A Context is Worth a Thousand Lies: Evading Intrusion Detectors via Intelligent Context Distortion

Magdy Nasr
University of Manitoba, Canada
Email: {nasrm1@myumanitoba.ca}

Vansh Rastogi
University of Manitoba, Canada
Email: {rastogiv@myumanitoba.ca}

Azadeh Tabiban✉
University of Manitoba, Canada
Email: {azadeh.tabiban@umanitoba.ca}

Abstract*—Provenance-based Intrusion Detection Systems (PIDSes) have become one of the most promising solutions for detecting sophisticated attacks. However, many existing PIDSes can be evaded by approaches obfuscating malicious nodes among frequent benign events (i.e., gadgets) or replacing rare malicious events with frequent benign sequences. While effective against earlier PIDSes, those approaches largely overlook advanced node-level PIDSes, which leverage semantic information, temporal ordering, and graph representation learning to model the full context of nodes. This enriched view makes evasions that rely only on frequency of events less effective, as malicious nodes can still be distinguished by their semantic and structural relationships. Based on such an observation, we propose Contorter, an evasion framework that leverages the enhanced embeddings of node-level PIDSes to guide gadget generation and uncovers their blind spots for improving robustness. Specifically, Contorter identifies benign nodes of the same entity type as each malicious node, and selects one with a similar context that is labeled benign with high confidence by the target PIDS. It then replicates edges of the selected node around the malicious node to further align their contexts and hide the latter from the PIDS. We implement and evaluate Contorter on Darpa E3, OpTC, Unicorn, and StreamSpot datasets, and based on four exemplar node-level PIDSes. The results show that Contorter can reduce the recall to as low as 0% (average 59% reduction), while keeping the false positive rate mostly unchanged to avoid suspicion. Finally, compared to prior approaches, Contorter achieves over a 35% greater reduction in recall under the same assumptions, while it requires about seven times fewer added edges.

1. Introduction

Modeling system behavior using sequences of system calls has long been a foundational technique in anomaly-based intrusion detection (e.g., [1], [2]). However, these sequence-based approaches have been shown to be vulnerable to evasion attacks [3], [4], [5], where attackers obfuscate malicious events among benign-looking ones, aka "gadgets". Provenance-based Intrusion Detection Systems (PIDSes) offer enhanced robustness against such attacks by capturing causal relationships between system activities (e.g., OS system calls) as a provenance graph [6], [7] and leveraging machine learning (ML) to detect anomalous subgraphs or paths [8], [9], [10]. Yet, researchers have demonstrated that even such ML-based PIDSes can be evaded by attackers injecting frequent benign-looking gadgets obfuscating malicious ones [11] or replace rare events with frequent sequences of gadgets [12].

Recent PIDSes have shifted toward fine-grained detection by analyzing individual nodes or edges. In particular, node-level PIDSes [13], [14], [15], [16] identify suspicious nodes while maintaining a balance between detail and efficiency [13], [16]. Such PIDSes enhance detection by combining semantic and temporal features with Graph Neural Networks (GNN) [13], [17], masked graph representation learning for subtle behavioral deviations [14], [18], and Variational Autoencoder for anomaly detection [15]. Evaluations of these systems indicate that current evasion approaches face various challenges to evade such advanced PIDSes in practice: i) The addition of random frequent gadgets may cause semantic and structural inconsistencies that are anomalous and detectable by the PIDS [13], [14], [19]. ii) Adding a small set of randomly selected frequent gadgets limits their obfuscation capability, while a large number of such redundant gadgets would be anomalous itself [13]. iii) Existing approaches merely rely on attackers to select contextually relevant and effective gadgets and avoid suspicious ones, which may be impractical [12], [20]. iv) Many strategies are most effective for specific groups of PIDSes rather than providing a unified approach [21].

Our key observation is that existing evasion approaches disregard the semantics (information about the nodes' represented system entity and their connected edges) and causal dependencies between events, which limits their effectiveness against advanced PIDSes performing fine-grained detection [13], [14], [15], [17], [22]. In this work, we show for the first time that a careful consideration of semantic and causal dependencies (i.e., context) can guide a semi-automated approach for generating gadgets and achieve more successful evasion. To this end, we propose Context Distorter (Contorter)[1], a framework that enables the evasion

. ✉ Corresponding author.

1. Our title plays on the phrase "A picture is worth a thousand lies", a social insight about the deceptive influence of images on our perception.

of node-level PIDSes to inform efforts toward more robust detectors. Our key idea is twofold. First, most advanced PIDSes detect malicious nodes as anomalous based on their context. Therefore, to evade such PIDSes, we can trigger gadgets that render the occurrence of the malicious nodes expected. Second, the learned representation of nodes capture their typical interactions and context. Therefore, tapping into nodes' representation can guide the selection of gadgets that often occur in similar contexts, thereby achieving effective evasion with fewer and more contextually relevant (i.e., plausible) gadgets. Specifically, for each malicious node, Contorter selects the benign node of the same entity type whose vector representation is similar to that of the malicious node. Then, it extracts and replicates the events of this benign node for the malicious node. Such events are represented as edges attached to the malicious node and shift its embedding toward contextually similar benign nodes in vector space, which would prevent its detection and reduce the chance of raising suspicion. In summary, our main contributions are as follows:

- We address the research gap that existing evasion approaches overlook semantics and context of system entities, which limits their effectiveness against advanced PIDSes. By leveraging contextual similarity between nodes, Contorter enables a semi-automated and context-aware evasion of fine-grained PIDSes while reducing the potential for excessive or suspicious out-of-context gadgets.
- We provide a unified methodology for Contorter which: i) identifies benign nodes of the same system entity type as each malicious node; ii) among those, it selects a contextually similar benign node that is classified with high confidence by the target PIDS; iii) replicates plausible edges of the selected node as gadgets for the malicious node.
- We implement and evaluate Contorter based on four representative node-level PIDSes and public datasets. Our results demonstrate that Contorter effectively shifts the embedding of malicious nodes toward the benign cluster and enables a high degree of evasion (average 59% reduction in recall). Compared to existing evasion approaches, Contorter achieves over a 35% greater reduction in recall, while requiring seven times fewer gadgets.
- To help the cybersecurity community in building more robust PIDSes, we release the source code of this project at https://github.com/focusResearchLab/Contorter.

2. Preliminaries

In this section, we provide a motivating example, our threat model, and a background of our exemplar PIDSes.

2.1. Motivating Example

Figure 1 illustrates the application of node-level PIDSes (left), the limited effectiveness of existing evasion approaches (middle), and our key ideas (right) to prevent the detection of "Malicious Upgrade" attack from Darpa OpTC dataset [23]. For simplicity, we depict a subset of the attack steps (nodes with red borders) and their relationships.

In this scenario, the attacker exploits the software update mechanism of Notepad++ to download a malicious binary, `update.exe`. Once executed, `update.exe` interacts with the privileged Windows service `wmiprvse.exe`, which then opens `svchost.exe` to communicate with the malicious server and establish a remote shell for reconnaissance. The attacker then migrates the malicious session from the compromised `svchost.exe` into `lsass.exe` to exfiltrate credentials.

