FLASH: A Comprehensive Approach to Intrusion Detection via Provenance Graph Representation Learning

Mati Ur Rehman University of Virginia
Hadi Ahmadi Corvic Inc.
Wajih Ul Hassan University of Virginia

2024 IEEE Symposium on Security and Privacy (SP)

Outline

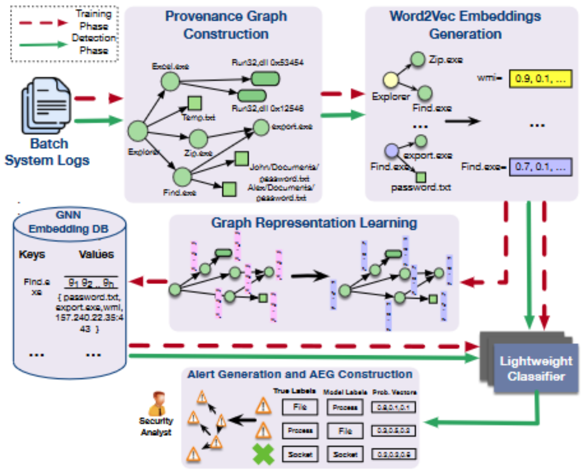
- Introduction
- Limitation
- FLASH Design
- Evaluation
- Discussion

Introduction

Challenges of Existing GNN techniques

- Lack of scalability
- Slow detection speed
- Temporal & Causal Ordering Disregard
- Semantic Information Neglect

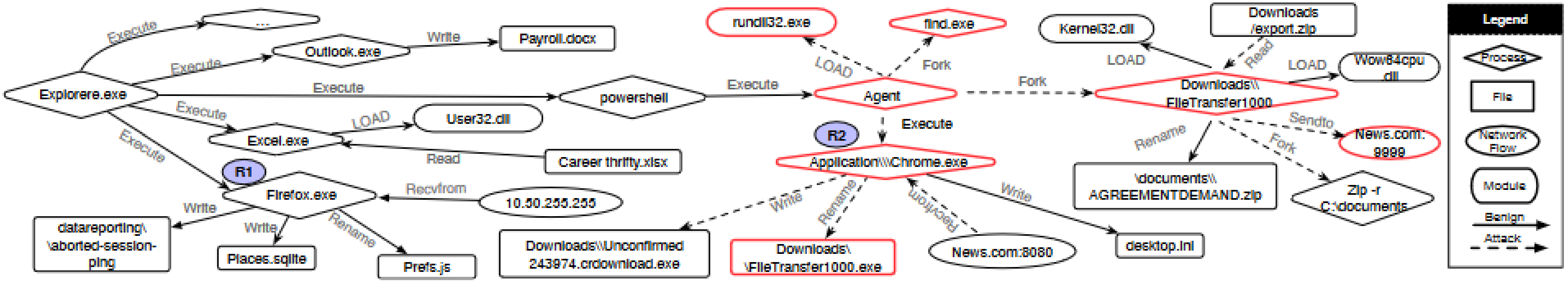
GNN + Word2Vec



Contributions

- Superior detection performance
- Enhance the ability of IDSes to combat mimicry attack
- Implement scalability

Limitation



Problem:

- Semantic Information Neglect
- Temporal & Causal Ordering Disregard
- Scalability Challenges
- Coarse-grained Detection
- Contextual Alerts
- Robustness Against Mimicry Attacks

Comparison of existing GNN-IDSes

	Semantic Encoding	Temporal Encoding	Scalable	Detection Granularity	Contextual Alerts	Robust Against Mimicry Attacks
FLASH	✓	✓	✓	**Node**	✓	✓
ThreaTrace [67]	X	X	X	Node	✓	✓
Unicorn [30]	X	✓	✓	Graph	X	X
ProvDetector [66]	X	X	X	Graph	X	X
StreamSpot [51]	X	X	✓	Graph	X	X
ProGrapher [69]	X	✓	✓	Graph	X	-
ShadeWatcher [71]	✓	X	X	Edge	✓	-

