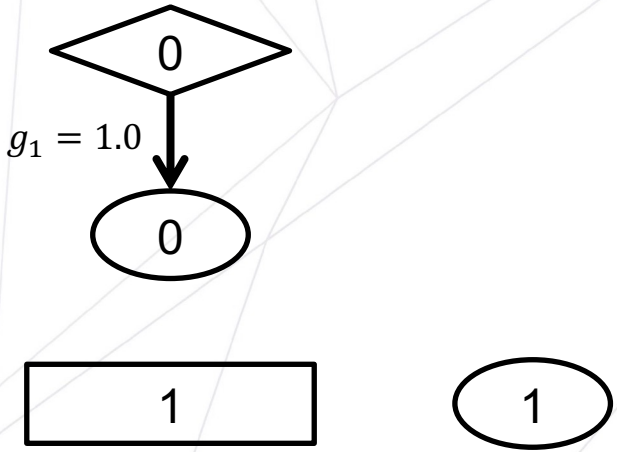


Node Tag	Gradients
A	$\partial/\partial a_A = 1$
B	$\partial/\partial a_B = 1$
C	$\partial/\partial a_C = 1$
D	$\partial/\partial a_D = 1$

Calculate Gradients

Record & update tag gradients during tag propagation

- Initialize a_n, g_e, and t_e according to the default setting
- Perform tag propagation
- Update and record gradients when tag changes

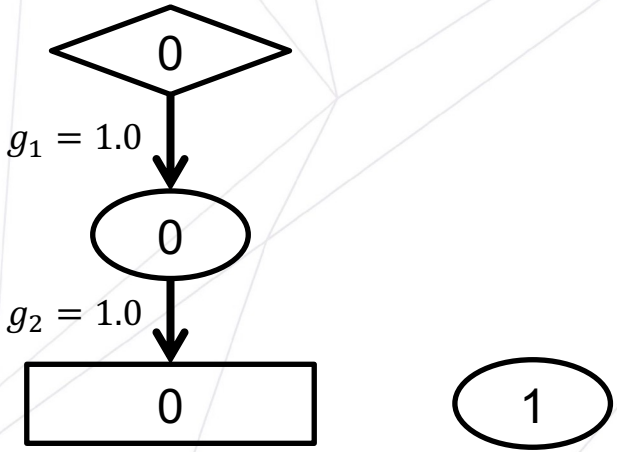


Node Tag	Gradients
A	$\partial/\partial a_A = 1$
B	$\partial/\partial a_A = 1, \partial/\partial g_1 = -1$
C	$\partial/\partial a_C = 1$
D	$\partial/\partial a_D = 1$

Calculate Gradients

Record & update tag gradients during tag propagation

- Initialize a_n, g_e, and t_e according to the default setting
- Perform tag propagation
- Update and record gradients when tag changes

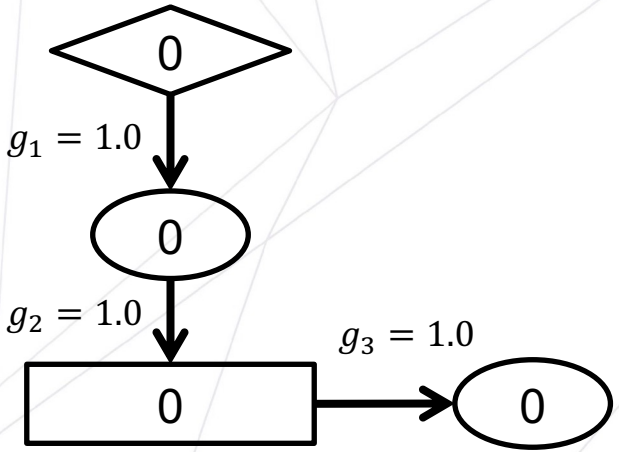


Node Tag	Gradients
A	$\partial/\partial a_A = 1$
B	$\partial/\partial a_A = 1, \partial/\partial g_1 = -1$
C	$\partial/\partial a_A = 1,\ \partial/\partial g_1 = -1, \partial/\partial g_2 = -1$
D	$\partial/\partial a_D = 1$

Calculate Gradients

Record & update tag gradients during tag propagation

- Initialize a_n, g_e, and t_e according to the default setting
- Perform tag propagation
- Update and record gradients when tag changes



Node Tag	Gradients
A	$\partial/\partial a_A = 1$
B	$\partial/\partial a_A = 1, \partial/\partial g_1 = -1$
C	$\partial/\partial a_A = 1,\ \partial/\partial g_1 = -1, \partial/\partial g_2 = -1$
D	$\partial/\partial a_A = 1,\ \partial/\partial g_1 = -1, \partial/\partial g_2 = -1,$ $\partial/\partial g_3 = -1,$