Detection by Node-level PIDSes. Now, assume an existing node-level PIDS (e.g., [13], [14]) is applied to detect malicious nodes in this attack (illustrated on the left side of Fig. 1). The PIDS learns the typical behavior of system entities by representing each node as a vector that combines the semantics of the node with the aggregated vector representations of its neighbors. It then detects malicious nodes whose vector differs significantly from that of benign ones. For instance, based on the typical semantic and causal dependencies of nodes within a neighborhood, Flash [13] predicts the type of the node `update.exe` as file (captured by the red vector). Therefore, it flags `update.exe` as anomalous, since there is a mismatch between the type predicted by its context (i.e., file) and its actual type (i.e., process). On the other hand, the process node `wscript.exe` is not flagged, as its semantic and causal dependencies (captured by the green vector) lead Flash to predict its type as a process, matching its actual type.

Challenges Faced by Existing Evasion Approaches. To evade the PIDS, the attacker may leverage existing approaches such as [11] and [12] which are proven to be effective against many ML-based PIDSes. For instance, the attacker may inject frequent substructures or replace rare, suspicious events with frequent event sequences and, using expert knowledge, verify that they achieve the original attack objective and are not visibly suspicious. Despite their success in evading older PIDSes, those evasion approaches face significant challenges in bypassing node-level ML-based detection mechanisms (illustrated in the middle of Fig. 1). First, selecting frequent gadgets without considering the semantic and causal dependencies, cannot effectively shift the vector representation of `update.exe` toward the cluster of benign nodes (e.g., typical representation of process nodes) [13], [14]. Second, ineffective adjustment of the malicious node's vector may lead to adding more gadgets, which itself would be anomalous [13]. Third, the added frequent gadgets may be visibly suspicious (e.g., terminating the browser) or arbitrarily shift the malicious node's embedding when triggered out of their expected context [12]. Finally, although not shown in the figure, existing evasion frameworks typically apply methodologies specific to each PIDS (e.g., injecting benign substructures or low-regularity paths to evade unweighted or downsampled graph encodings, respectively) [11], which may increase the effort for generalizing the evasion.

Key Ideas Behind Contorter. Instead of generating gadgets based on the frequency of events, Contorter allows evading advanced PIDSes by leveraging the contextual similarity

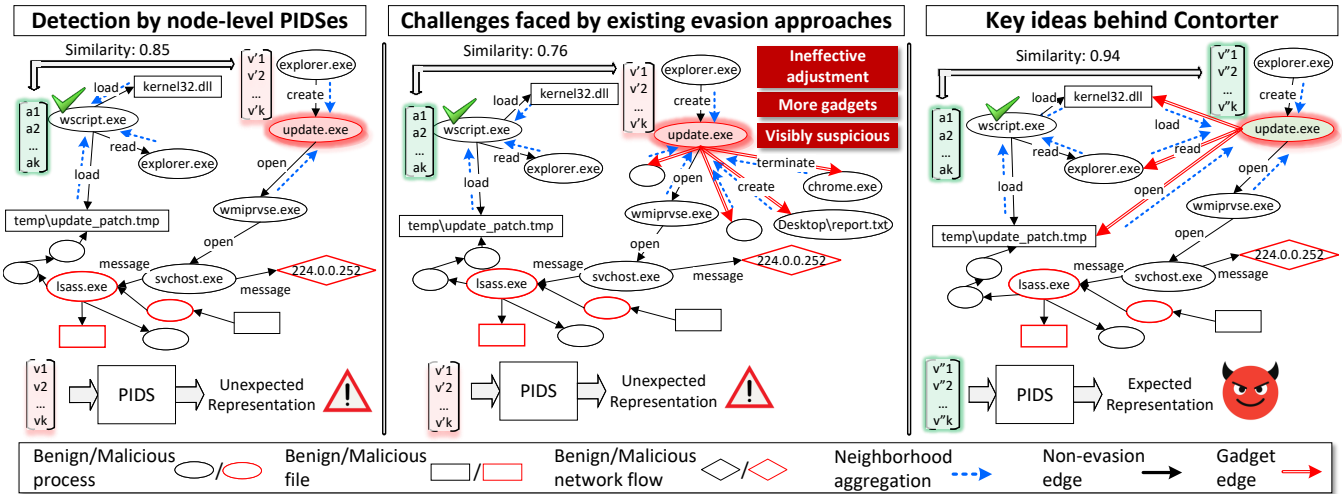


Figure 1: Example of detecting a malicious node by its anomalous vector (left); limitation of existing evasions (middle); the key ideas of our evasion approach (right).

of nodes. Specifically, Fig. 1 (right) illustrates the main ideas of Contorter for obfuscating malicious nodes (e.g., `update.exe`). First, we identify all benign nodes of the same type as the target malicious node (e.g., all nodes representing process system entities). Among these, we choose the node whose embedding is most similar to the malicious node and that the PIDS classifies as benign with high confidence (e.g., `wscript.exe`). Next, we replicate the interactions of `wscript.exe` with other system entities (e.g.,with `kernel32.dll` and `explorer.exe`), for `update.exe`. These replicated events appear as edges connected to the `update.exe` node (shown in red) and shift its embedding closer to benign process nodes, leading the PIDS to misclassify it as benign.

2.2. Example Provenance-based IDSes

We design Contorter based on a detailed analysis of node-level PIDSes, which cover different combinations of commonly adopted strategies for the granularity of detection, ML-based mechanism, and embedded features (illustrated in Table 8, Appendix C). Specifically, we detail the application of Contorter based on four representative PIDSes: Flash [13], Magic [14], NodLink [15], and Threatrace [17].

Flash. Flash [13] is a semi-supervised PIDS that combines semantic encoding with graph representation learning. It constructs provenance graphs from batches of system audit logs and encodes causal, temporal, and semantic attributes of nodes such as their process names, file paths, and event types using positional encoding and Word2Vec [24]. Then, Flash applies GNN [25], which aggregates initial node embeddings to derive their latent representations, and predicts each node's system entity based on those representations. A node is flagged as malicious when its predicted system entity type does not match its true system entity type.

Magic. Magic [14] is a self-supervised PIDS that leverages masked graph representation learning and distance-based outlier detection. Magic first represents the type of nodes and edges as one-hot vectors, which are processed by a Graph Attention Network (GAT)-based autoencoder to learn node embeddings that capture their semantic and causal relationships. It then applies a K-nearest neighbor (KNN) outlier detection algorithm [26] to detect nodes whose embeddings significantly deviate from those of the benign ones.

NodLink. NodLink [15] is an unsupervised PIDS that leverages semantic embedding and reconstruction errors to detect only anomalous process nodes. It trains a model of benign processes based on their associated command line, the file paths they accessed, and the network connections they made. These features are embedded for each process using FastText [27], and combined through a weighted sum. Next, NodLink trains a Variational Autoencoder (VAE) on the vector of benign processes. For detection, it applies the trained VAE to compute the anomaly score for each process and flags the process when its error exceeds a threshold.

