Orthrus: Achieving High Quality of Attribution in Provenance-based Intrusion Detection Systems

(USENIX Security'25)

EPITA Seminar, June 26th 2025

Tristan Bilot

Université Paris-Saclay, Isep, Iriguard, University of British Columbia (UBC)

Outline

- Background
- Design
- Evaluation
- Conclusion
- Bonus: PIDSMaker

Advanced Persistent Threats (APT)

Sophisticated, targeted, and prolonged cyberattacks carried out by nation-states or hacker groups

Frequency: **74%** increase in APT attempts in 2024 [1].

Key figures:

- **150** days on average before being detected [2].
- **60%** are attributed to nation-states (e.g., China, Russia, North Korea) in 2024 [3].
- **89%** are associated with espionage [2].

[4] https://securitybrief.asia/story/advanced-persistent-threats-rise-by-74-in-2024-report
[5] https://fr.vectra.ai/topics/advanced-persistent-threat
[6] https://go.crowdstrike.com/2025-global-threat-report.html

Advanced Persistent Threats (APT)

Principaux groupes :

APT28 - Fancy Bear (Russie RU) : Fuite électorale américaine via WikiLeaks (2016)

APT38 - Lazarus (Corée du Nord KP) : Ransomware mondial WannaCry (2017)

APT41 - Wicked Panda (Chine CN) : Espionnage et cybercrime financier (2021)

Wannacry ransomware

System Provenance

- **System Provenance** records interactions between system objects at the kernel level.

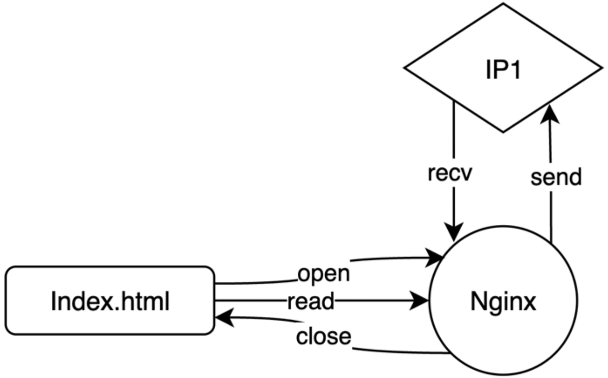
System Provenance

- **System Provenance** records interactions between system objects at the kernel level.
- A **system provenance graph** models interactions between system entities (e.g. Processes, Files, or Sockets, etc.) as a graph
- The graph is usually ***directed***, ***attributed*** and ***dynamic*** graph.

System Events

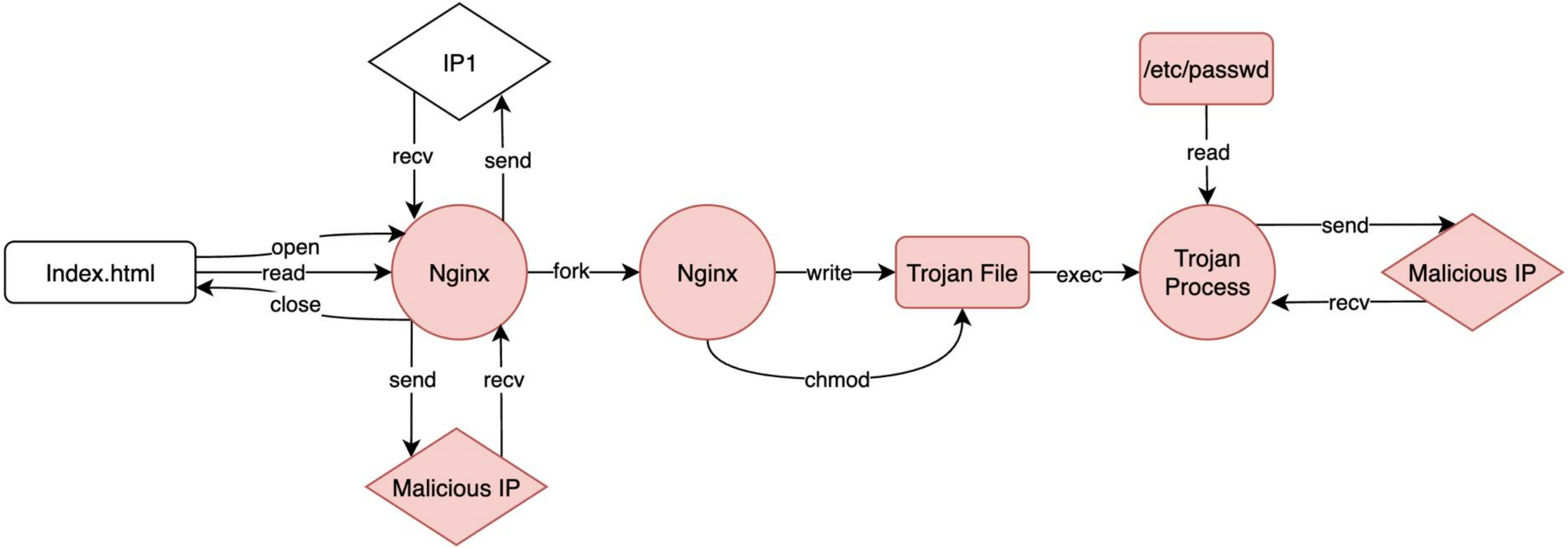
<Timestamp 1> Nginx, receive, IP1
<Timestamp 2> Nginx, open, index.html
<Timestamp 3> Nginx, read, index.html
<Timestamp 4> Nginx, send, IP1
<Timestamp 5> Nginx, close, Index.html
……

System Provenance Graph



System Provenance

When there exists attacks, the abnormal system behaviors result in a different structure of the provenance graph.



Provenance-based Intrusion Detection System (PIDS)

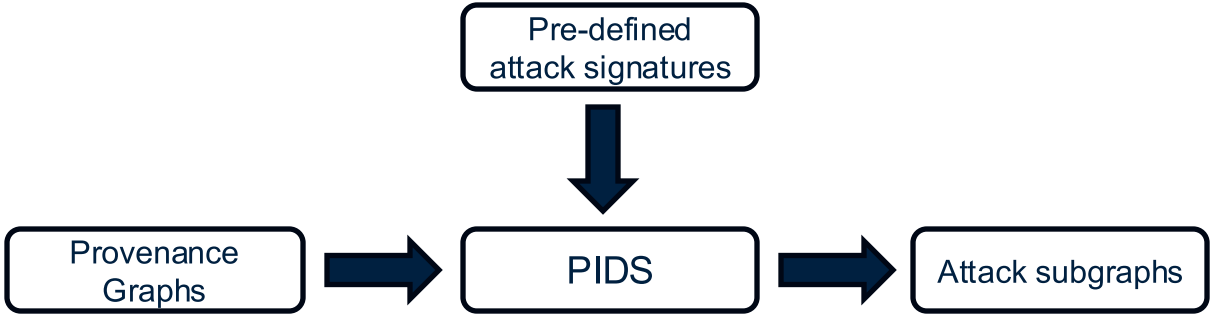
A PIDS aims to detect the malicious system behaviors in provenance graphs.

Category:

- Signature-based PIDS
- Anomaly-based PIDS

Signature-based PIDS

A signature-based PIDS detects attacks based on existing signatures/patterns from past attacks



Pros:

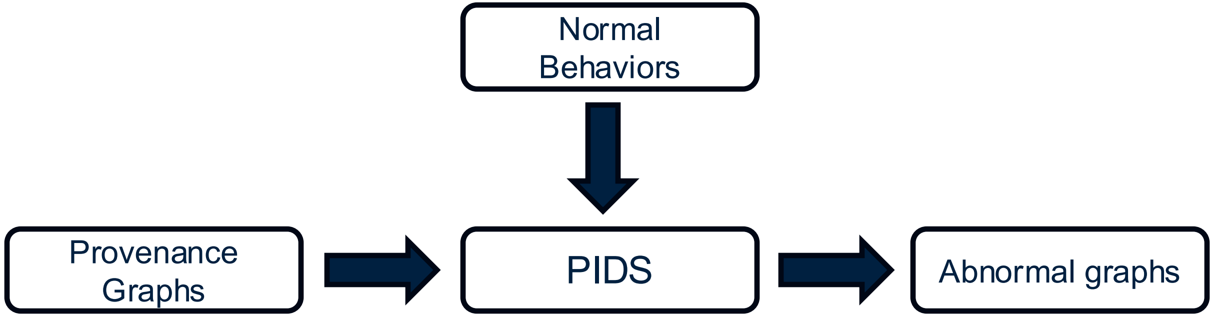
- Detect well existing attacks
- Lightweight, easy to install

Cons:

- Requires labelled attacks
- Cannot generalize to variants (e.g. obfuscated) or new attacks (e.g. zero-days/APTs)

Anomaly-based PIDS

An anomaly-based PIDS considers attacks as highly-anomalous events and do not rely on existing attacks



Pros:

- Detect variants & unknown attacks such as zero-days and APTs
- Do not need labelled attacks

Cons:

- Sensitive to false positives
- Sensitive to concept drift

Limitations in SOTA PIDSs: low attribution quality

Research problem:

Existing anomaly-based PIDSs neglect attribution quality

Attribution quality?

