Accelerating Provenance-based Intrusion Detection Research with PIDSMaker

Tristan Bilot

University of British Columbia

NDSS PRISM Workshop Tutorial · 2026

Outline

01 **Background**
Typical PIDS architecture

02 **The Problem**
Why PIDS evaluation is painful

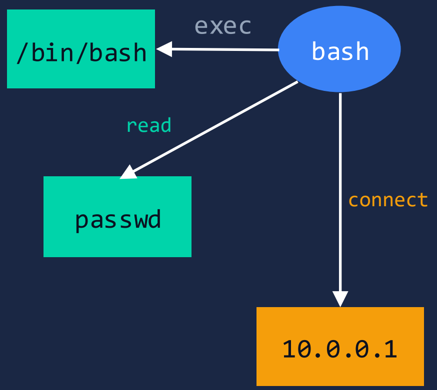
03 **PIDSMaker**
Architecture, datasets & features

04 **Live Demo**
Analyzing outputs

Provenance-based Intrusion Detection (PIDS)

Detect APT attacks by applying self-supervised graph learning to provenance graphs

Provenance Graph



→

PIDS

Self-supervised
Graph Learning

Encoder-Decoder

Train on benign only
High error → anomaly

→

Detection (e.g. APT)

Node-level

Flag malicious processes, files

Edge-level

Flag suspicious interactions

Graph-level

Flag entire attack subgraphs

Note: APT and zero-day attacks struggle to be detected with traditional rule/signature methods.

The PIDS Research Workflow is Painful

When working on a new PIDS, most researchers have to...

Preprocess huge datasets
DARPA TC/OpTC requires massive memory (100–500 GB) and custom pipelines

Implement system from scratch
Build their own training, inference, and evaluation code

Tune hyperparameters
Learning rates, hidden dims, thresholds — often done inconsistently across systems

Re-implement baselines
Reproduce other systems for comparison, often with missing code or details

Run ablation studies
Manually test each component variant, with no shared infrastructure

Inconsistent Evaluation Undermines Progress

Preprocessing

Different graph construction and features — even on the same dataset

Ground-truth Labels

Neighborhood vs. descendant vs. node/edge labeling — inflates or depresses metrics

Dataset Splits

No standard temporal boundaries, host selection, or attack mix

Detection Granularity

Node vs. edge vs. graph-level — coarser = inflated metrics, less actionable

"The study reveals that no system is fully reproducible end-to-end, with flaws including missing components, configuration issues, and undocumented behaviors." — Abrar et al., REP'25

PIDSMaker

A unified framework addressing these challenges

❌ Re-implement baselines	→	✅ 8 systems already integrated, run in one command
❌ Implement from scratch	→	✅ Modular pipeline with reusable components
❌ Preprocess huge datasets	→	✅ Pre-built database dumps, ready to query
❌ Tune hyperparameters	→	✅ Built-in grid search
❌ Run ablation studies	→	✅ Native support for component ablation

github.com/ubc-provenance/PIDSMaker

8 Integrated Detection Systems

We selected recent ML-based PIDSs published at top-tier security venues (2021–2025)

System	Venue	Year
ThreaTrace	TIFS	2021
NodLink	NDSS	2024
MAGIC	USENIX Sec	2024
Kairos	IEEE S&P	2024
Flash	IEEE S&P	2024
R-CAID	IEEE S&P	2024
Orthrus	USENIX Sec	2025
Velox	USENIX Sec	2025

Selection criteria

- Top-tier security venues
- Published 2021–2025
- PIDSs for APT detection
- Similar architecture

Reimplementation

- Reimplemented based on official code, except R-Caid (not open-source)

→ This common structure enables a unified framework

Shared Architecture, Different Components

All 8 systems are modularized into components

	Features	Transform.	Text Emb.	Encoder	Decoder	Objective	Threshold
ThreaTrace	type+distrib	–	–	SAGE	–	Node Type	Fixed
NodLink	cmd,path,IP	Undirected	FastText	Weighted Sum	VAE+MLP	Node Recon	Val Thresh
MAGIC	node+edge	No redund.	–	GAT	GAT+MLP	Masked Rec	K-D Tree
Kairos	path,IP+port	–	HFH	TGN+Attn	MLP	Edge Type	Fixed
Flash	cmd,path,IP	–	W2V+Pos	SAGE	XGBoost	Node Type	Fixed
R-CAID	path+name	Pseudo-graph	Doc2Vec	GAT	–	Node Type	K-Means
Orthrus	all attrs	–	Word2Vec	TGN-Light	MLP	Edge Type	Val+KMeans
Velox	all attrs	–	Word2Vec	Linear	MLP	Edge Type	Val Thresh

PIDSMaker makes these components interchangeable: pick any component via YAML to create new system variants
You can add your own components!