Threatrace. Threatrace [17] is a semi-supervised PIDS that leverages the structural information of nodes and graph representation learning. Threatrace encodes each node by capturing the number of its incoming and outgoing edges representing different event types. Next, it applies GNN to aggregate nodes' vector representations, and accordingly, predict the type of their represented system entity. Threatrace flags a node as malicious when its type is predicted incorrectly or when the ratio between the probabilities of its two most likely types is below a threshold.

2.3. Threat Model

Similar to existing evasion mechanisms [11], [12] and MITRE T1592 [28], we assume attackers can obtain or infer audit data (e.g., /var/log/audit). Following [11], we assume that in black-box settings, when direct access to logs is not possible, attackers may gain a limited view of system activity by approximating information reflected

in logs through monitoring process behavior and network connections (e.g., via *ps* and *netstat*), or side channels (e.g., [29]). We also assume that attackers know the malicious nodes of the original attack (i.e., ground truth), which is reasonable given attackers' familiarity with the attack. Moreover, while attempting to evade detection, the attacker is still required to achieve the original objectives of the attack. Further, similar to existing provenance-based solutions and evasion frameworks [11], [20], [30], we assume the underlying operating systems, auditing frameworks, and PIDSes are parts of the trusted computing base and their integrity is protected through hardening [31] and tamper-evident logging mechanisms [32], [33]. Above assumptions are applicable to both our white-box and black-box models. We primarily focus on white-box model, where attackers additionally know or can infer the architecture of the target PIDS (e.g., through reconnaissance such as inspecting logs or alerts). We assume the attacker can verify the evasion success for malicious system entities while fewer verification attempts are advantageous for reducing delays. For instance, the attacker may train a surrogate model of the PIDS on a local host using public datasets or activity logs of the target host, or directly obtain the detection model from the PIDS, as in [12]. In our black-box model, attackers cannot know which PIDS is deployed on the target host, cannot access the node embeddings, and detection confidence. We recognize that in such settings, where the PIDS is unknown, attackers can only verify the evasion (i.e., query the PIDS model) by observing alerts after they have already been raised and are visible to the analyst. Therefore, we assume that black-box attackers cannot query the PIDS to verify and refine the evasion, which further limits their knowledge compared to prior work [12].

3. Methodology

This section provides an overview of Contorter, then details its modules and application to evade various PIDSes.

Overview. As shown in Fig. 2, Contorter consists of three main stages as follows. It first captures batches of logged events as a provenance graph with malicious nodes, which it aims to obfuscate. Next, Contorter identifies benign nodes of the same entity type as each malicious node and selects the one whose context is similar to that of the malicious node, and is classified as benign with high confidence by the target PIDS. Finally, it replicates the edges of the selected node for that malicious node. Algorithm 1 summarizes our approach.

3.1. Provenance Construction

PIDSes initiate the detection by capturing the causal relationships between audit logs as a provenance graph [13], [14], [15]. Specifically, they parse raw audit logs, and transform them into a provenance graph $G = (V, E)$. As PIDSes perform the detection based on this graph, it is also provided as input to Contorter for identifying potential candidates and generating gadgets. Such a provenance graph can be reconstructed by the attacker based on the system logs of the target host and contains the malicious nodes involved in the attack, which Contorter aims to hide (i.e., prevent the PIDS from flagging them). In the black-box setting, where attackers cannot access logs, they may reconstruct an approximate provenance graph by correlating observable system behavior, such as process relationships (e.g., via *ps*), file activity (e.g., inferred from file metadata using *stat*), and network connections (e.g., via *netstat*).

Algorithm 1: Contorter

Input: Provenance graph $G = (V, E)$
Node embedding and detection model Emb, D
Malicious nodes $VM \subseteq V$
Output: Curated Provenance Graph $G' = (V, E')$
// $E' = E \cup$ new edges
foreach *malicious node* $v_m \in VM$ **do**
  $Cands \leftarrow \text{TypeSel}(v_m.type, v_b.type), \forall v_b \in V - VM$ // Candidates with same type as v_m
  $Cands \leftarrow \text{FootprintOpt}(v_b, R), \forall vb \in Cands$ // Candidates with acceptable # of edges
  $Cands \leftarrow \text{CSMax}(Emb(v_m), Emb(v_b), T_s), \forall v_b \in Cands$ // Ranked T_S most contextually similar to v_m
  $v_c \leftarrow Cands[0]$
  $e' \leftarrow \text{GadRet}(v_c)$
  if $\text{OccVer}(e', G)$ **then**
    $E' \leftarrow E \cup \{e'\}$ // Replicate interactions
  $i \leftarrow 0$
  while $EVer(v_m)$ *is false* **do**
    // while v_m is flagged
    $Cands \leftarrow \text{ImpMax}(Cands, D)$ // Ranked $Cands$ by classification confidence
    $v_c \leftarrow Cands[i]$
    $e' \leftarrow \text{GadRet}(v_c)$
    if $\text{OccVer}(e', G)$ **then**
      $E' \leftarrow E \cup \{e'\}$ // Replicate interactions
    $i \leftarrow i + 1$
return $G' = (V, E')$

3.2. Candidate Selection

This section describes the modules that, for each malicious node, select benign candidate nodes to enable effective gadget generation. Those modules are applied incrementally, with each one incorporating additional features to further refine the selection of candidates.

3.2.1. Type-based Selection (TypeSel). Our Type-based Selection (TypeSel) module guides modifying the context of each malicious node toward resembling that of benign nodes of the same entity type. Our intuition is that, as nodes of the same type are typically embedded closely, small adjustments to a malicious node's edges may be enough to shift its embedding to the cluster of benign nodes of the same type (i.e., obfuscating the malicious node). Moreover, a malicious node appears less suspicious within a neighborhood typical for its type (e.g., process nodes within neighborhoods typical

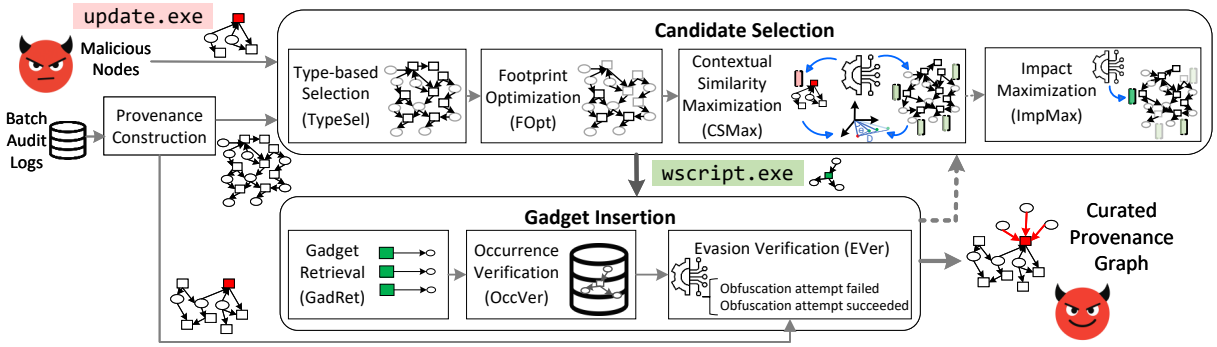


Figure 2: Overview of Contorter methodology.

for processes rather than files). Therefore, for each malicious node, TypeSel identifies all benign nodes of the same type as potential candidates and provides them to the subsequent modules for further refinement.