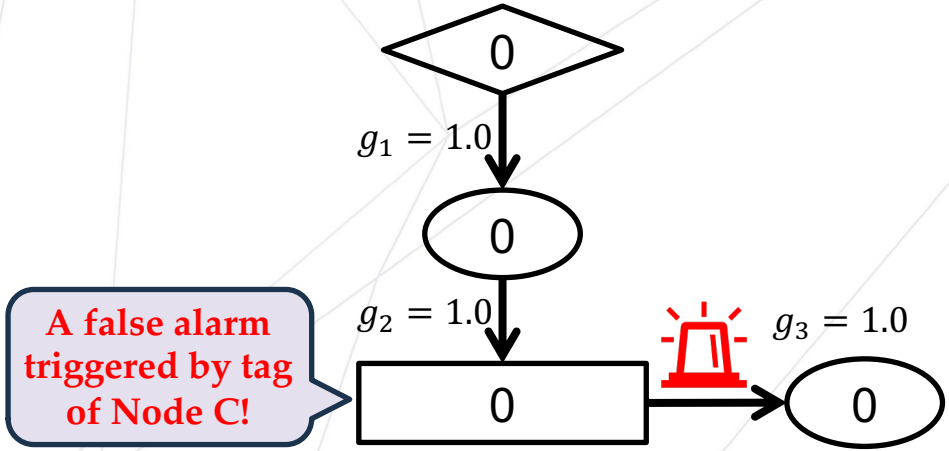
Key Question 3: How to update/train the rule parameters?

Update Rule Parameters

Calculate loss when there is a mis-detection on the training set

Update rule parameters p according to the gradients

- $p = p - learning\_rate \times gradient$



Node Tag	Gradients
A	$\partial/\partial a_A = 1$
B	$\partial/\partial a_A = 1, \partial/\partial g_1 = -1$
C	$\boldsymbol{\partial/\partial a_A = 1,\ \partial/\partial g_1 = -1, \partial/\partial g_2 = -1}$
D	$\partial/\partial a_A = 1,\ \partial/\partial g_1 = -1, \partial/\partial g_2 = -1,$ $\partial/\partial g_3 = -1,$

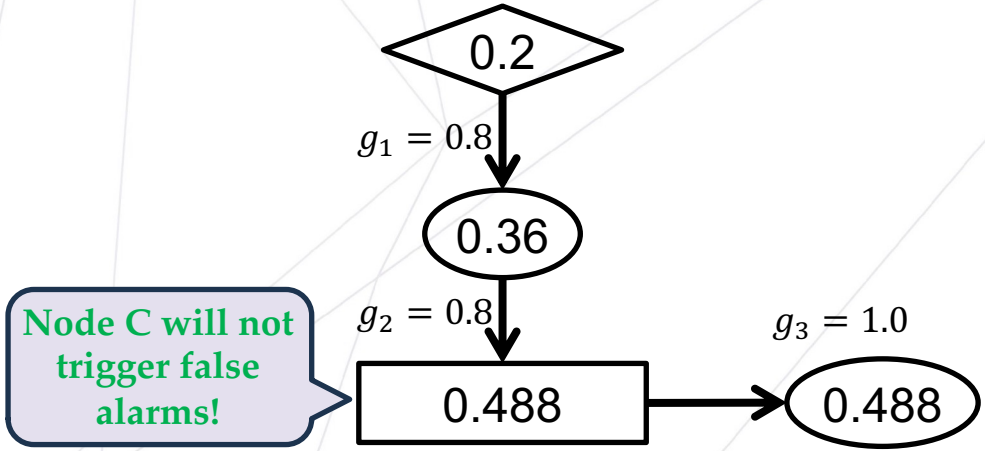
Update Rule Parameters

Calculate loss when there is a mis-detection on the training set

Update rule parameters p according to the gradients

- $p = p - learning\_rate \times gradient$

Repeat iterations until convergence and apply learned rule parameters in testing



Evaluation | Detection Accuracy

On DARPA Engagement datasets, compared to recent work, CAPTAIN

- **Node-level detection:** Reduces the false alarm rate by **90%**
- **Edge-level detection:** Reduces the false alarm rate by **93%**
- Maintain similar true positive rates to baselines

	MORSE	SHADEWATCHER	CAPTAIN
# of Consumed Events	5,188,230	724,236	5,188,230
# of False Alarm Events	22,500	2,405	1,099
False Alarm Rate	0.434%	0.332%	0.0212%

Edge-level Detection Results

	TP	FP(0-hop)	FP(1-hop)	FN(0-hop)
Engagement3 CADETS				
FLASH	16	4503	4485	10
KAIROS	15	1017	1003	11
NODLINK[1]	3	120	114	2
MORSE	16	51	43	10
CAPTAIN	16	34	26	10
Engagement3 TRACE				
FLASH	5	27202	27178	19
NODLINK[1]	4	170	170	0
MORSE	10	243	234	14
CAPTAIN	10	12	11	14
Engagement3 THEIA				
FLASH	2	53230	53050	13
KAIROS	12	3566	3422	3
NODLINK[1]	4	62	58	0
MORSE	11	220	213	4
CAPTAIN	11	194	187	4

[1]Since NODLINK only provides detected process, we evaluate it on process detection accuracy.