Modular Pipeline Architecture

7-stages pipeline with automatic restart (each stage writes its output to disk)

Construction →	Transformation →	Featurization →	Batching →	Training →	Evaluation →	Triage
Type File Path Process Path Process Cmd Socket IP Socket Port	Undirected DAG R-CAID Pseudo None	Word2Vec Doc2Vec FastText ALaCarte HFH Only Type	Global Intra-graph Inter-graph	<u>Encoders</u> TGN, SAGE GAT, GIN R-CAID, MAGIC … <u>Decoders</u> MLP, GAT Inner product … <u>Objectives</u> Node Type Edge Type Node Recon Masked Recon.	Max Val Loss Mean Val Loss ThreaTrace MAGIC Flash, NodLink Node/Edge lvl	DepImpact None

On-disk caching: each stage's output is stored on disk, using a unique hash to enable automatic restart.

Standardized Datasets & Ground Truth

DARPA TC E3

5 datasets

CADETS, THEIA, ClearScope, FiveDirections, Trace

DARPA TC E5

5 datasets

CADETS, THEIA, ClearScope, FiveDirections, Trace

DARPA OpTC

3 datasets (hosts)

H051, H201, H501 (Windows)

⚠ Systems use different detection tasks (neighborhoods, ancestors, nodes) → comparison impossible.

Our approach: consistent node-level labels

- Most granular detection level — reduces analyst workload
- Following standardized labels published in Orthrus (USENIX Sec’25)
- You can add your own ground-truth!

13 Preprocessed Datasets

All available as PostgreSQL dumps — no manual preprocessing required

Dataset	OS	Attacks	Size (GB)
CADETS_E3	FreeBSD	3	10
THEIA_E3	Linux	2	12
CLEARSCOPE_E3	Android	1	4.8
FIVEDIRECTIONS_E3	Linux	2	22
TRACE_E3	Linux	3	100
CADETS_E5	FreeBSD	2	276
THEIA_E5	Linux	1	36
CLEARSCOPE_E5	Android	2	49
FIVEDIRECTIONS_E5	Linux	4	280
TRACE_E5	Linux	1	710
optc_h201	Windows	1	9
optc_h501	Windows	1	6.7
optc_h051	Windows	1	7.7

DARPA TC E3

5 hosts · 11 attacks
~149 GB total

DARPA TC E5

5 hosts · 10 attacks
~1.35 TB total

DARPA OpTC

3 hosts · 3 attacks
~23 GB total

Installation & Architecture

Two Docker containers — plug in your GPU and go

PostgreSQL Container

- Stores preprocessed provenance data
- Loads provided database dumps
- DARPA TC E3/E5 + OpTC
- No manual preprocessing needed

← data →

PIDSMaker Container

- Framework entrypoint
- Python env + full pipeline
- PyTorch + GPU-connected (CUDA)
- Reads data from PostgreSQL

```
$ docker compose up  →  ready to run experiments
```

PIDSMaker Features

YAML Configuration

Declare a complete PIDS without writing code.

Rapid Prototyping

Mix & match components across systems.

Hyperparameter Tuning

Parallelizable grid search tuning.

Ablation Studies

Search for the best components.

Instability Measurement

N-run aggregation: mean, std, std_rel.

Metrics & Visualization

30+ metrics, score distributions, logging.

Each feature explained →

YAML-driven Configuration

orthrus.yml

```
construction:
  node_features:
    subject: type, path, cmd_line
    file: type, path
    netflow: type, remote_ip
featurization:
  used_method: word2vec
  emb_dim: 128
training:
  lr: 0.00001
  encoder:
    used_methods: tgn, graph_attention
evaluation:
  threshold_method: max_val_loss
triage:
  used_method: depimpact
```

```
$ ./run.sh orthrus CADETS_E3
```

No code required

CLI overrides

--training.lr=0.0001 overrides YAML

Automatic restart

Starts the pipeline at the necessary stage

Logging on Weights & Biases (W&B)

Metrics, figures, resources in real time

Rapid Prototyping: Mix & Match

custom_system.yml

```
_include_yml: orthrus

featurization:
  used_method: fasttext

training:
  encoder:
    used_methods: sage
    sage:
      activation: relu
      num_layers: 2
```

```
$ ./run.sh custom_system CADETS_E3
```

What this does:

INHERITED

construction, transformation, batching, evaluation, triage

OVERRIDDEN

FastText featurization (was Word2Vec)

OVERRIDDEN

GraphSAGE encoder (was TGN)

Enables systematic design-space exploration

Hyperparameter Tuning

1. Define a search space

```
method: grid
parameters:
  training.lr:
    values: [0.001, 0.0001]
  training.node_hid_dim:
    values: [32, 64, 128, 256]
```

Searches through all possible combinations.