Application to Different PIDSes. Recall that Flash and Threatrace flag a node as anomalous when the system entity type predicted for that node based on its local neighborhood differs from its actual type. Therefore, making a malicious node's neighborhood similar to that of benign nodes with the same type may cause these PIDSes to predict its true type. Similarly, Magic flags a node when its embedding (capturing the type of the node, its neighbors, and the edge connecting them) is far from that of benign clusters. As nodes of the same type are typically embedded closer, it is easier to shift the malicious node's embedding toward that of benign nodes of the same type as the malicious node. Finally, for NodLink, candidates are selected only among process nodes, as it focuses on detecting malicious processes.

Example 1. To obfuscate the malicious node `update.exe` representing a process in our Motivating Example (Section 2.1), TypeSel identifies all benign process nodes. Table 1 shows example nodes from the provenance graph that are assessed by TypeSel. Nodes representing processes `wscript.exe`, `desktop.ini`, `Microsoft Office`, `Google Chrome.lnk`, and `chrmstp.exe` are selected as candidates by TypeSel and passed to the next modules. However, the node representing the file `cryptbase.dll` is excluded.

TABLE 1: Example of nodes assessed by Contorter to select candidates for `update.exe`. TypeSel leverages the column *Type* (the adoption of other columns will be explained later).

Candidate	Type	# Edges	Similarity	Confidence
wscript.exe	Process	5	0.89	0.71
desktop.ini	Process	3	0.75	0.96
Microsoft Office	Process	3	0.67	
Google Chrome.lnk	Process	2		
chrmstp.exe	Process	113		
cryptbase.dll	File			

3.2.2. Footprint Optimization (FOpt). An insufficient number of edges added to a malicious node may not effectively shift its embedding toward the benign cluster, while adding too many edges may produce an anomalous embedding [13] and requires triggering more gadgets, which itself can raise suspicion [12]. Therefore, among candidates provided by TypeSel for a malicious node, our Footprint Optimization (FOpt) module obtains those whose number of edges *e(n)* satisfies $e_{\min} \leq |e(n)| \leq e_{\max}$, where $e_{\min}$ and $e_{\max}$ are the minimum and maximum allowed number of edges for those candidates, respectively. The parameters $e_{\min}$ and $e_{\max}$ can be adjusted by the attacker to balance the risk of getting noticed by analysts and the extent of vector modification. We will evaluate Contorter across various specified ranges for FOpt in Section 4.7.

Application to Different PIDSes. Flash captures the edges connected to each node in an array of features. Therefore, in our implementation for Flash, FOpt is applied by setting an acceptable length limit for each node's feature array. In Magic, the provenance graph is stored as a graph object, and thus, FOpt is applied by directly setting a constraint for the number of edges connected to each node. NodLink computes process nodes' embeddings using their neighbor file and network flow nodes. Therefore, FOpt retains only candidate nodes whose number of such neighbors is within an acceptable range. Threatrace encodes each node as a fixed-size vector, where each position represents the number of connected edges of a particular event type. Accordingly, FOpt imposes a constraint on the sum of position values.

Example 2. Following Example 1, the third column of Table 1 shows the number of edges connected to each candidate node. Given the acceptable range of $[3, 30]$ for connected edges, candidate nodes such as `Google Chrome.lnk` (2 edges) and `chrmstp.exe` (113 edges) are excluded by our FOpt module, and other three candidates (row 1-3) are passed to our next modules.

3.2.3. Contextual Similarity Maximization (CSMax). Considering contextual similarity when selecting a candidate increases the potential for successful evasion as follows: First, aligning the context of a malicious node with that of an already similar candidate would intrinsically result in less conspicuous modifications (i.e., more contextually relevant gadgets). This reduces the reliance on attackers' expertise and manual effort to discard out-of-context suspicious gadgets. Second, such a contextual alignment requires fewer changes than making the malicious node's context

similar to that of arbitrary nodes. Based on this intuition, our Contextual Similarity Maximization (CSMax) module computes the similarity between the vector representation of all candidate nodes obtained by the previous steps and that of the target malicious node. Next, it orders the candidates based on their similarity score and provides the top T_s candidates to the subsequent modules.

Application to Different PIDSes. CSMax computes node similarity based on the embeddings produced by the PIDS (see Section 2.2), and these embeddings can be obtained by a white-box attacker. As the embedding of node features in Flash, NodLink, and Threatrace relies on directional similarity [34], CSMax computes the cosine similarity between the malicious node's embedding (e_m) and each candidate's embedding (e_b), as $\frac{\mathbf{e}_m \cdot \mathbf{e}_b}{\|\mathbf{e}_m\| \|\mathbf{e}_b\|}$. It then ranks them based on their similarity score. Magic detects malicious nodes based on the Euclidean distance between each node's embedding and the embedding of benign nodes. Accordingly, CSMax computes the average Euclidean distance between the embedding of the malicious node (e_m) and k candidates' embedding (e_b) as $\frac{1}{k}\sum_{i=1}^{k}\|e_m - e_b\|_2$. It then ranks the benign candidates by their Euclidean distance to the target malicious node, assigning higher ranks to those with smaller distance.

Example 3. Following Example 2, CSMax computes the contextual similarity between the candidates provided by previous modules (rows 1–3 in Table 1) and the target malicious node `update.exe`, and then ranks the candidates by similarity. In this example, suppose T_s is specified such that the first two candidates (row 1-2) are selected and passed to the subsequent modules. Therefore, the candidate node `Microsoft Office` is excluded by CSMax due to its lower contextual similarity with the malicious node (which could otherwise lead to potentially out-of-context gadgets).

3.3. Gadget Insertion

In this section, we present the modules that obfuscate a target malicious node by replicating the edges of the highest-ranked candidate around the malicious node.

3.3.1. Gadget Retrieval (GadRet). Our Gadget Retrieval (GadRet) module transforms the edges connected to the candidate node into gadgets. To this end, it first retrieves all 1-hop neighbors of the selected candidate node and the edges connecting the candidate to those neighbors. Next, it generates a sequence of gadgets involving the malicious node, such that the resulting edges (once added to the provenance graph) match the retrieved edges in terms of event type, direction, and connectivity to the candidate node's 1-hop neighbors, but originate from or terminate at the malicious node instead. Formally, let E_c be the set of edges connected to v_c, denoted as:

$$E_c = \{(v_c, v_i, r) \in E \mid v_i \in \mathcal{N}(v_c)\} \cup \{(v_i, v_c, r) \in E \mid v_i \in \mathcal{N}(v_c)\} \quad (1)$$

where each triplet represents a directed edge between the candidate node v_c and a node v_i in its one-hop vicinity $\mathcal{N}(v_c)$, r denotes the type of event the edge represents, and E denotes the set of all directed edges in the graph. GadRet creates a sequence of gadgets $[e_i, e_{i+1}, ...]$ such that when captured as edges, they would be represented as:

$$E_m = \{(v_m, v_i, r) \mid (v_c, v_i, r) \in E_c\} \cup \{(v_i, v_m, r) \mid (v_i, v_c, r) \in E_c\} \quad (2)$$

where each triplet denotes a directed edge between the target malicious node v_m and one of the 1-hop neighbors of v_c, and E_c is the set of edges directly connected to v_c. This set of edges (E_m) is assessed by our next module for improving the plausibility of evasion, and those that pass the verification are replicated around the target malicious node.