It refers to the amount of effort required from a human analyst to investigate an IDS' predictions

Reasons:

- The significant imbalance between classes in the intrusion detection problem
- They generate too many FPs and use evaluation strategies to ignore those FPs

Limitations in reported attribution quality

Significant imbalance in intrusion detection problem

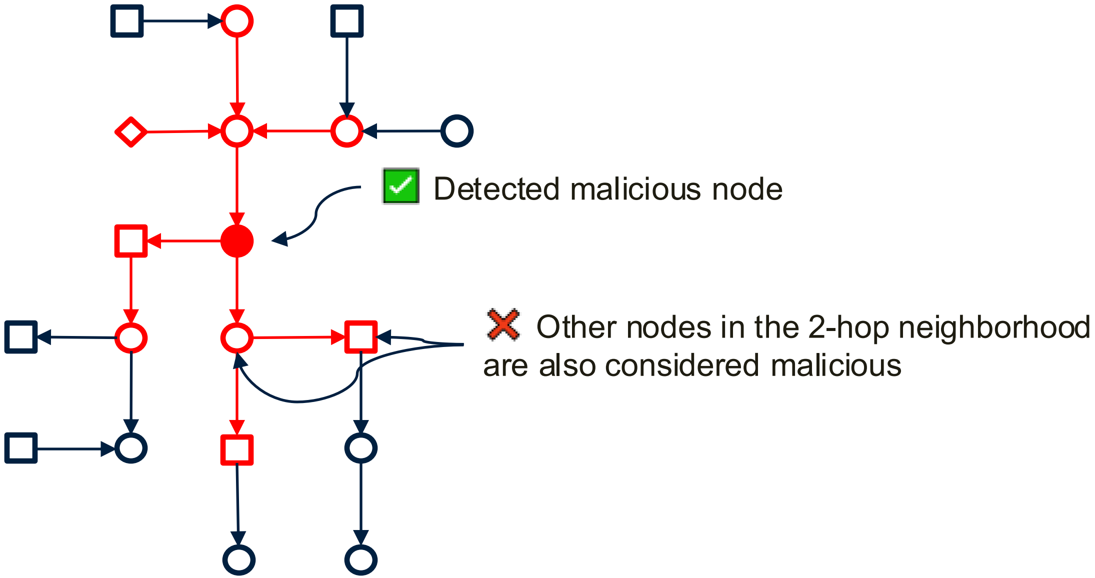
Prevalence of malicious nodes in the test set ranges from ~ 1 : 10,000 to ~ 1 : 1,000,000

Datasets	Training	Validation	Test	Total	Neigh.	Batch	Source	Ours	Prevalence
E3-CADETS	449,325	40,581	268,153	758,059	12,852	4,929	2,062	68	2.5×10^{-4}
E3-THEIA	410,023	34,365	699,295	1,143,683	25,362	51,098	35,794	118	1.7×10^{-4}
E3-CLEARSCOPE	132,121	797	111,394	244,312	32,451	8,727	2,750	41	3.7×10^{-4}
E5-CADETS	3,275,875	1,245,539	3,111,378	7,632,792	20,524	717,783	401,065	123	4.0×10^{-5}
E5-THEIA	745,773	234,896	747,452	1,728,121	162,714	61,368	9,374	69	9.2×10^{-5}
E5-CLEARSCOPE	171,771	3,842	150,725	326,338	48,488	8,636	1,020	51	3.4×10^{-4}

Limitations in reported attribution quality

Evaluation strategies of prior work: 1) Neighborhood Approach

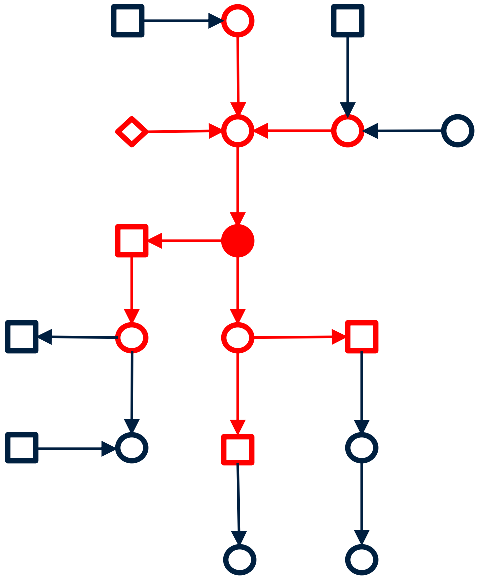
ThreaTrace (TIFS’22), Flash (S&P’24) and Magic (USENIX’24) consider 2-hop neighbors of a malicious node as malicious

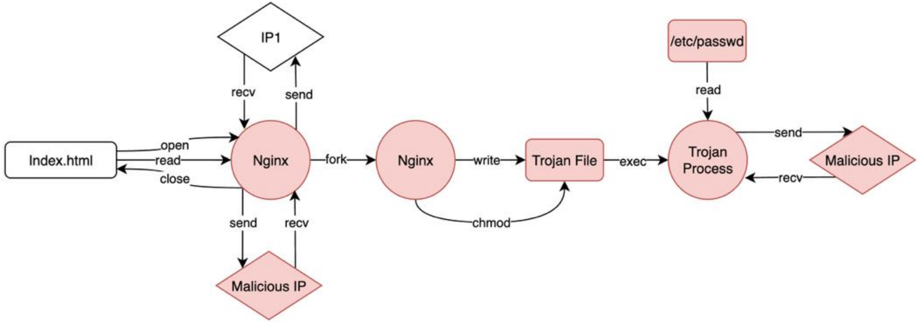


Limitations in reported attribution quality

Evaluation strategies of prior work: 1) Neighborhood Approach

ThreaTrace (TIFS'22), Flash (S&P'24) and Magic (USENIX'24) consider 2-hop neighbors of a malicious node as malicious





Malicious activity propagates to neighbors so they assume all neighbors are malicious

Limitations in reported attribution quality

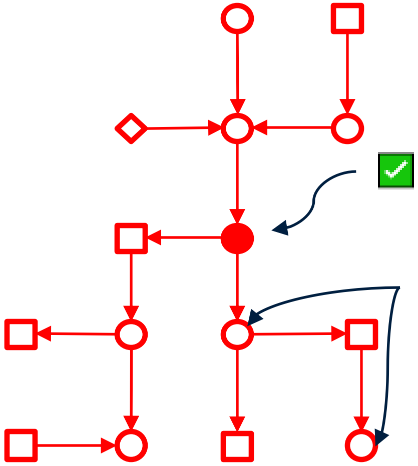
Evaluation strategies of prior work

2) Batch Approach

Kairos (S&P'24) and EdgeTorrent (RAID'23)

consider all nodes in the same batch as malicious

Batch graph



✅ Detected malicious node

❌ All nodes in the batch are also considered malicious

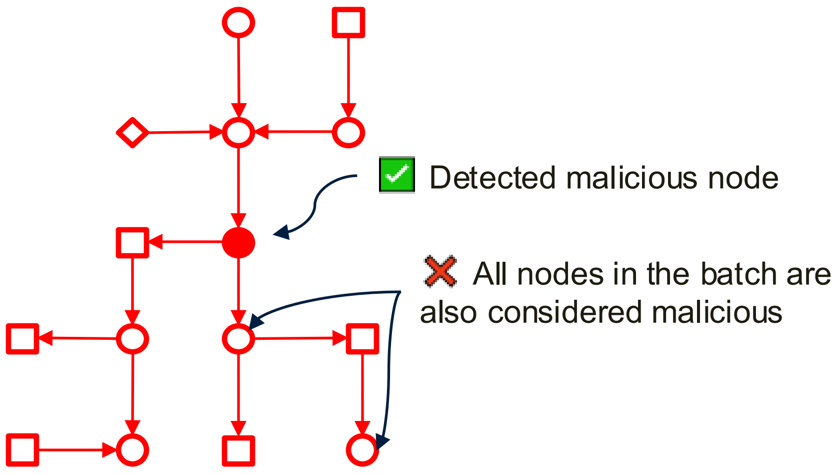
Limitations in reported attribution quality

Evaluation strategies of prior work

2) Batch Approach

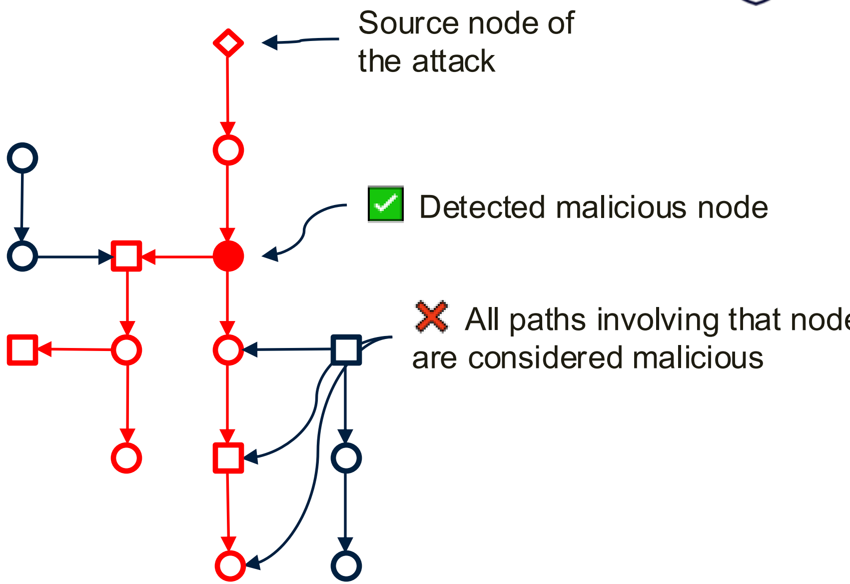
Kairos (S&P'24) and EdgeTorrent (RAID'23) consider all nodes in the same batch as malicious

Batch graph



3) Source Approach

R-CAID (S&P'24) identifies the source node and all parents/children are considered as malicious