FLASH Design

FLASH composed of five modules

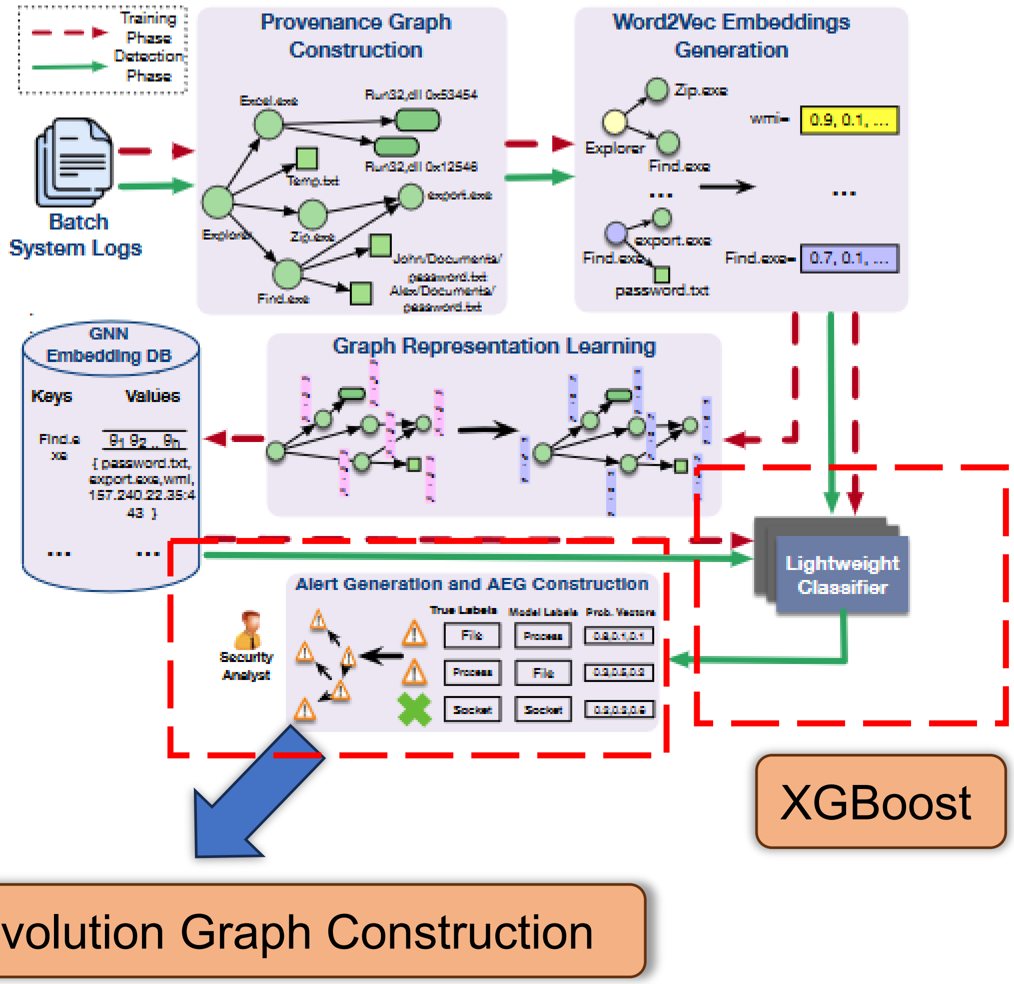
- Provenance graph constructor
- Word2Vec-based semantic encoder
- GNN-based contextual encoder
- Embedding database
- Anomaly detector

Algorithm 1: NodeSentenceEmbeddings

```
Inputs : Provenance Graph G;
         Trained Word2Vec Model w2v;
Output: Array V of sentence encoding.
D ← list([])
/* Iterating over neighbors of the node                    */
foreach N ∈ G do
    /* Getting the syscall performed on this node
       */
    A = GetActionPerformed(N)
    /* Getting node properties like process name,
       file path, command line, etc.                        */
    S = GetNodeAttributes(N)
    /* Concatenating the words into a list                  */
    D.append(A)
    D.append(S)
end
/* Initializing the embedding vector                        */
V ← list([])
/* Iterating over words of document D                       */
foreach w ∈ D do
    /* Getting Word2Vec embeddings for this word */
    E = w2v(w)
    V.append(E)
end
/* Giving weight to each index of the vector V to
   capture the temporal order of system events              */
P = GetPositionalEcondingVector( len(V) )
V ← V + P
/* Averaging the embeddings for all words to get
   one vector for the complete sentence.                    */
V ← V.mean()
return V
```