Example 4. Following Example 3, suppose a node representing `wscript.exe` is selected as the highest-ranked candidate to obfuscate `update.exe`. Fig. 3 (left) shows this candidate, its 1-hop neighbors and edges between them. GadRet extracts the edges connected to `wscript.exe` and accordingly, creates a list of potential gadgets represented as edges around `update.exe` (illustrated on the right side of Fig. 3), which preserve the event type, the direction, and the affected system entity (e.g., `explorer.exe`, `kernel32.dll`, etc.) of the original edges.

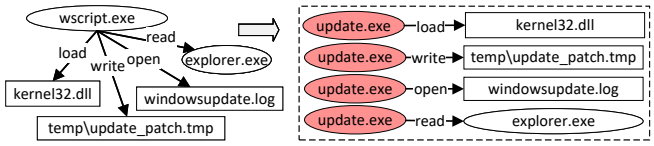


Figure 3: Example of gadget retrieval.

3.3.2. Occurrence Verification (OccVer). In addition to previous modules that enhance the contextual relevance of gadgets, our Occurrence Verification (OccVer) module aims to further improve the plausibility by allowing only gadgets observed in historical logs. Triggering gadgets matching those previously observed in the system is more realistic and less likely to attract analysts' attention. Accordingly, OccVer module conservatively discards any gadgets not found in historical logs (i.e., logs of the target host or public datasets). Specifically, for a gadget of type r represented by a directed edge between v_m and v_i, OccVer searches historical provenance graphs for a matching edge (v_s, v_d, e) where $e = r$, and the direction of edge between v_m and v_i matches that in the historical record; the gadget is retained only if such an edge exists. Although the attacker can further ensure the plausibility by manually inspecting the validated gadgets, such a semi-automated approach (instead of sole reliance on manual verification [12]) can help achieve more efficient and less error-prone evasion, even for attackers with limited expertise.

Example 5. Following Example 4, OccVer searches the historical provenance graph for the gadgets obtained by GadRet to obfuscate `update.exe` (illustrated on the right side of Fig. 3). For instance, it locates `update.exe`−(`open`)→`explorer.exe` in the provenance graph, and thus retains it for the next modules.

3.3.3. Evasion Verification (EVer). Our Evasion Verification (EVer) module evaluates the effectiveness of the candidate selected for each malicious node. Specifically, EVer first curates the provenance graph by replicating the gadgets verified by OccVer around the malicious node. Next, it queries the PIDS model by applying it to the curated provenance graph and assesses whether the added gadgets successfully obfuscate the malicious node. Finally, EVer provides the malicious nodes that remain flagged to the next module in our Candidate Selection stage (Section 3.4), which is invoked only when the previously selected candidates fail. Note that this verification is not performed in our black-box setting, as it assumes the attacker lacks access to the PIDS model (Section 2.3).

Example 6. Following Example 5 and after adding gadget edges around the malicious node `update.exe`, the embedding of this node becomes similar to that of the candidate node `wscript.exe` (their cosine similarity increases from 0.85 to about 0.94), which prevents the PIDS from flagging `update.exe`. Note that although this modification is sufficient for evasion, the embeddings of `update.exe` before and after the modification remain highly similar (cosine similarity = 0.98). Such a subtle embedding modification implies a higher contextual relevance of the gadgets, which may draw less attention from the analysts of the target host.

3.4. Impact Maximization (ImpMax)

Replicating edges of the most similar candidate may not always shift the malicious node's embedding sufficiently toward the benign cluster (e.g., when the candidate is close to the malicious node but distant from most benign nodes). In such cases, our Impact Maximization (ImpMax) module selects another contextually similar candidate that the PIDS classifies as benign with a high *confidence*, as replicating such a candidate's edges around the malicious node can more effectively shift its embedding. Fig. 4 shows how ImpMax works with our other modules. Specifically, for the T_s most similar nodes provided by CSMax, our ImpMax module first computes the confidence of the target PIDS in classifying each candidate as benign. It then ranks those candidates by confidence and iterates through them. It begins with the highest-ranked candidate, replicates its edges around the malicious node, and verifies the evasion as detailed in Section 3.3. The iteration stops when it identifies a candidate that prevents the PIDS from flagging the malicious node. We will evaluate Contorter under different values of T_s in Section 4.7. In the following, we detail how the confidence is computed for each exemplar PIDS.

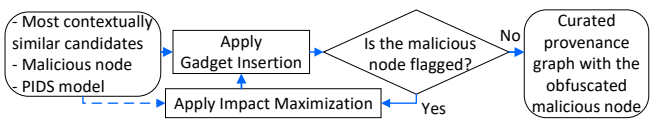


Figure 4: Identifying a similar candidate with high classification confidence for a given malicious node.

Application to Different PIDSes. Flash leverages several (10 to 22) GNN models trained by different randomly initialized weights. Each model predicts the type of the system entity represented by every node, and a node is considered benign when at least one model predicts its type correctly. Therefore, our ImpMax module computes the classification confidence by computing $\frac{\text{\# of correct predictions}}{\text{Total \# of models}}$, which assigns candidates whose type is correctly predicted by a larger number of models with a higher confidence score. Similarly, as Magic detects malicious nodes based on the average distance of nodes' vector from the benign cluster, candidates with a smaller distance from nearest benign neighbors are assigned with a higher confidence score. NodLink performs the detection based on the reconstruction error of nodes, and thus, ImpMax computes the reconstruction error of candidates as their confidence score. Threatrace flags a node as malicious when the ratio between the probability assigned to its true type and the probability assigned to the second most likely type is low. Therefore, ImpMax assigns higher confidence scores to nodes with a larger ratio between such highest two probabilities.

Example 7. For the sake of this example, suppose the obfuscation of `update.exe` is not successful using the most contextually similar candidate, `wscript.exe`. In this case, among the two most contextually similar candidates provided by CSMax, our ImpMax module selects the one with the highest classification confidence. Specifically, the process node `desktop.ini` is selected with the classification confidence of 0.968, which is greater than the confidence of the other candidate, `wscript.exe` (0.717). After replicating the edges of `desktop.ini` around `update.exe`, the cosine similarity between their embeddings increases from 0.80 to 0.96, causing the PIDS to predict the type of `update.exe` as a process, and thus not flag it.

In our experiments (Section 4.4), we show that although leveraging ImpMax is not always required for a successful evasion, this module is especially helpful when different clusters are formed farther from each other or when nodes have high intra-cluster distance.

4. Experiments

To evaluate Contorter, we seek to answer the following research questions.