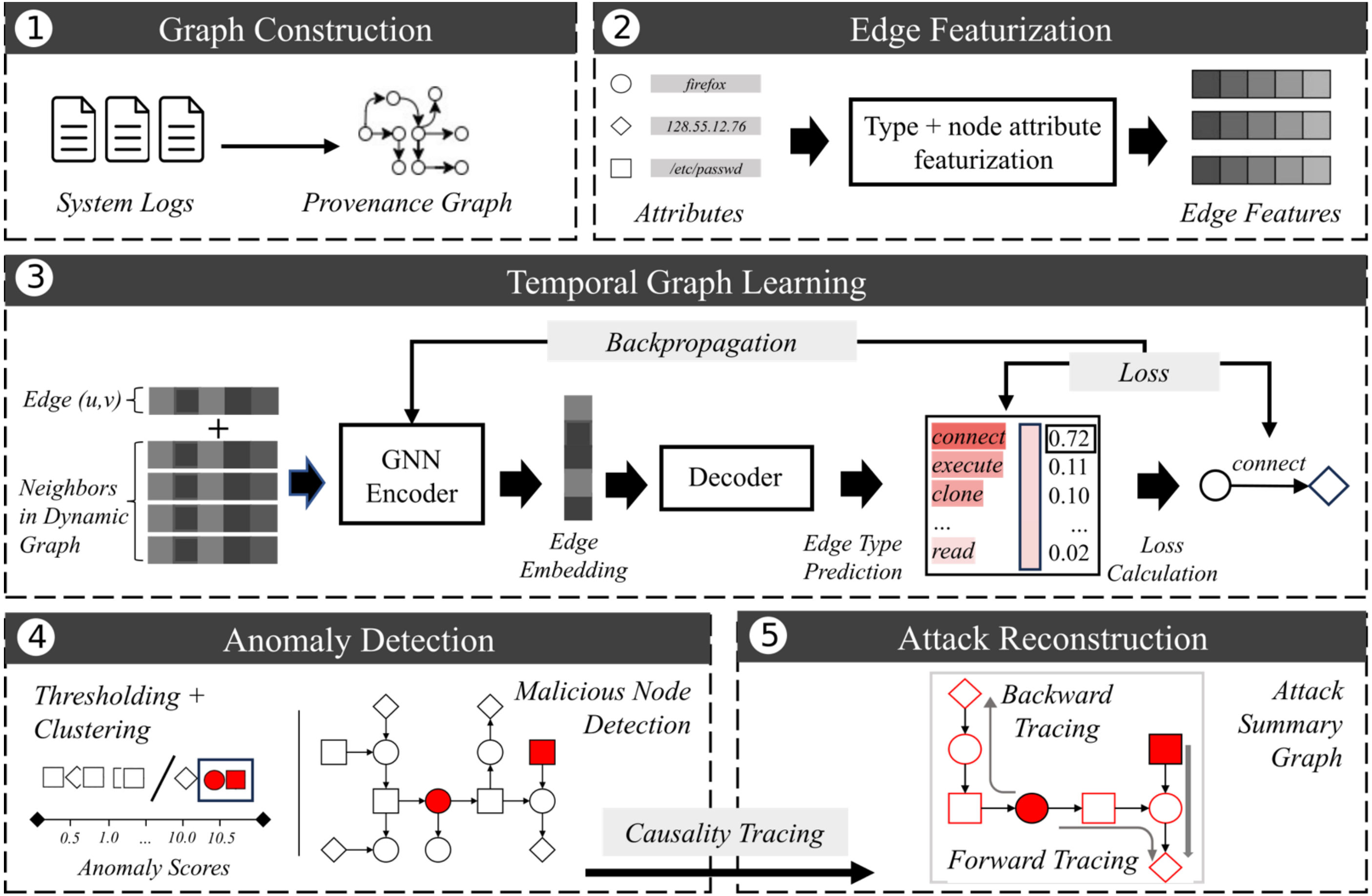
Limitations in reported attribution quality

Our approach to evaluation

Orthrus does **not** use any strategies to reduce false positives, it detects attacks at the **node level**

Datasets	Training	Validation	Test	Total	Neigh.	Batch	Source	Ours	Prevalence
E3-CADETS	449,325	40,581	268,153	758,059	12,852	4,929	2,062	68	2.5×10^{-4}
E3-THEIA	410,023	34,365	699,295	1,143,683	25,362	51,098	35,794	118	1.7×10^{-4}
E3-CLEARSCOPE	132,121	797	111,394	244,312	32,451	8,727	2,750	41	3.7×10^{-4}
E5-CADETS	3,275,875	1,245,539	3,111,378	7,632,792	20,524	717,783	401,065	123	4.0×10^{-5}
E5-THEIA	745,773	234,896	747,452	1,728,121	162,714	61,368	9,374	69	9.2×10^{-5}
E5-CLEARSCOPE	171,771	3,842	150,725	326,338	48,488	8,636	1,020	51	3.4×10^{-4}

Design: Overview of Orthrus



Design: Graph Construction

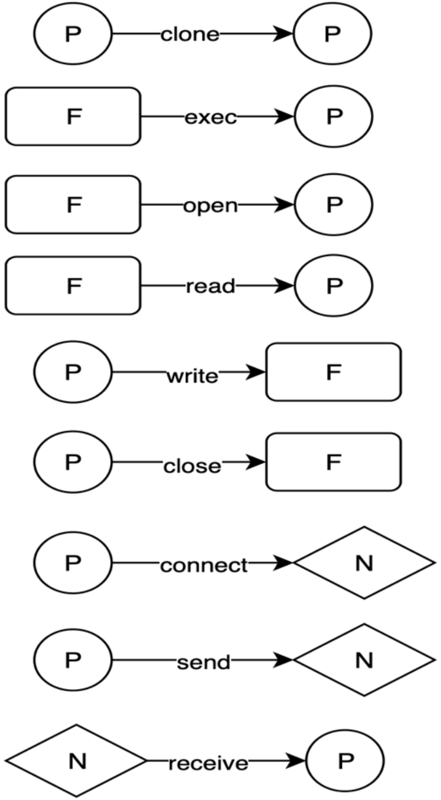
1. System Objects and corresponding attributes

System Objects	Attributes
Process	executable name, cmd options
File	file path
Netflow	local address, local port, remote address, remote port

Design: Graph Construction

2. System Events and Edge Directions

System Objects	Attributes
Process	executable name, cmd options
File	file path
Netflow	local address, local port, remote address, remote port

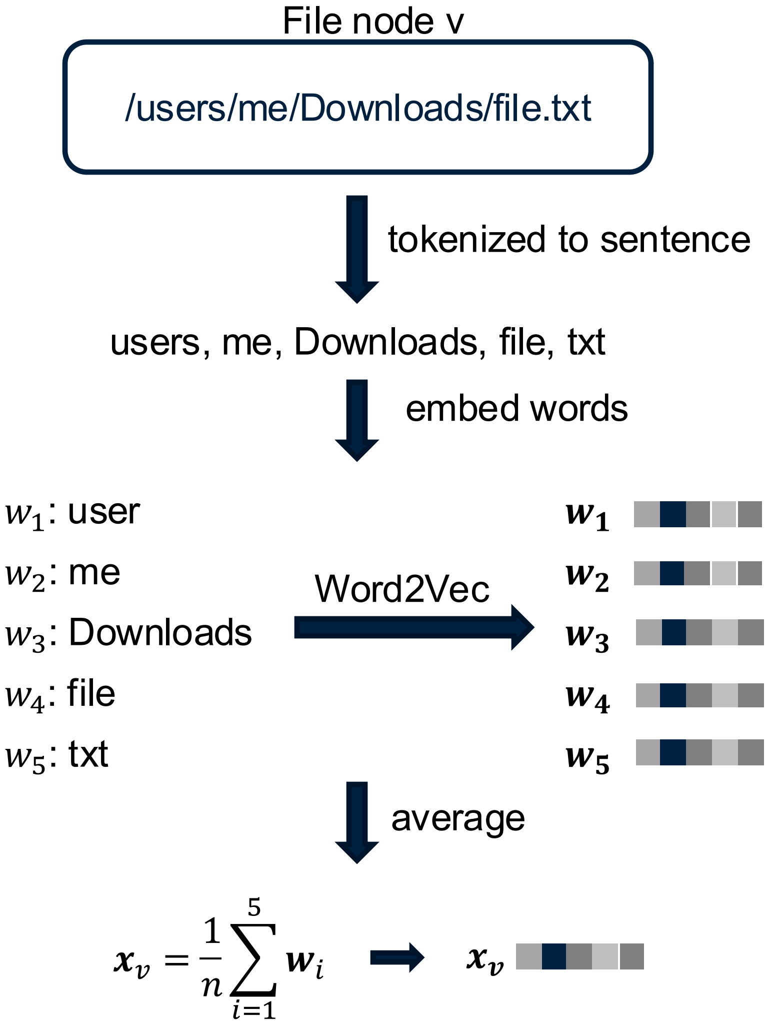


Design: Edge Featurization

1. Encoding Node Attributes

- Textual attributes of nodes are tokenized to sentences
- Embed each word in the sentence by Word2Vec
- Node embedding is the average of word embeddings

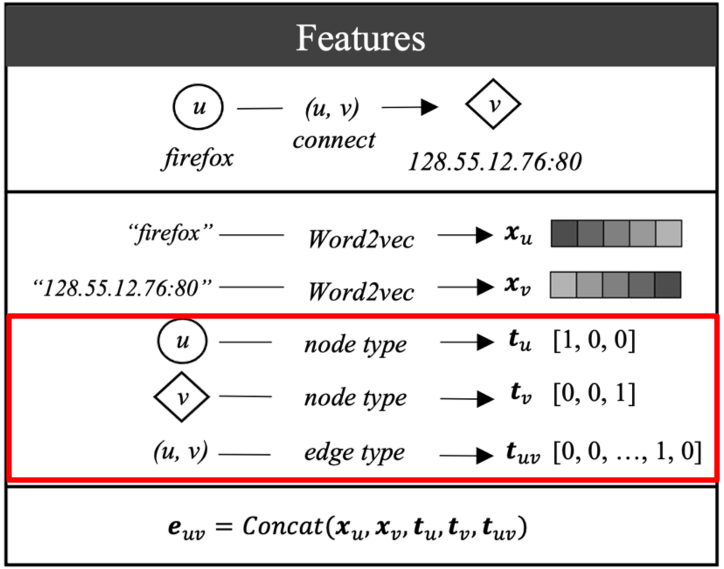
$$\boldsymbol{x}_v = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{w}_i$$