Algorithm 2: Attack Evolution Graph Generation

```
Inputs : Graph G(V, E); Alerts N; Hop length h
Output: AEG Graphs List IG
List_G ← []
foreach Alert n ∈ N do
    Paths ← GetCausalPaths(n, h)
    // List to store paths containing alert nodes
    AttackPaths ← []
    foreach P ∈ Paths do
        AlertNodes ← P ∩ N
        if len(AlertNodes) > 1 then
            CompactPath ← KeepAlertNodesOnly(P)
            AttackPaths ← AttackPaths + CompactPath
        end
    end
    List_G.append(AttackPaths)
end
IG ← []
foreach Pathlist ∈ List_G do
    // Connect all paths originating from an alert
       node n to construct a graph.
    AEGraph ← ConvertToGraph(Pathlist)
    IG ← IG + AEGraph
end
return IG
```



Evaluation

- RQ1. How does FLASH detection accuracy compare to the existing systems?
- RQ2. How does FLASH's GNN optimizations enhances the performance?
- RQ3. How does the batch size parameter affect FLASH's performance, accuracy, and resource usage?
- RQ4. How robust is FLASH against mimicry attacks?
- RQ5. What are the results of the ablation study on various FLASH components and hyperparameters?
- RQ6. How effectively does FLASH assist in the alert validation process?

benchmarks: ThreaTrace and Unicorn .

Platform:
a machine equipped with 8 Intel vCPUs, 80 GB RAM, an NVIDIA RTX2080 GPU, and Ubuntu 18.04.6 LTS.

Batch size:
event batch size of 250k

Evaluation

RQ1. Detection Performance

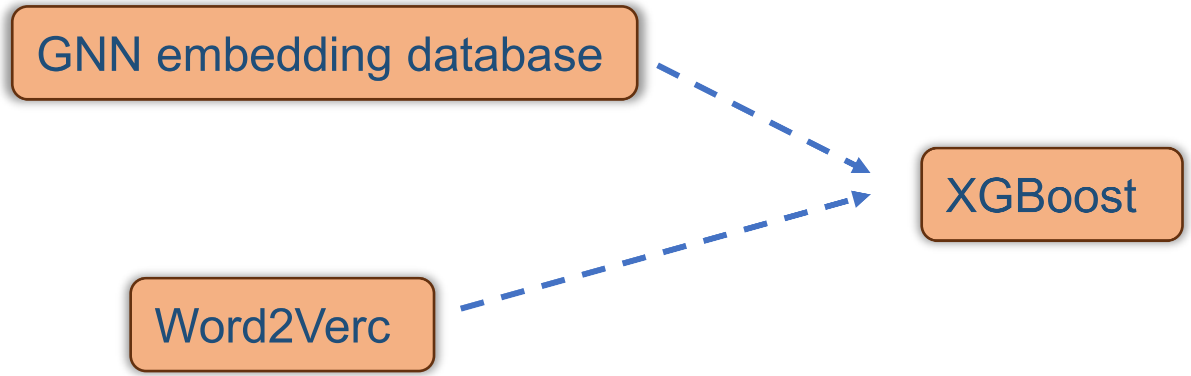
TABLE 2: Comparison of FLASH against ThreaTrace using only the GNN as the anomaly detector and using a GNN embeddings database along with a lightweight classifier. Prec.: Precision; Rec.: Recall;

Datasets	ThreaTrace				FLASH (GNN)				FLASH (GNN + XGBoost)			
	Prec.	Rec.	F-Score	TP/ FP/ FN/ TN	Prec.	Rec.	F-Score	TP/ FP/ FN/ TN	Prec.	Rec.	F-Score	TP/ FP/ FN/ TN
Cadets (E3)	0.90	0.99	0.95	12848/ 1361/ 4/ 705,605	0.94	0.99	0.96	12851/ 818/ 1/ 706,148	0.95	0.99	0.97	12851/ 720/ 1/ 706,246
Trace (E3)	0.72	0.99	0.83	67382/ 26774/ 1/ 2,389,233	0.95	0.99	0.97	67382/ 3477/ 1/ 2,412,530	0.95	0.99	0.97	67382/ 3805/ 1/ 2,412,202
Theia (E3)	0.87	0.99	0.93	25297/ 3765/ 65/ 3,501,561	0.92	0.99	0.95	25318/ 2282/ 44/ 3,503,044	0.93	0.99	0.96	25318/ 1875/ 44/ 3,503,451
Fivedirections (E3)	0.67	0.92	0.78	389/ 188/ 36/ 569,660	0.72	0.93	0.81	395/ 150/ 30/ 569,698	0.70	0.93	0.80	395/ 170/ 30/ 569,678
OpTC (Attack 1)	0.84	0.85	0.84	53/ 10/ 9/ 552,491	0.91	0.94	0.92	58/ 6/ 4/ 552,495	0.90	0.92	0.91	57/ 6/ 5/ 552,495
OpTC (Attack 2)	0.85	0.87	0.86	358/ 64/ 52/ 553,066	0.92	0.94	0.93	387/ 32/ 23/ 553,098	0.94	0.92	0.93	378/ 22/ 32/ 553,108
OpTC (Attack 3)	0.86	0.87	0.86	155/ 25/ 23/ 181,699	0.92	0.92	0.92	163/ 15/ 15/ 181,709	0.92	0.93	0.92	165/ 15/ 13/ 181,709