- **RQ1.** Does Contorter effectively evade state-of-the-art ML-based PIDSes? What is the impact of attacker's knowledge on the evasion?
- **RQ2.** How does Contorter compare to other evasion approaches?
- **RQ3.** How does the confidence of detection degrade as Contorter adds more gadgets?
- **RQ4.** How effective is each module of Contorter in the success of the evasion?
- **RQ5.** Can the attacker deploy our gadgets in practice?
- **RQ6.** How scalable is Contorter in handling large provenance graphs as the number of malicious nodes increases?

• **RQ7.** How does the effectiveness of Contorter vary with different values of its internal parameters?

Experimental Setup and Dataset. All experiments were performed on a machine equipped with a 24-core Intel Core i9-13900K CPU, 128 GB RAM, an NVIDIA RTX 4090 GPU, and Ubuntu 18.04.6 LTS. We evaluate Contorter based on four public datasets: Darpa E3 [23], Darpa OpTC [35], Unicorn [10], and StreamSpot [9]. The Darpa E3 dataset contains system-level provenance traces from Darpa's Transparent Computing Engagement 3, where a red team executed coordinated, multi-stage attacks on enterprise-style networks alongside benign activities. The traces were collected using four mechanisms, Cadets, Theia, Trace, and Fivedirections each deployed in a separate environment targeted by the same attacks but differing in operating systems, instrumentation, and collection duration. The OpTC dataset comprises three enterprise-scale attacks that differ in scope, duration, and adversarial behavior. Additionally, we evaluate Contorter based on StreamSpot and Unicorn datasets, which are commonly used for evaluating graph-level detection.

For consistency, we use the same Darpa E3 training and testing partitions provided in Flash's open-source implementation to evaluate evasion in Flash, Threatrace, and NodLink. However, when trained and tested on the same E3 partitions as other PIDSes, Magic had low recall even before evasion, which prevented a fair evaluation of our evasion approach. Therefore, for Darpa E3, we evaluate Magic's evasion using the training and testing partitions provided in its open-source implementation. For OpTC, since none of our benchmarked PIDSes explicitly specify the training and testing partitions, we train PIDSes on a partition without malicious (ground truth) nodes and evaluate them on a separate partition containing both benign and malicious nodes.

Ground Truth and Metrics. For Flash and Threatrace, we aim to obfuscate the malicious nodes indicated by the ground truth in Flash [13]. For NodLink [15], we consider only the process nodes among those ground truth malicious nodes, as its detection scope is limited to malicious processes. For Magic, which uses different training and testing partitions from the other PIDSes, we rely on the ground truth provided in its implementation. For node-level detection, a True Positive (TP) is a node flagged by the PIDS that also appears in the ground truth, and a False Positive (FP) is a node flagged by the PIDS that does not appear in the ground truth. Node-level recall is defined as $\frac{TP}{TP+FN}$, and precision is defined as $\frac{TP}{TP+FP}$; We define False Positive Rate (FPR) as $\frac{FP}{FP+TN}$, where TP, FN, FP, and TN are the numbers of true positive, false negative, false positive, and true negative nodes. We also measure F1 by calculating $\frac{2 \cdot Precision \cdot Recall}{Precision+Recall}$.

4.1. Effectiveness

This section answers RQ1 by evaluating evasion against four PIDSes: Flash [13], NodLink [15], Threatrace [17], and Magic [14]. Section 4.1.1 and Section 4.1.2 present the results for node-level and graph-level detection, respectively.

4.1.1. Evading Node-level Detection. Table 2 shows that, on average, Contorter reduces the recall to almost 0.2 (59% reduction). For Cadets, the evasion is more effective in Flash, Magic, and Threatrace (99% reduction) than in NodLink (around 74% reduction). One reason is that in NodLink, nodes from different clusters are embedded closer to each other than in other PIDSes (e.g., average inter-cluster similarity of 0.49 in NodLink versus 0.23 in Flash). Therefore, candidates from such distant clusters in Flash can cause a larger shift in the embedding modification of malicious nodes than in NodLink. Further, in NodLink, Contorter achieves complete evasion (zero recall) for Theia and Trace, while it reduces the recall to 0.26 for Cadets. This may be due to the lower number of edges initially connected to malicious process nodes in Theia and Trace (an average of about four, compared with 11 in Cadets), which allows their embeddings to shift more significantly with the added gadgets. Moreover, for Theia dataset, evasion is lower in Threatrace than in the other PIDSes. This may be due to the higher FPR of Threatrace on this dataset before evasion, which subsequently reduces the number of effective candidates available for each malicious node. Evasion is lower on Fivedirections than on other E3 datasets, which is mainly due to its larger number of malicious process nodes interacting with many system entities, which reduces the impact of added gadgets on shifting their embeddings.

Contorter is more effective in Darpa E3 than in OpTC for Flash and Threatrace. This may be due to the larger and more diverse set of candidates in E3 (average of 71,000 nodes) compared to OpTC (average of 40,000 nodes, where about 80% of benign nodes share similar neighborhoods [13]), which increases the likelihood of finding suitable candidates in E3. Additionally, OpTC includes richer semantic information than E3, which enables enhanced detection and subsequently, hinders the evasion. Upon examining our results, we observe that most malicious processes in OpTC can be obfuscated by Contorter, which explains the greater evasion for NodLink, whose recall is based only on processes. Finally, our results show that FPR remains nearly unchanged in most cases, which shows that Contorter hides malicious nodes without introducing additional false positives that could raise suspicion. FPR only increases considerably (63%) in evading Threatrace on Cadets, where 99% of malicious nodes have the same connected edge type ("execute"). As Threatrace encodes nodes only based on the number of their connected edges with certain types, such malicious nodes would have similar embeddings, which causes our CSMax module to select similar candidates whose edges are subsequently replicated for many malicious nodes. Such a repetitive added substructure makes the context of benign nodes anomalous, thereby increasing the FPR.

2. In our experiments, we followed Wang et al. [36] by considering TPs to be ground truth nodes that are within the 1-hop vicinity of flagged nodes. This may explain the different recall values for OpTC prior to evasion compared to those in the original papers, which assume ground truth nodes within 2-hop vicinity of flagged nodes can be considered as TPs.

TABLE 2: Results of evading node-level detections. Parentheses indicate the difference from the results without evasion.