Design: Edge Featurization

2. Encoding Types

- 3 node types and 10 edge types are encoded to one-hot embeddings



Design: Edge Featurization

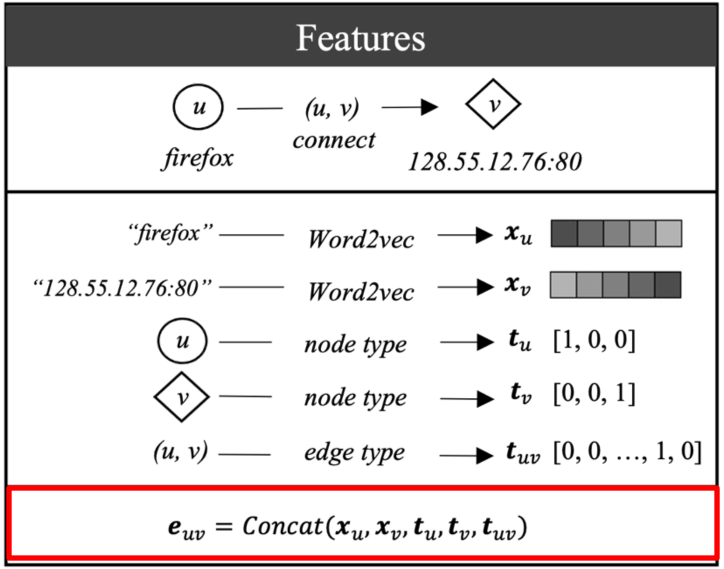
2. Encoding Types

- 3 node types and 10 edge types are encoded to one-hot embeddings

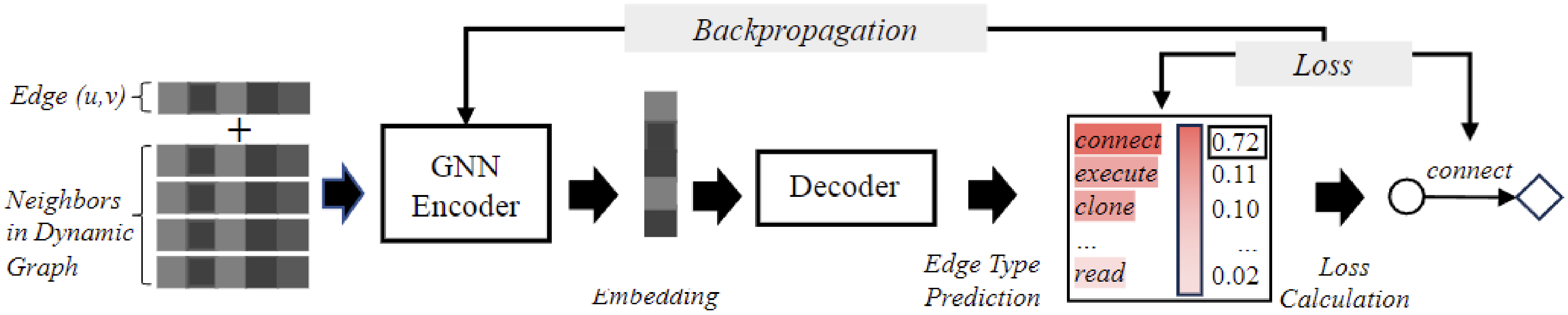
3. Compute Edge Feature Vector

- Edge feature vectors are computed as the concatenation of node attribute embeddings and type embeddings

$$e_{uv} = Concat(x_u, x_v, t_u, t_v, t_{uv})$$



Design: Temporal Graph Learning

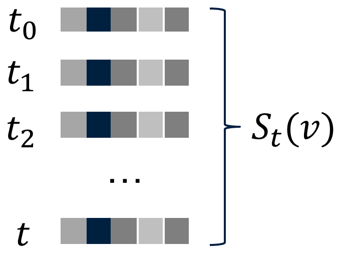


- An encoder-decoder architecture is used to capture both temporal and spatial semantics of events
- A GNN encoder learns spatio-temporal edge embeddings from the *dynamic graph*
- The decoder is trained to predict edge types from embeddings and reconstruction errors are calculated
- Orthrus trains on benign data to learn normal host behaviors, which is self-supervised

Design: Temporal Graph Learning

1. GNN Encoder

Dynamic Graph → Sampling → Attention-based GNN → Node Embedding $\boldsymbol{h}_v$



- For event from u to v at time t, information of u is aggregated to v
- The N last events connecting to v are sampled from all previous events $S_t(v)$ for information aggregation

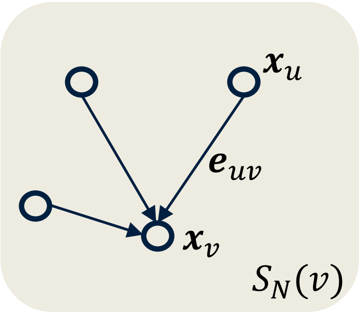
$$S_t(v) = \{(u, v) \in E \mid t_{uv} < t, t_{uv} \in T\}$$

$$S_N(v) = SAMPLE(S_t(v), N, T, t)$$

Design: Temporal Graph Learning

1. GNN Encoder

Dynamic Graph → Sampling → Attention-based GNN → Node Embedding $\boldsymbol{h}_v$



- Orthrus employs an attention-based GNN encoder
- Attention mechanism allows each node to focus on the most relevant neighboring information during aggregation
- *Attention coefficients* are calculated between v and each sampled neighbor node $u \in S_N(v)$

$$\boldsymbol{\alpha}_{u,v} = softmax\left(\frac{(\boldsymbol{W}_3\boldsymbol{x}_v)^T(\boldsymbol{W}_4\boldsymbol{x}_u + \boldsymbol{W}_5\boldsymbol{e}_{uv})}{\sqrt{d}}\right)$$

$$\boldsymbol{h}_v = \boldsymbol{W}_1\boldsymbol{x}_v + \sum_{(u,v)\in S_N} \boldsymbol{\alpha}_{u,v}(\boldsymbol{W}_2\boldsymbol{x}_u + \boldsymbol{W}_3\boldsymbol{e}_{uv})$$

Design: Temporal Graph Learning

2. Decoder

- The decoder is trained to predict edge type $\hat{y}_{uv}$ based on the embeddings of end nodes $\boldsymbol{h}_u$ and $\boldsymbol{h}_v$

$$\hat{y}_{uv} = \sigma(\boldsymbol{W}_g \cdot Concat(\boldsymbol{W}_s\boldsymbol{h}_u, \boldsymbol{W}_d\boldsymbol{h}_v))$$

- The reconstruction error is set as the Cross-Entropy (CE) loss across all edge types to prediction

$$L_{uv} = CE(\hat{y}_{uv}, y_{uv})$$

- By optimizing this loss, the model is trained to learn temporal and spatial patterns of normal behaviors
- During inference, **abnormal** events lead to **high** reconstruction error, which is assigned to the end node of the edges

Design: Anomaly Detection

1. Automatic Anomaly Thresholding

- The predicted distribution of anomaly scores must be separated in malicious/benign classes
- We compute a threshold as the max loss in the validation set (containing benign-only data)

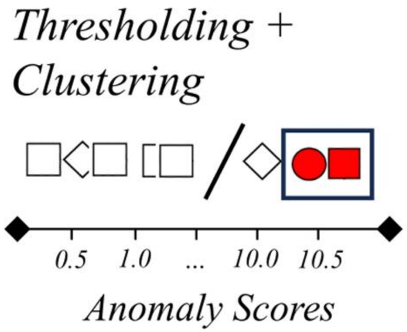
Design: Anomaly Detection

1. Automatic Anomaly Thresholding

- The predicted distribution of anomaly scores must be separated in malicious/benign classes
- We compute a threshold as the max loss in the validation set (containing benign-only data)

2. Outlier Clustering

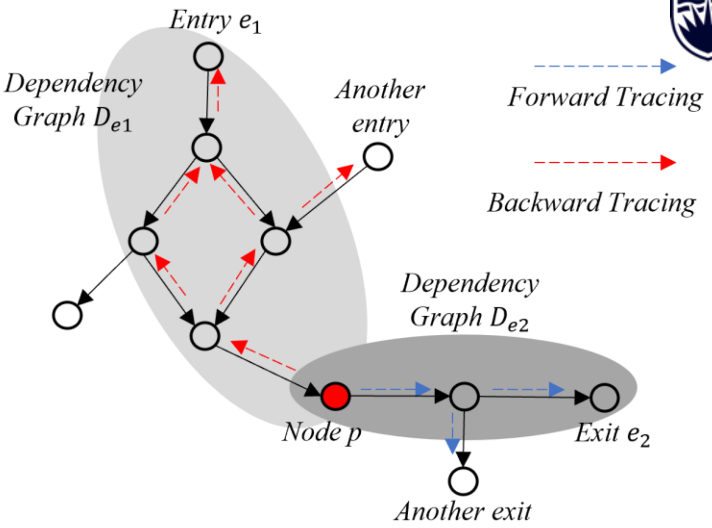
- We then separate thresholded nodes in two clusters (with K-means)
- The highest cluster contains most anomalous nodes



Design: Attack Reconstruction

1. Dependency Analysis

- Aim to predict malicious edges
- Conduct causality analysis starting from detected anomaly node p and potential attack *entry* and *exit* nodes are identified
- A *dependency graph* is defined as the set of all paths between an entry/exit and p