TABLE 3: Comparison of FLASH and Unicorn detector.

Datasets	System	Precision	Recall	F-score
StreamSpot	Unicorn	0.95	0.97	0.96
	FLASH	1.0	0.96	0.98
Unicorn SC-1	Unicorn	0.85	0.96	0.90
	FLASH	0.92	0.96	0.94
Unicorn SC-2	Unicorn	0.75	0.80	0.78
	FLASH	0.96	0.96	0.96
Theia (E3)	Unicorn	1.0	1.0	1.0
	FLASH	1.0	1.0	1.0
Cadets (E3)	Unicorn	0.98	1.0	0.99
	FLASH	1.0	1.0	1.0

RQ2. Scalability Analysis of FLASH



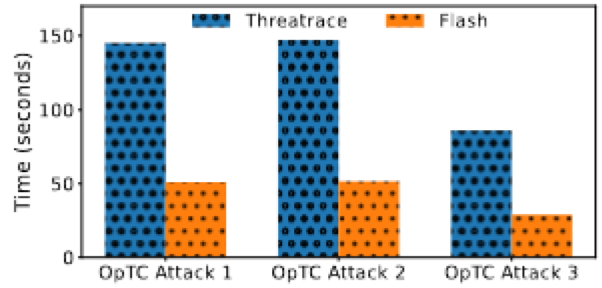
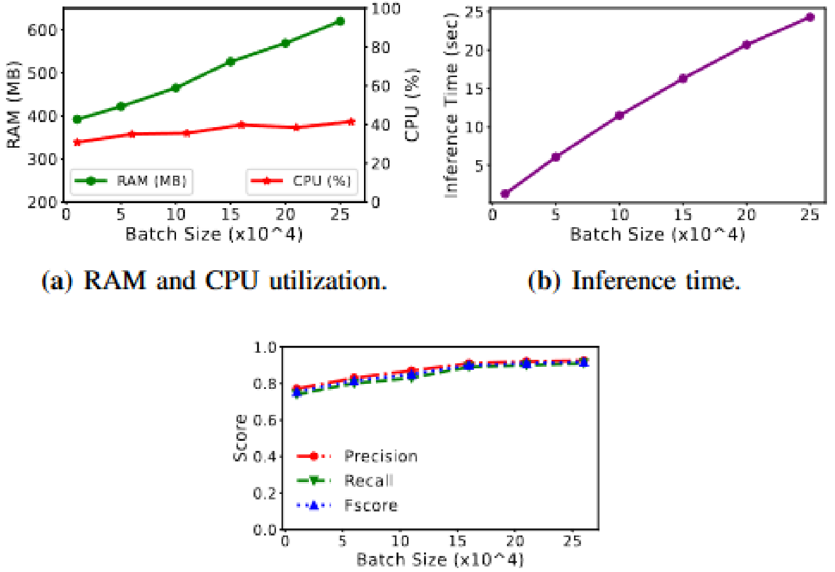


Figure 3: Inference times using one host logs from OpTC dataset. FLASH leverages embedding database to accelerate inference.

Evaluation

RQ3. Role of Batch Size

CPU utilization remains relatively constant, while memory consumption exhibits an almost linear growth with respect to the batch size



(a) RAM and CPU utilization. (b) Inference time.

(c) Detection performance.