Dataset	Flash[2]			NodLink			Threatrace			Magic		
	Recall	F1	FPR	Recall	F1	FPR	Recall	F1	FPR	Recall	F1	FPR
Cadets	0.00 (–1.00)	0.01 (–0.31)	0.01 (-0.15)	0.26 (–0.74)	0.18 (–0.51)	0.00 (0.00)	0.00 (–0.99)	0.00 (–0.39)	0.75 (+0.63)	0.00 (–0.99)	0.00 (–0.97)	0.00 (0.00)
Theia	0.00 (–1.00)	0.00 (–0.81)	0.04 (0.00)	0.00 (–0.70)	0.00 (–0.82)	0.00 (0.00)	0.14 (–0.86)	0.07 (–0.08)	0.24 (–0.64)	0.00 (–0.99)	0.00 (–0.99)	0.00 (0.00)
Trace	0.00 (–0.99)	0.00 (–0.73)	0.04 (0.00)	0.00 (–0.80)	0.00 (–0.61)	0.00 (0.00)	0.00 (–0.94)	0.00 (–0.38)	0.15 (–0.09)	0.00 (-0.99)	0.00 (–0.99)	0.01 (0.00)
Fivedirections	0.29 (–0.44)	0.02 (–0.04)	0.02 (0.00)	0.16 (–0.67)	0.28 (–0.55)	0.00 (0.00)	0.33 (–0.65)	0.00 (–0.02)	0.10 (+0.01)	0.06 (-0.94)	0.00 (0.00)	0.06 (+0.02)
OpTC1	0.19 (–0.05)	0.18 (+0.01)	0.00 (0.00)	0.00 (–0.50)	0.00 (–0.58)	0.01 (+0.01)	0.73 (–0.19)	0.02 (–0.00)	0.87 (–0.04)	0.38 (-0.02)	0.01 (0.00)	0.36 (0.00)
OpTC2	0.19 (0.00)	0.27 (+0.06)	0.00 (0.00)	0.09 (–0.77)	0.12 (–0.69)	0.03 (0.00)	0.77 (0.00)	0.01 (0.00)	0.65 (0.00)	0.62 (-0.30)	0.00 (0.00)	0.80 (0.00)
OpTC3	0.15 (–0.29)	0.21 (–0.05)	0.00 (0.00)	0.00 (–0.63)	0.00 (–0.49)	0.07 (-0.01)	0.66 (0.00)	0.01 (0.00)	0.37 (0.00)	0.82 (-0.11)	0.01 (0.00)	0.17 (0.00)
Avg. E3	**0.07 (–0.86)**	**0.01 (–0.47)**	**0.03 (–0.04)**	**0.11 (–0.73)**	**0.12 (–0.62)**	**0.00 (0.00)**	**0.12 (–0.86)**	**0.02 (–0.22)**	**0.31 (–0.02)**	**0.02 (–0.98)**	**0.00 (–0.74)**	**0.02 (+0.01)**
Avg. OpTC	**0.18 (–0.11)**	**0.22 (+0.01)**	**0.00 (0.00)**	**0.03 (–0.63)**	**0.04 (–0.59)**	**0.04 (0.00)**	**0.72 (–0.06)**	**0.01 (0.00)**	**0.63 (–0.01)**	**0.61 (–0.14)**	**0.01 (0.00)**	**0.44 (0.00)**
Avg. Total	**0.12 (–0.54)**	**0.10 (–0.27)**	**0.02 (–0.02)**	**0.07 (–0.69)**	**0.08 (–0.61)**	**0.02 (0.00)**	**0.38 (–0.52)**	**0.02 (–0.12)**	**0.45 (–0.02)**	**0.27 (–0.62)**	**0.00 (–0.42)**	**0.20 (0.00)**

TABLE 3: Black-box evasion results. Parentheses indicate the difference from the results without evasion.

Dataset	Flash	NodLink	Threatrace	Magic
Cadets	1.00 (0.00)	0.46 (-0.54)	0.00 (-0.99)	0.00 (-0.99)
Theia	0.85 (-0.15)	0.00 (-0.70)	0.94 (-0.06)	0.99 (0.00)
Trace	0.00 (-1.00)	0.80 (0.00)	0.00 (-0.94)	0.00 (-0.99)
OpTC1	0.20 (-0.04)	0.00 (-0.58)	0.87 (-0.05)	0.39 (-0.01)
OpTC2	0.19 (0.00)	0.14 (-0.71)	0.22 (-0.50)	0.84 (-0.08)
OpTC3	0.44 (0.00)	0.00 (-0.63)	0.66 (0.00)	0.83 (-0.10)
Average	**0.45 (-0.20)**	**0.23 (-0.53)**	**0.45 (-0.42)**	**0.51 (-0.36)**

Node-level Black-box. We also evaluate Contorter under strict black-box settings, where attackers cannot know the target PIDS, node embeddings, and detection confidence, and cannot query the PIDS to enhance the evasion, which significantly limits their knowledge compared to prior work [12]. In such a setting, Contorter selects the optimal candidate only based on the type of its represented system entity and the number of its interactions. Table 3 shows that, on average, black-box attackers leveraging Contorter can achieve 41% recall reduction (18% less reduction than white-box in Table 2). These results demonstrate the effectiveness of the foundation of our approach leveraging the similarity of nodes (i.e., node types in black-box settings) to guide node-level context modification. Additionally, the lower evasion degree in black-box compared to white-box settings further highlights the importance of our adoption of node embeddings and model confidence available in white-box settings. For instance, although our black-box model achieves a high degree of evasion on Cadets for Threatrace, Magic and NodLink, it does not evade Flash on this dataset, as without considering the evasion confidence, Contorter selects candidates that are falsely flagged by Flash and cannot sufficiently shift the embedding of malicious nodes. Finally, Flash demonstrates greater overall robustness against black-box attackers (20% average recall reduction) than other PIDSes. This is mainly due to the enriched node embeddings in Flash that capture semantic, temporal, and causal dependencies (instead of only one of those dependencies as in most other PIDSes), which hinders black-box attackers lacking access to such embeddings.

4.1.2. Evading Graph-level Detection. Table 4 shows our results on evading anomaly detection for datasets with graph-level ground truth (i.e., Unicorn and StreamSpot). For those datasets, Flash flags a graph as suspicious when the number of its flagged nodes exceeds a pre-specified threshold, Th (e.g., 330 nodes). This means that to achieve one false negative at the graph level, Contorter must successfully obfuscate at least Th malicious nodes in a given graph, which is more challenging than in node-level datasets, where each false negative corresponds to a single obfuscated node. Despite the increased challenge, our results demonstrate the effectiveness of Contorter in evading Flash, reducing recall by 96% and 88% on StreamSpot and Unicorn, respectively. Our investigation of datasets shows that the frequency of unique tokens is higher in StreamSpot, which results in embeddings that form more widely separated clusters [37]. Candidates from such more distant clusters can cause a more significant shift in malicious nodes' embedding, which leads to higher evasion success on the StreamSpot dataset.

TABLE 4: Results of evading graph-level detection. Shaded cells indicate black-box evasion. Parentheses show the difference from the results without evasion.

Dataset	Flash		NodLink		Magic	
	Rec.	FPR	Rec.	FPR	Rec.	FPR
StreamSpot	0.00 (–0.96)	0.00 (0.00)	0.00 (–0.65)	0.69 (0.00)	0.79 (–0.21)	0.00 (0.00)
Unicorn	0.08 (–0.88)	0.04 (0.00)	0.97 (+0.38)	0.59 (0.00)	0.85 (–0.11)	0.00 (0.00)

Unlike Flash, NodLink does not provide an implementation for graph-level detection, and it requires a different detection threshold than Flash as NodLink focuses on process nodes. Therefore, to evaluate the evasion capability of Contorter against NodLink based on Unicorn and StreamSpot, we leveraged the nodes flagged by Flash as ground truth. As Table 4 shows, Contorter significantly reduces recall by 65% on StreamSpot, but it does not evade NodLink on Unicorn. One reason is that the semantic information of nodes is encoded as arbitrary numbers in Unicorn. This prevents FastText in NodLink from capturing meaningful similarities (e.g., between files with "`.log`" extension), and instead misleads it into learning irrelevant similarities

of digit patterns in randomly assigned numbers. This may cause CSMax to select candidates that provide anomalous out-of-context gadgets, which would be flagged.