Node-level Detection Results

Evaluation | Efficiency & Overhead

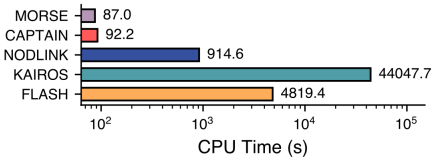
CAPTAIN is more efficient compared to **embedding-based PIDS**

- Detection Latency (reduce by at least **57%**)
- CPU Usage (reduce by at least **90%**)
- Memory Usage (reduce by at least **30%**)

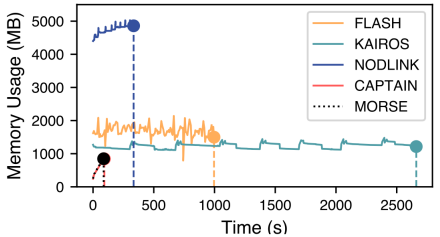
Compared to **existing rule-based PIDS**, a slightly additional overhead (**5.6%** more CPU usage) to significantly reduce FP (by over **90%**).

	Buffer Time	Preprocessing Time	Detection Time
Engagement3 TRACE			
FLASH	57:49	107:50	64:24
SHADEWATCHER	N/A[1]	100:22	3:40[2]
NODLINK	00:10	135:42	2:48
MORSE	0	58:20	1:29
CAPTAIN	0	58:20	1:31
Engagement3 CADETS			
KAIROS	15:00	15:34	29:46
FLASH	82:52	18:57	7:41
NODLINK	00:10	6:18	6:41
MORSE	0	7:22	1:19
CAPTAIN	0	7:22	1:23

[1] We did not find a clear number in their codes or paper.
[2] SHADEWATCHER extracts the last 10% interactions as the testing set, while the testing set of us is around 2.5 times larger.



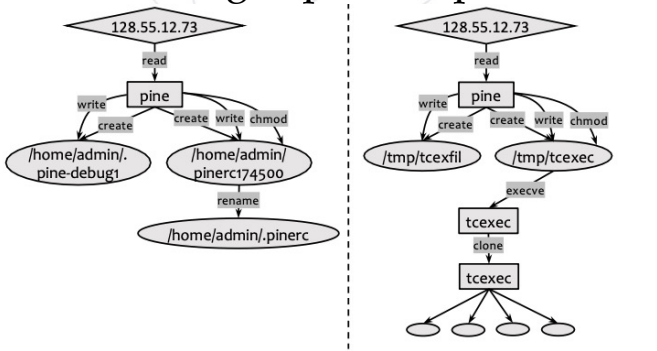
(a) Total CPU time used



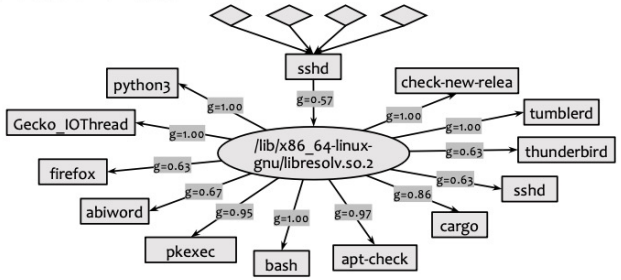
(b) Memory usage over time

Evaluation | Case Study

- Each alarm from CAPTAIN has **clear semantics** and **explicable detection process**
- CAPTAIN's customized rule parameters for each individual nodes and edges can:
  - distinguish between similar graph patterns
  - address tag explosion problem



(a) The graph on the left side represents normal behavior in the training set, while the graph on the right side depicts an attacker executing a malicious email attachment.



(b) CAPTAIN assigns different propagation rates (g_e) to events to control dependency explosion more precisely.

Conclusion

- During the battle against APT attacks, there is a dilemma between detection accuracy and efficiency faced by existing **rule-based PIDS** and **embedding-based PIDS.**
- We proposed CAPTAIN, a differentiable rule-based detection framework that can optimize parameterized detection rules using gradient descent algorithms.
- The results on multiple datasets demonstrate the superiority of CAPTAIN compared to existing rule-based and embedding-based PIDS.

Thanks for your listening!

To Access Code & Experiments

https://github.com/LexusWang/CAPTAIN

lingzhiwang2025@u.northwestern.edu