Figure 4: Influence of batch size parameter K on different performance metrics of FLASH

RQ4. Robustness against Mimicry Attacks

Routine approaches : make the nodes within the attack graph have similar embeddings to nodes involved in benign activities.

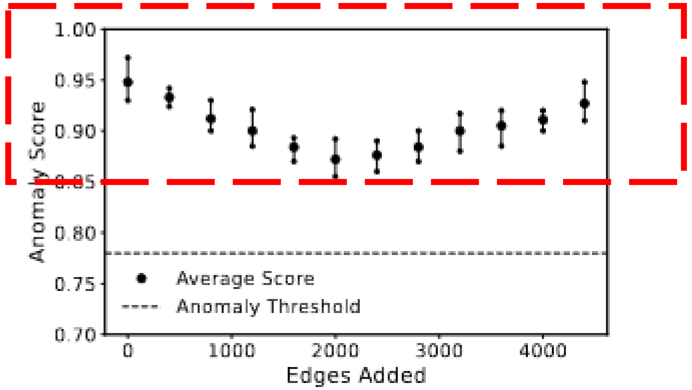


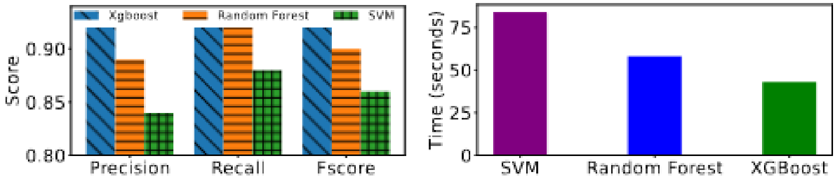
Figure 5: Adversarial mimicry attack against our system.

Explain: too many benign nodes may be regarded as an anomaly.

Evaluation

RQ5. Ablation Study

Varying Lightweight Classifiers



(a) Detection comparison. (b) Time overhead comparison.

Figure 6: Detection and time comparison of different classifiers.

Effect of Weighted Cross Entropy Loss

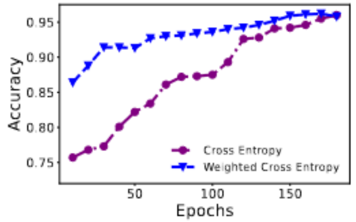


Figure 7: Effect of Weighted Cross Entropy on GNN Learning.

Efficacy of GNN Embeddings

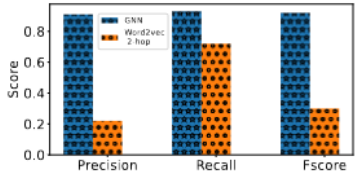


Figure 8: GNN vs. Word2Vec for capturing structural information.

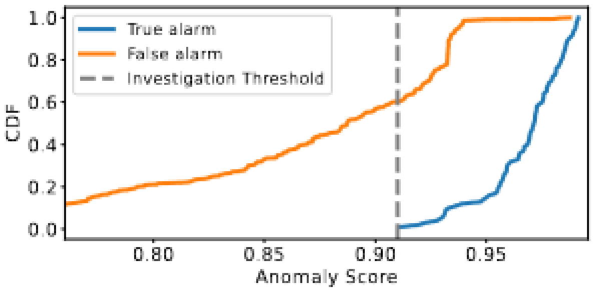
Effect of Temporal Ordering.

TABLE 4: Effect of considering temporal ordering.

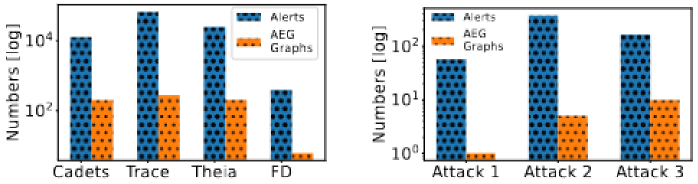
Temporal Order	Precision	Recall	F-Score	TP	FP
No	0.72	0.99	0.83	67382	26774
Yes	0.84	0.99	0.91	67382	12845

RQ6. Accelerating Alert Validation

Separation threshold



generate AEGs



(a) DAPRA Engagement 3 (b) DAPRA OpTC

Figure 11: Number of AEGs generated from the threat alerts present in DAPRA E3 and OpTC.

Discussion

Will unobserved benign activates generate many false activate ?

In this paper, GNN-based offline embedding are used to train benign data, but this approach cannot embed new benign data in a timely manner. How to address this problem?