Design: Attack Reconstruction

2. Critical Dependency Identification

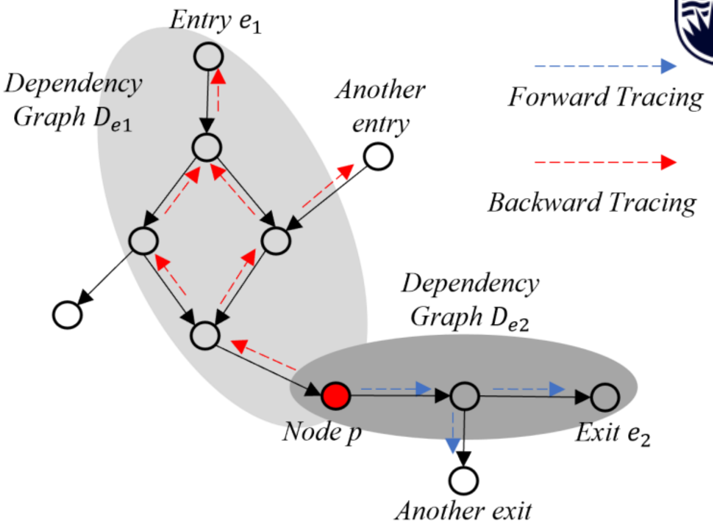
- We associate every node in dependency graphs two scores: 1) *degree score* and 2) *anomaly score*

$$f_D(u) = Outdegree(u)/Indegree(u)$$

$$f_A(u) = \frac{1}{|E|}\sum_{uv \in E_u} L_{uv}$$

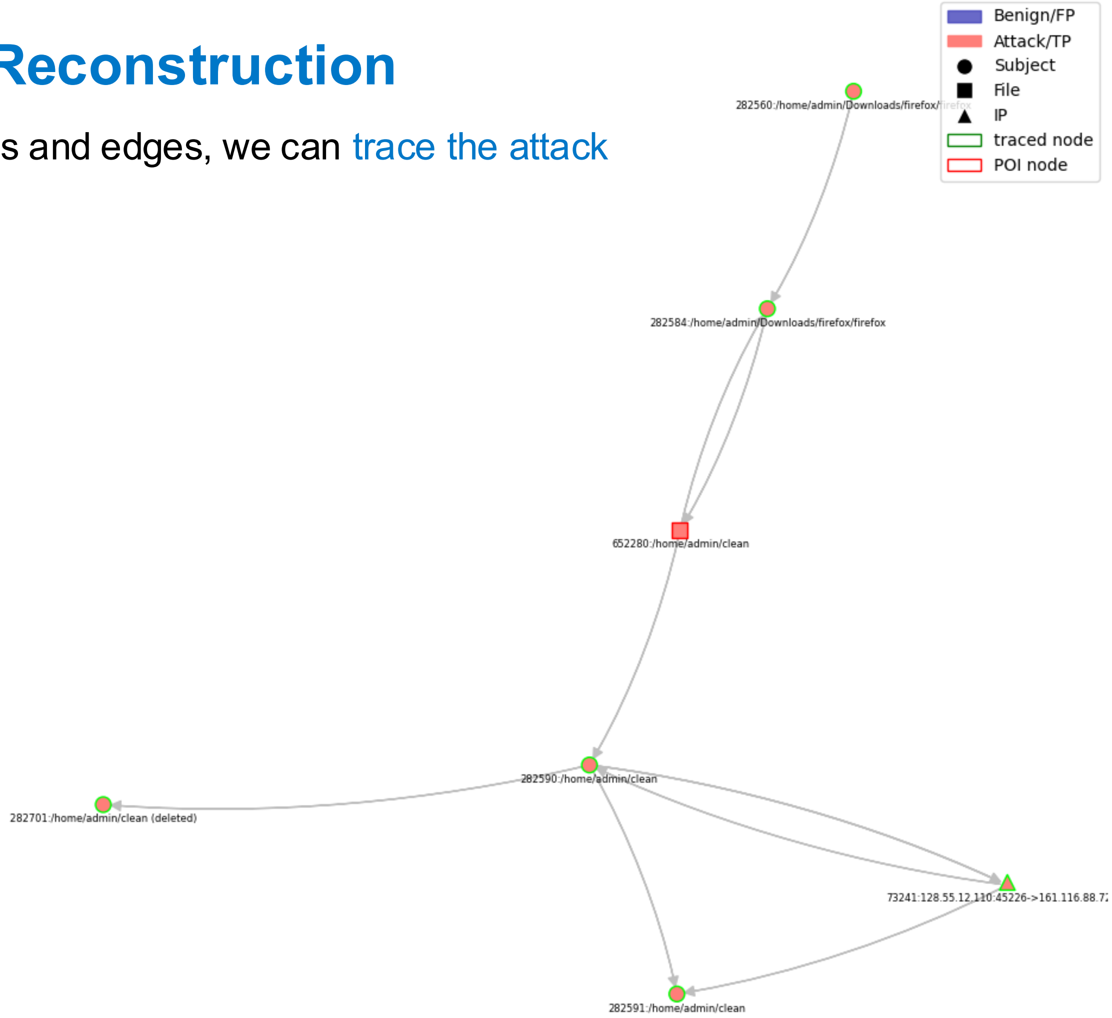
- A *criticality score* is calculated for each dependency graph and corresponding entry/exit
- Dependency graphs corresponding to top-1 entry and top-1 exit are selected as the reconstruction of attack

$$f_C(e) = \frac{1}{|V_e|}\sum_{u \in V_e} (\hat{f}_D(u) + \hat{f}_A(u))$$



Design: Attack Reconstruction

Based on detected nodes and edges, we can trace the attack



Evaluation

RQ1: Is Orthrus able to detect all attacks?

RQ2: What is the quality of attribution?

RQ3: Is Orthrus computationally efficient?

RQ4: How do hyperparameters influence performance?

RQ5: How the different Orthrus components contribute to overall performance?

RQ6: How robust is Orthrus against adversarial attacks?

Evaluation: Datasets

- Benchmark datasets published by DARPA’s Transparent Computing (TC) programs.
- TC organized several adversarial engagements that simulated real-world APTs on enterprise networks.

  - Simulation Duration: two weeks

  - Benign activities: browse website, check emails, SSH connection, etc.

  - Attack activities: browser vulnerability exploitation, malicious process execution, sensitive data leakage.

Evaluation: Datasets

- We use the well-established datasets from DARPA's Transparent Computing (TC) program
- DARPA conducted several real-world APT attacks in their networks

  - Simulation Duration: 2 weeks

  - Benign activities: browse website, check emails, SSH connection, etc.

  - Attack activities: browser vulnerability exploitation, malicious process execution, sensitive data leakage.

Datasets	Training	Validation	Test	Total	Neigh.	Batch	Source	Ours	Prevalence
E3-CADETS	449,325	40,581	268,153	758,059	12,852	4,929	2,062	68	2.5×10^{-4}
E3-THEIA	410,023	34,365	699,295	1,143,683	25,362	51,098	35,794	118	1.7×10^{-4}
E3-CLEARSCOPE	132,121	797	111,394	244,312	32,451	8,727	2,750	41	3.7×10^{-4}
E5-CADETS	3,275,875	1,245,539	3,111,378	7,632,792	20,524	717,783	401,065	123	4.0×10^{-5}
E5-THEIA	745,773	234,896	747,452	1,728,121	162,714	61,368	9,374	69	9.2×10^{-5}
E5-CLEARSCOPE	171,771	3,842	150,725	326,338	48,488	8,636	1,020	51	3.4×10^{-4}

Evaluation: Baselines

We evaluate Orthrus against 5 state-of-the-art baselines from top-tier venues

- **SIGL** (USENIX Sec'21)
- **ThreaTrace** (IEEE TIFS'22)
- **Flash** (S&P'24)
- **Kairos** (S&P'24)
- **MAGIC** (USENIX Sec'24)

Details:

- All baselines are based on GNNs
- Most have been evaluated on the same datasets
- We reimplemented all baselines in a unified & open-source framework (PISMaker)

RQ1: Is Orthrus able to detect all attacks?

Dataset	System	E3	E5
CADETS	ORTHRUS	✓ 3/3	✓ 2/2
	Kairos	✗ 0/3	✗ 0/2
	ThreaTrace	✓ 3/3	✓ 2/2
	SIGL	✗ 0/3	✗ 0/2
	MAGIC	✓ 3/3	✓ 2/2
	Flash	✓ 3/3	✓ 2/2
THEIA	ORTHRUS	✓ 2/2	✓ 1/1
	Kairos	~ 1/2	✗ 0/1
	ThreaTrace	✓ 2/2	✓ 1/1
	SIGL	~ 1/2	✗ 0/1
	MAGIC	✓ 2/2	✓ 1/1
	Flash	✓ 2/2	✓ 1/1
CLEARSCOPE	ORTHRUS	✓ 1/1	✓ 3/3
	Kairos	✗ 0/1	~ 1/3
	ThreaTrace	✓ 1/1	✓ 3/3
	SIGL	✓ 1/1	~ 2/3
	MAGIC	✓ 1/1	✓ 3/3
	Flash	✗ 0/1	✓ 3/3

An attack is considered detected if the system flags any node directly involved in the attack as malicious

RQ2: What is the quality of attribution?