2. Run the search

```
$ ./run.sh SYSTEM DATASET \
    --tuning_mode=hyperparameters
```

Parallelizable across GPUs and machines.
All results tracked in W&B.

Pick best config from W&B

3. Re-run with best hyperparameters

After tuning, copy best hyperparameters to a file and use --tuned :

```
$ ./run.sh orthrus CADETS_E3 --tuned
```

Ablation Studies

1. Define ablations

```
method: grid
parameters:
  featurization.used_method:
    values: [only_ones, word2vec]
  training.encoder.used_methods:
    values: [none, gin]
```

Tries all variants.

2. Run the experiments

```
$ ./run.sh SYSTEM DATASET \
    --tuning_mode=hyperparameters
    --tuning_file_path=ablation.yml
```

Same mechanism as tuning.

Fair: same protocol applied to ALL systems

3. Compare results across configurations

Adding ablation tables is recommended:

Featurizer	Encoder	Prec.	Rec.	AP
only_ones	none	...	...	...
word2vec	gin	best	best	best
...	...	...	...	...

→ Identify which component matters most

Instability Measurement

Neural networks exhibit run-to-run variability, yet most PIDS studies report single-run results.

One command, N runs

```
$ ./run.sh orthrus CADETS_E3 \
    --experiment=run_n_times
```

Runs pipeline N times (default 5), aggregates all metrics.

For every metric, reports:

`*_mean`	Average across N runs
`*_std`	Standard deviation
`*_std_rel`	Relative std ($\sigma/\mu \times 100$)

Example output (Precision, 5 runs):

precision_mean	precision_std	precision_std_rel
0.847	0.032	3.78%

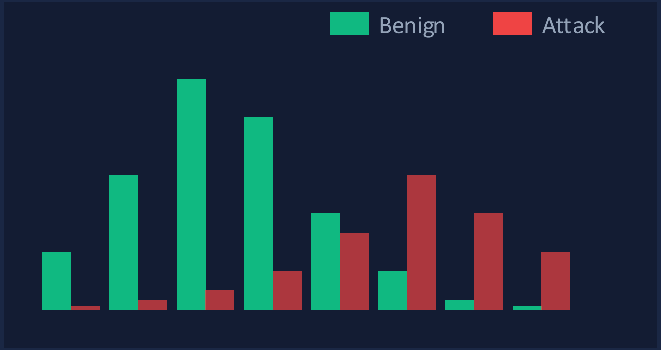
$$\tilde{\sigma} = (\sigma / \mu) \times 100$$

Relative standard deviation formula

Visualization & Metrics

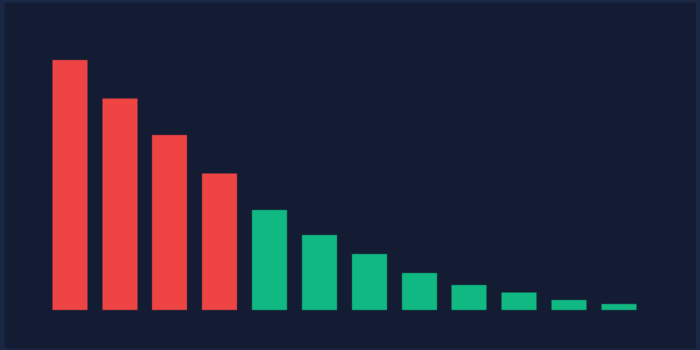
Anomaly Score Distribution

Separation between benign and attack.



Top-Ranked TPs and FPs

For comprehensive analysis of predicted nodes.



30+ metrics per epoch: Precision, Recall, F1, AUC-ROC, AP, ADP, Discrimination Curve... All in W&B.

PIDSMaker is Open Source

We want to grow this with the community

Add your system

Integrate your PIDS into the pipeline via YAML + a few Python modules

Add new datasets

Contribute preprocessed database dumps for new benchmarks

Add components

New encoders, decoders, featurizers, or evaluation methods

Report & compare

Run experiments, share results, help build reproducible baselines

github.com/ubc-provenance/PIDSMaker

tbilot@cs.ubc.ca