Dataset	System	TP	FP	TN	FN	Precision	MCC	Training Time	GPU Memory
E3-CADETS	ORTHRUS-full	25	23	268,062	43	0.52	**0.44**	**4min40**	**3.82GB**
	ORTHRUS-ano	10	0	268,085	58	**1.00**	0.38		
	Kairos	0	9	268,076	68	0.00	0.00	22min49	3.93GB
	Threatrace	61	252,117	15,968	7	0.00	0.00	28min28	5.22GB
	SIGL	0	80	268,005	68	0.00	0.00	4h48	10.07GB
	MAGIC	63	79,766	188,319	5	0.00	0.02	13h18	4.22GB
	Flash	13	2,381	265,704	55	0.01	0.03	10h33	19.18GB
E3-THEIA	ORTHRUS-full	48	11	699,166	70	0.81	**0.57**	**3min58**	**2.03GB**
	ORTHRUS-ano	8	0	699,177	110	**1.00**	0.26		
	Kairos	4	0	699,177	114	**1.00**	0.18	24min21	2.53GB
	Threatrace	88	671,883	27,294	30	0.00	-0.01	10min19	4.51GB
	SIGL	1	29	699,148	117	0.03	0.02	14h07	10.44GB
	MAGIC	115	394,906	304,271	3	0.00	0.01	11h39	5.35GB
	Flash	22	32,082	667,095	96	0.00	0.01	6h51	36.93GB
E3-CLEARSCOPE	ORTHRUS-full	2	6	111,347	39	0.25	**0.11**	**2min50**	**0.65GB**
	ORTHRUS-ano	1	1	111,352	40	**0.50**	**0.11**		
	Kairos	0	7	111,346	41	0.00	0.00	9min52	0.74GB
	Threatrace	41	87,501	23,852	0	0.00	0.01	3min55	4.90GB
	SIGL	1	11,372	99,981	40	0.00	0.00	1h01	9.71GB
	MAGIC	40	101,737	9,616	1	0.00	0.00	1h37	9.75GB
	Flash	0	15,137	96,216	41	0.00	-0.01	19h01	11.60GB

Matthews Correlation Coefficient (MCC):
$$\mathrm{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

RQ2: What is the quality of attribution?

Dataset	System	TP	FP	TN	FN	Precision	MCC	Training Time	GPU Memory
E3-CADETS	ORTHRUS-full	25	23	268,062	43	0.52	**0.44**	**4min40**	**3.82GB**
	ORTHRUS-ano	10	0	268,085	58	**1.00**	0.38		
	Kairos	0	9	268,076	68	0.00	0.00	22min49	3.93GB
	Threatrace	61	252,117	15,968	7	0.00	0.00	28min28	5.22GB
	SIGL	0	80	268,005	68	0.00	0.00	4h48	10.07GB
	MAGIC	63	79,766	188,319	5	0.00	0.02	13h18	4.22GB
	Flash	13	2,381	265,704	55	0.01	0.03	10h33	19.18GB
E3-THEIA	ORTHRUS-full	48	11	699,166	70	0.81	**0.57**	**3min58**	**2.03GB**
	ORTHRUS-ano	8	0	699,177	110	**1.00**	0.26		
	Kairos	4	0	699,177	114	**1.00**	0.18	24min21	2.53GB
	Threatrace	88	671,883	27,294	30	0.00	-0.01	10min19	4.51GB
	SIGL	1	29	699,148	117	0.03	0.02	14h07	10.44GB
	MAGIC	115	394,906	304,271	3	0.00	0.01	11h39	5.35GB
	Flash	22	32,082	667,095	96	0.00	0.01	6h51	36.93GB
E3-CLEARSCOPE	ORTHRUS-full	2	6	111,347	39	0.25	**0.11**	**2min50**	**0.65GB**
	ORTHRUS-ano	1	1	111,352	40	**0.50**	**0.11**		
	Kairos	0	7	111,346	41	0.00	0.00	9min52	0.74GB
	Threatrace	41	87,501	23,852	0	0.00	0.01	3min55	4.90GB
	SIGL	1	11,372	99,981	40	0.00	0.00	1h01	9.71GB
	MAGIC	40	101,737	9,616	1	0.00	0.00	1h37	9.75GB
	Flash	0	15,137	96,216	41	0.00	-0.01	19h01	11.60GB

Matthews Correlation Coefficient (MCC):

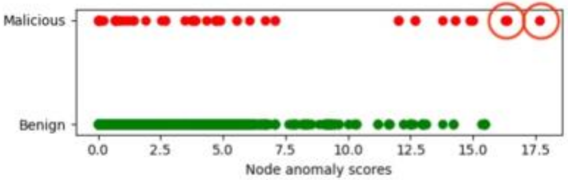
$$\mathrm{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

RQ2: What is the quality of attribution?

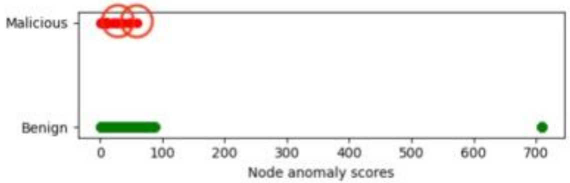
Dataset	System	TP	FP	TN	FN	Precision	MCC	Training Time	GPU Memory
E5-CADETS	ORTHRUS-full	2	10	3,111,245	121	**0.17**	**0.05**	**42min35**	21.10GB
	ORTHRUS-ano	1	5	3,111,250	122	**0.17**	0.04		
	Kairos	0	6	3,111,249	123	0.00	0.00	4h03	23.85GB
	Threatrace	91	3,104,018	7,237	32	0.00	-0.03	5h45	**17.31GB**
	SIGL	0	66	3,111,189	123	0.00	0.00	38h00	22.72GB
	MAGIC	123	3,110,714	541	0	0.00	0.00	77h13	79.36GB
	Flash	45	33,941	3,077,314	78	0.00	0.08	101h26	80.19GB
E5-THEIA	ORTHRUS-full	13	2	747,381	56	0.87	0.4	**14min30**	4.23GB
	ORTHRUS-ano	2	0	747,383	67	**1.00**	**0.17**		
	Kairos	0	2	747,381	69	0.00	0.00	1h02	**4.16GB**
	Threatrace	66	739,322	8,061	3	0.00	0.00	2h51	11.59GB
	SIGL	0	23	747,360	69	0.00	0.00	40h20	24.44GB
	MAGIC	1	296,554	450,829	68	0.00	-0.01	13h21	16.95GB
	Flash	43	295,729	451,654	26	0.00	0.00	47h50	80.18GB
E5-CLEARSCOPE	ORTHRUS-full	4	8	150,666	47	**0.33**	**0.16**	**22min19**	**1.72GB**
	ORTHRUS-ano	2	7	150,667	49	0.22	0.09		
	Kairos	1	3	150,671	50	0.25	0.07	1h06	2.26GB
	Threatrace	41	142,487	8,187	10	0.00	-0.01	44min53	5.94GB
	SIGL	10	63	150,610	41	0.14	0.16	82h50	16.38GB
	MAGIC	51	139,385	11,289	0	0.00	0.01	11h39	48.24GB
	Flash	15	4,552	146,122	36	0.00	0.03	25h34	11.60GB

Matthews Correlation Coefficient (MCC): $$\mathrm{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

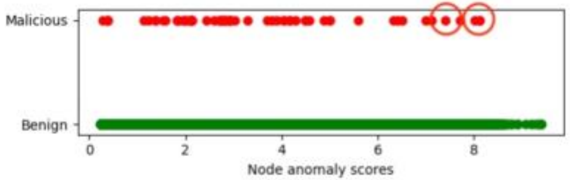
RQ2: What is the quality of attribution?



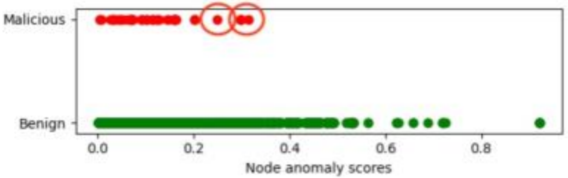
(a) ORTHRUS.



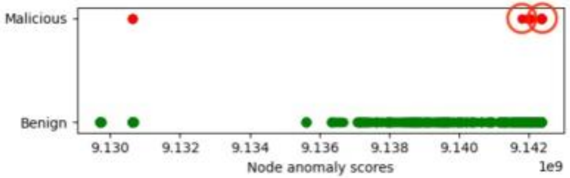
(b) Threatrace.



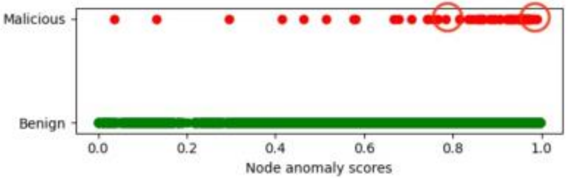
(c) Kairos.



(d) SIGL.



(e) Magic.



(f) Flash.

- State-of-the-art systems struggle in distinguish malicious and benign nodes in term of anomaly score
- Our model learns better the deviation between benign and malicious patterns of nodes

RQ3: Is Orthrus computationally efficient?

Dataset	System	TP	FP	TN	FN	Precision	MCC	Training Time	GPU Memory
E3-CADETS	ORTHRUS-full	25	23	268,062	43	0.52	**0.44**	**4min40**	**3.82GB**
	ORTHRUS-ano	10	0	268,085	58	**1.00**	0.38		
	Kairos	0	9	268,076	68	0.00	0.00	22min49	3.93GB
	Threatrace	61	252,117	15,968	7	0.00	0.00	28min28	5.22GB
	SIGL	0	80	268,005	68	0.00	0.00	4h48	10.07GB
	MAGIC	63	79,766	188,319	5	0.00	0.02	13h18	4.22GB
	Flash	13	2,381	265,704	55	0.01	0.03	10h33	19.18GB
E3-THEIA	ORTHRUS-full	48	11	699,166	70	0.81	**0.57**	**3min58**	**2.03GB**
	ORTHRUS-ano	8	0	699,177	110	**1.00**	0.26		
	Kairos	4	0	699,177	114	**1.00**	0.18	24min21	2.53GB
	Threatrace	88	671,883	27,294	30	0.00	-0.01	10min19	4.51GB
	SIGL	1	29	699,148	117	0.03	0.02	14h07	10.44GB
	MAGIC	115	394,906	304,271	3	0.00	0.01	11h39	5.35GB
	Flash	22	32,082	667,095	96	0.00	0.01	6h51	36.93GB
E3-CLEARSCOPE	ORTHRUS-full	2	6	111,347	39	0.25	**0.11**	**2min50**	**0.65GB**
	ORTHRUS-ano	1	1	111,352	40	**0.50**	**0.11**		
	Kairos	0	7	111,346	41	0.00	0.00	9min52	0.74GB
	Threatrace	41	87,501	23,852	0	0.00	0.01	3min55	4.90GB
	SIGL	1	11,372	99,981	40	0.00	0.00	1h01	9.71GB
	MAGIC	40	101,737	9,616	1	0.00	0.00	1h37	9.75GB
	Flash	0	15,137	96,216	41	0.00	-0.01	19h01	11.60GB

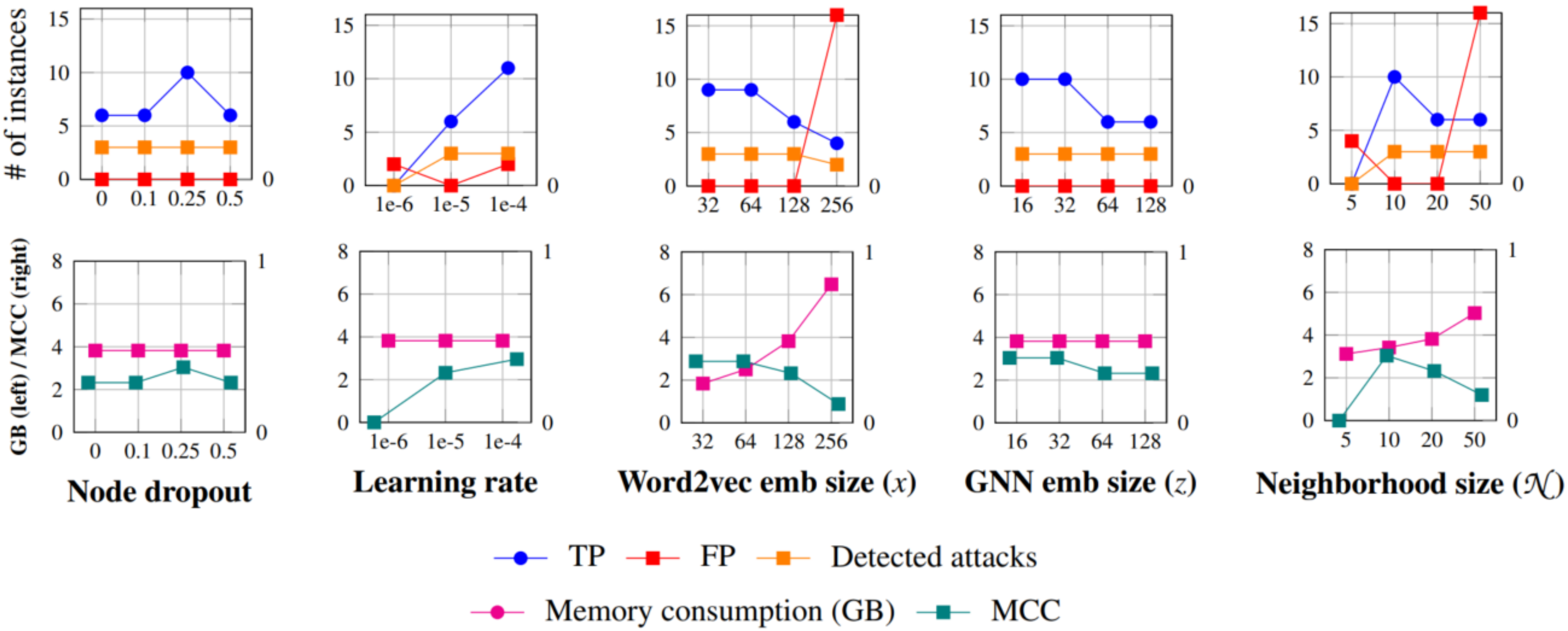
- Orthrus is the most computationally efficient system on most datasets

RQ3: Is Orthrus computationally efficient?

Dataset	System	TP	FP	TN	FN	Precision	MCC	Training Time	GPU Memory
E5-CADETS	ORTHRUS-full	2	10	3,111,245	121	**0.17**	**0.05**	**42min35**	21.10GB
	ORTHRUS-ano	1	5	3,111,250	122	**0.17**	0.04		
	Kairos	0	6	3,111,249	123	0.00	0.00	4h03	23.85GB
	Threatrace	91	3,104,018	7,237	32	0.00	-0.03	5h45	**17.31GB**
	SIGL	0	66	3,111,189	123	0.00	0.00	38h00	22.72GB
	MAGIC	123	3,110,714	541	0	0.00	0.00	77h13	79.36GB
	Flash	45	33,941	3,077,314	78	0.00	0.08	101h26	80.19GB
E5-THEIA	ORTHRUS-full	13	2	747,381	56	0.87	0.4	**14min30**	4.23GB
	ORTHRUS-ano	2	0	747,383	67	**1.00**	**0.17**		
	Kairos	0	2	747,381	69	0.00	0.00	1h02	**4.16GB**
	Threatrace	66	739,322	8,061	3	0.00	0.00	2h51	11.59GB
	SIGL	0	23	747,360	69	0.00	0.00	40h20	24.44GB
	MAGIC	1	296,554	450,829	68	0.00	-0.01	13h21	16.95GB
	Flash	43	295,729	451,654	26	0.00	0.00	47h50	80.18GB
E5-CLEARSCOPE	ORTHRUS-full	4	8	150,666	47	**0.33**	**0.16**	**22min19**	**1.72GB**
	ORTHRUS-ano	2	7	150,667	49	0.22	0.09		
	Kairos	1	3	150,671	50	0.25	0.07	1h06	2.26GB
	Threatrace	41	142,487	8,187	10	0.00	-0.01	44min53	5.94GB
	SIGL	10	63	150,610	41	0.14	0.16	82h50	16.38GB
	MAGIC	51	139,385	11,289	0	0.00	0.01	11h39	48.24GB
	Flash	15	4,552	146,122	36	0.00	0.03	25h34	11.60GB

- Orthrus is the most computationally efficient system on most datasets

RQ4:How do hyperparameters influence performance?



- Orthrus can detect all 3 attacks of E3-CADETS dataset, even if the parameter changes a lot
- Results demonstrate the robustness of Orthrus

RQ5:How the different Orthrus components contribute to overall performance?

Ablation Study: we replace or remove one component at a time.

Component	With Component (✓)	Without Component (✗)
Featurization	ORTHRUS' Word2vec embedding (§4.2)	Hierarchical hashing as in Kairos
Encoding	ORTHRUS' encoder (§4.3)	Kairos' TGN encoder
Clustering	ORTHRUS' anomaly detection algorithm (§4.4)	Automatic anomaly thresholding only
Reconstruction	ORTHRUS' attack reconstruction algorithm (§4.5)	No tracing algorithm used

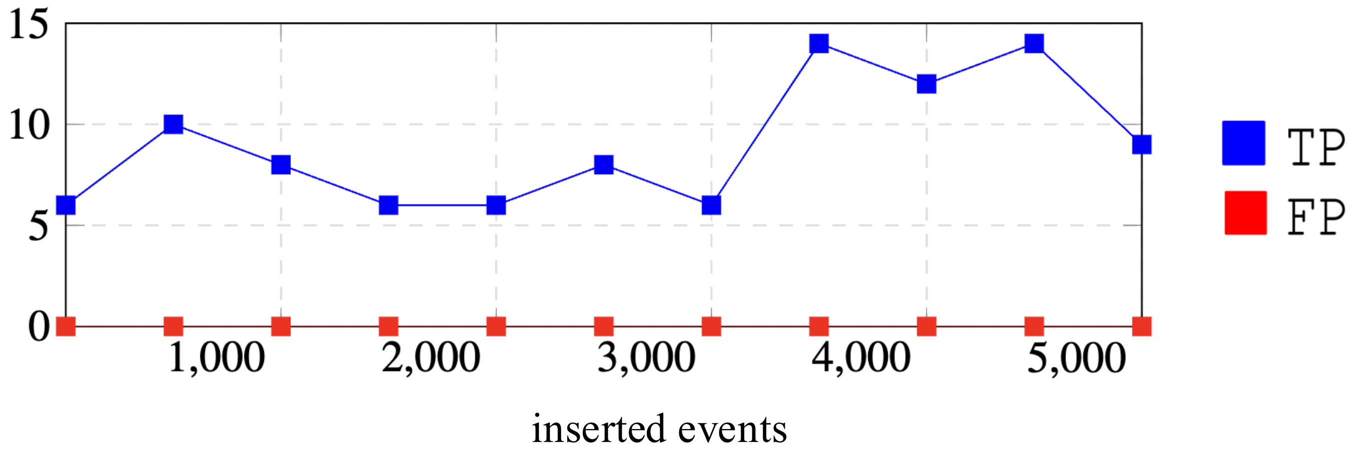
RQ5:How the different Orthrus components contribute to overall performance?

Dataset	**Featurization**	**Encoding**	**Clustering**	**Reconstruction**	TP	FP	Precision	Memory
E3-THEIA	✗	✓	✓	✓	51	13	0.79	2.03GB
	✓	✗	✓	✓	41	772	0.05	5.75GB
	✓	✓	✗	✓	48	11	0.81	2.03GB
	✓	✓	✓	✗	8	0	1.00	2.03GB
	✓	✓	✓	✓	48	11	0.81	2.03GB
E5-THEIA	✗	✓	✓	✓	0	155	0.00	4.23GB
	✓	✗	✓	✓	13	53	0.20	11.10GB
	✓	✓	✗	✓	20	11,420	0.00	4.23GB
	✓	✓	✓	✗	2	0	1.00	4.23GB
	✓	✓	✓	✓	13	2	0.87	4.23GB

The darker the precision, the more important the component is.

RQ6:How robust Orthrus is against adversarial attacks?

- We simulate a mimicry attack
- The attacker inserts benign events surrounding the attack to evade detection



Future work

1. **Impact of the capture mechanism**: address the impact of the capture mechanism of provenance data and make the system more universal
2. **Training time**: it is important to reduce training time consumption because PIDSes need to be re-trained regularly to address concept drift

PIDSMaker

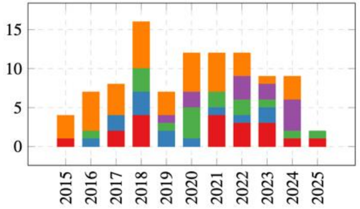
As part of our second USENIX Sec'25 paper, **we open-sourced our framework** to build PIDSs based on deep learning and GNN architectures 🎉

Sometimes Simpler is Better: A Comprehensive Analysis of State-of-the-Art Provenance-Based Intrusion Detection Systems

Tristan Bilot[1 2 3†], Baoxiang Jiang[4†], Zefeng Li[5], Nour El Madhoun[2], Khaldoun Al Agha[1], Anis Zouaoui[3], Thomas Pasquier[5]
[1]*Université Paris-Saclay,* [2]*LISITE, Isep,* [3]*Iriguard,*
[4]*Xi'an Jiaotong University,* [5]*University of British Columbia*

Abstract

Provenance-based intrusion detection systems (PIDSs) have garnered significant attention from the research community over the past decade. Although recent studies report near-perfect detection performance, we show that these systems are not viable for practical deployment. We implemented eight state-of-the-art systems within a unified framework and identified nine key shortcomings that hinder their practical adoption. Through extensive experiments, we quantify the impact of these shortcomings using cybersecurity-oriented



README | Apache-2.0 license

🥷 PIDSMaker

docs online | DOI 10.5281/zenodo.15603122 | license Apache-2.0

Paper | Documentation | Installation

The first framework designed to build and experiment with provenance-based intrusion detection systems (PIDSs) using deep learning architectures. It provides a single codebase to run most recent state-of-the-arts systems and easily customize them to develop new variants.

Currently supported PIDSs:

- Velox (USENIX Sec'25): Sometimes Simpler is Better: A Comprehensive Analysis of State-of-the-Art Provenance-Based Intrusion Detection Systems
- Orthrus (USENIX Sec'25): ORTHRUS: Achieving High Quality of Attribution in Provenance-based Intrusion Detection Systems
- R-Caid (IEEE S&P'24): R-CAID: Embedding Root Cause Analysis within Provenance-based Intrusion Detection
- Flash (IEEE S&P'24): Flash: A Comprehensive Approach to Intrusion Detection via Provenance Graph Representation Learning
- Kairos (IEEE S&P'24): Kairos: Practical Intrusion Detection and Investigation using Whole-system Provenance
- Magic (USENIX Sec'24): MAGIC: Detecting Advanced Persistent Threats via Masked Graph Representation Learning
- NodLink (NDSS'24): NODLINK: An Online System for Fine-Grained APT Attack Detection and Investigation
- ThreaTrace (IEEE TIFS'22): THREATRACE: Detecting and Tracing Host-Based Threats in Node Level Through Provenance Graph Learning

PIDSMaker

- 8 systems integrated

Supported PIDSs

- Velox (USENIX Sec'25): Sometimes Simpler is Better: A Comprehensive Analysis of State-of-the-Art Provenance-Based Intrusion Detection Systems
- Orthrus (USENIX Sec'25): ORTHRUS: Achieving High Quality of Attribution in Provenance-based Intrusion Detection Systems
- R-Caid (IEEE S&P'24): R-CAID: Embedding Root Cause Analysis within Provenance-based Intrusion Detection
- Flash (IEEE S&P'24): Flash: A Comprehensive Approach to Intrusion Detection via Provenance Graph Representation Learning
- Kairos (IEEE S&P'24): Kairos: Practical Intrusion Detection and Investigation using Whole-system Provenance
- Magic (USENIX Sec'24): MAGIC: Detecting Advanced Persistent Threats via Masked Graph Representation Learning
- NodLink (NDSS'24): NODLINK: An Online System for Fine-Grained APT Attack Detection and Investigation
- ThreaTrace (IEEE TIFS'22): THREATRACE: Detecting and Tracing Host-Based Threats in Node Level Through Provenance Graph Learning

PIDSMaker

- 8 systems integrated
- 9 datasets

Dataset	Compressed (GB)	Uncompressed (GB)
CLEARSCOPE_E3	0.6	4.8
CADETS_E3	1.4	10.1
THEIA_E3	1.1	12
CLEARSCOPE_E5	6.2	49
CADETS_E5	36	276
THEIA_E5	5.8	36
OPTC_H051	1.7	7.7
OPTC_H_501	1.5	6.7
OPTC_H201	2	9.1

Supported PIDSs

- Velox (USENIX Sec'25): Sometimes Simpler is Better: A Comprehensive Analysis of State-of-the-Art Provenance-Based Intrusion Detection Systems
- Orthrus (USENIX Sec'25): ORTHRUS: Achieving High Quality of Attribution in Provenance-based Intrusion Detection Systems
- R-Caid (IEEE S&P'24): R-CAID: Embedding Root Cause Analysis within Provenance-based Intrusion Detection
- Flash (IEEE S&P'24): Flash: A Comprehensive Approach to Intrusion Detection via Provenance Graph Representation Learning
- Kairos (IEEE S&P'24): Kairos: Practical Intrusion Detection and Investigation using Whole-system Provenance
- Magic (USENIX Sec'24): MAGIC: Detecting Advanced Persistent Threats via Masked Graph Representation Learning
- NodLink (NDSS'24): NODLINK: An Online System for Fine-Grained APT Attack Detection and Investigation
- ThreaTrace (IEEE TIFS'22): THREATRACE: Detecting and Tracing Host-Based Threats in Node Level Through Provenance Graph Learning

PIDSMaker

- Hands-on tutorial to create new PIDSs for research

PIDSMaker Documentation

Search

GitHub
usenixsec ☆6 0

Tutorial

In this tutorial, we craft a brand new architecture using existing components in the framework and evaluate it on node-level intrusion detection.

Architecture

Our goal is to implement a new system that satisfies the following requirements:

- Compute text embeddings from the textual attributes of entities, such as file paths, process commands, and socket IP addresses.
- Learn behavior-specific representations for both source and destination nodes.
- Leverage GraphSAGE layers to capture structural patterns within the provenance graph.
- Use the node embeddings generated by the encoder as input to a two-layer MLP decoder, training the model in a self-supervised manner to predict edge types—following an approach similar to that of the Kairos and Orthrus systems.
- In the final step, classify nodes as malicious if their predicted score exceeds a threshold, defined as the maximum loss observed on the validation set.

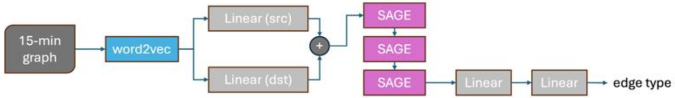


Table of contents

PIDSMaker

Support:

- Hyperparameter tuning
- Custom graph batching
- Instability/uncertainty measurement

Hyperparameter Tuning

PIDSMaker simplifies hyperparameter tuning by combining its efficient pipeline design with the power of W&B Sweeps.

Dataset-specific tuning

Tuning is configured using YAML files, just like system definitions. For example, suppose you've created a new system named `my_system`, and its configuration is stored in `config/my_system.yml`. To search for optimal hyperparameters on the `THEIA_E3` dataset, you can create a new tuning configuration file at `config/experiments/tuning/systems/theia_e3/tuning_my_system.yml` following the W&B syntax:

tuning_my_system.yml

```
method: grid

parameters:
  detection.gnn_training.lr:
    values: [0.001, 0.0001]
  detection.gnn_training.node_hid_dim:
    values: [32, 64, 128, 256]
  featurization.feat_training.used_method:
    values: [fasttext, word2vec]
```

Batching & Sampling

Batching

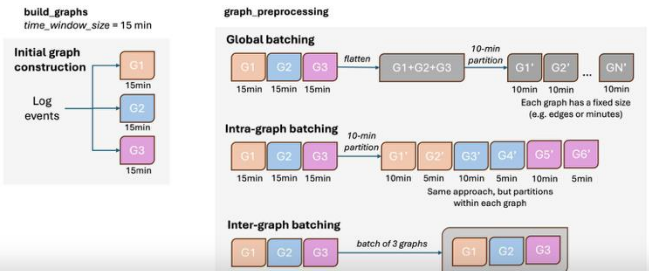
Batching refers to grouping edges, nodes, or graphs into a temporal graph provided as input to the model.

We provide three batching strategies that can be configured via dedicated batching arguments.

Global Batching: takes as input a large flattened graph comprising all events in the dataset and partitions it into equal-size graphs based on number of edges, minutes, or similar.

Intra-graph Batching: applies similar batching as global batching but within each built graph.

Inter-graph Batching: groups multiple graphs into a single batch. This batch is a large graph where all graphs are stacked together without any overlap, following the mini-batching strategy from PyG.



Instability

Measure instability across multiple iterations

Most systems are prone to instability, with some runs reaching high performance, while others fail dramatically. To quantify this instability, we run the system multiple times and compute the mean and standard deviation of key performance metrics. This can be done easily by using the `--experiment=run_n_times` tag:

```
./run.sh orthrus CADETS_E3 --tuned --experiment=run_n_times
```

PIDSMaker

Repo: https://github.com/ubc-provenance/PIDSMaker

The End

Merci!