Graph-level Black-box. As StreamSpot and Unicorn datasets lack node-level ground truth, Magic performs the detection based on graph-level embeddings, which subsequently prevents our approach from leveraging node embeddings and detection confidence. Therefore, we evaluate our evasion approach against Magic on StreamSopt and Unicorn under black-box settings, where attackers cannot know that the target PIDS is Magic, and thus they may leverage Contorter configured to evade a different PIDS. Specifically, Contorter curates Unicorn and StreamSpot with gadgets based on candidates selected for evading Flash, and then apply Magic on such curated datasets. Our results (shaded cells in Table 4) demonstrate the effectiveness of Contorter in reducing recall by an average of 16%, despite our lack of knowledge about the deployed PIDS.

4.2. Comparison to Baseline

To answer RQ2, we compare Contorter in black-box and white-box settings with two variants of an existing evasion approach [11], both operating in white-box settings. The first variant, Structural Modification (StructMod), follows the open-source robustness evaluation code of [13], which replicates edges around malicious nodes and connects them to the same set of randomly selected benign nodes. The semantic information of this modification (i.e., paths and types of newly added neighbors and edges) is not reflected in the updated node features. The second variant, Structural and Semantic Modification (StructSemMod), extends StructMod by updating nodes' features to reflect the semantics of new connections. In both variants, we increased the number of added edges until recall stopped decreasing. We also attempted to compare with ProvNinja [12], but its open-source implementation is incomplete [19] and does not specify the type of gadgets, preventing direct evaluation against semantic-aware PIDSes. To enable a fair comparison, we re-implemented ProvNinja by replacing the rarest edge between a malicious node and its neighbor with the most frequent path between the same pair in the training set. However, this implementation of ProvNinja did not reduce the recall, and thus it is excluded from our results.

Table 5 shows that Contorter reduces recall to 0.08 in white-box setting, compared with 0.85 and 0.43 for StructMod and StructSemMod (about 10 and five times lower), respectively. Moreover, Contorter achieves this level of evasion with approximately six and 25 times fewer added edges than those approaches. Additionally, Contorter reduces the average recall to 0.33 in black-box setting (a 10% greater reduction than the reduction achieved by StructSemMod operating in white-box setting). StructMod causes an increase in recall by 0.13, on average. One reason is that ignoring semantic information and randomly adding edges can create anomalous neighborhoods around malicious nodes, causing them to be flagged by the PIDS even if they were not initially flagged. Theia is the only dataset where all three approaches achieve successful evasion in white-box settings, likely due to its lack of semantic information, while the evasion in StructSemMod requires up to 2.8 times more added edges than those introduced by Contorter white-box (507,203 versus 177,182). However, the stricter assumptions of black-box settings does not allow the evasion on Theia. Both StructMod and StructSemMod fail on Trace even after adding over 24 million edges. This is likely due to the large number of malicious nodes in Trace (68,173), as connecting so many of them to the same set of benign nodes makes the embeddings of those benign nodes appear anomalous.

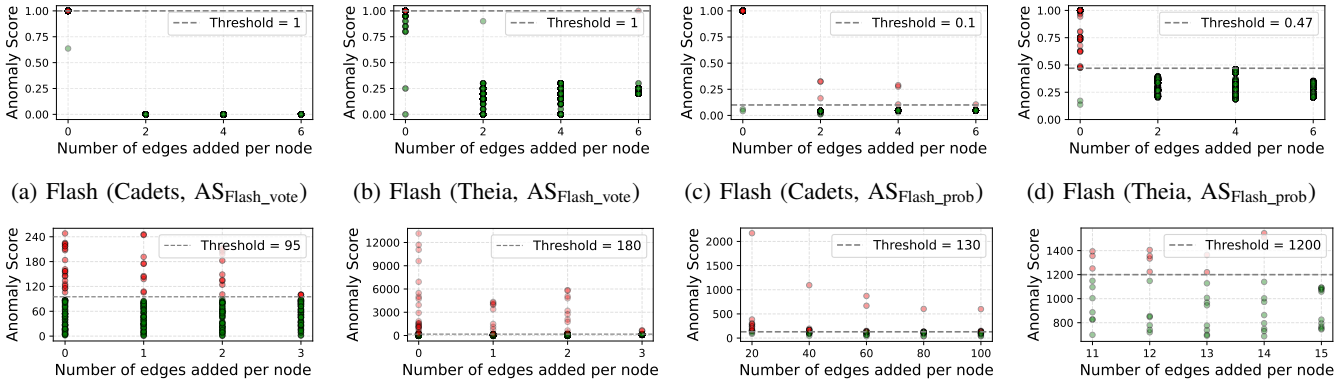
TABLE 5: Effectiveness of Contorter versus two baseline evasions in bypassing Flash [13]. Parentheses indicate the differences from results without evasion. *AEs*, *WB*, and *BB* mean Added Edges, White-box, and Black-box, respectively.

Dataset	StructMod (WB)		StructSemMod (WB)		Contorter (WB)		Contorter (BB)	
	Rec.	# AEs	Rec.	# AEs	Rec.	# AEs	Rec.	# AEs
Cadets	1.00 (0.00)	1,289,467	0.00 (-1.00)	257,175	0.00 (-1.00)	94,390	1.00 (0.00)	77,036
Theia	0.01 (-0.99)	138,312	0.00 (-1.00)	507,203	0.00 (-1.00)	177,182	0.85 (-0.15)	101,276
Trace	1.00 (+0.01)	1,363,497	0.99 (0.00)	24,229,020	0.00 (-0.99)	134,735	0.00 (-0.99)	67,406
StreamSpot	0.99 (+0.03)	2,361,578	0.95 (-0.01)	2,047,360	0.00 (-0.96)	520,126	0.00 (-0.96)	303,246
Unicorn	1.00 (+0.04)	226,235	0.96 (0.00)	225,060	0.08 (-0.88)	145,165	0.00 (-0.96)	133,753
OpTC1	0.97 (+0.73)	245,686	0.17 (-0.07)	3,628	0.19 (-0.05)	3,611	0.20 (-0.04)	2,993
OpTC2	1.00 (+0.81)	443,680	0.28 (+0.09)	8,233	0.19 (0.00)	3,390	0.19 (0.00)	3,717
OpTC3	0.87 (+0.43)	176,155	0.05 (-0.39)	1,296	0.15 (-0.29)	1,378	0.44 (0.00)	1,408
Average	0.85 (+0.13)	780,576	0.43 (-0.29)	3,409,971	**0.08 (-0.64)**	**134,247**	**0.33 (-0.39)**	**86,430**

4.3. Evasion Confidence

To answer RQ3, we measure the changes of anomaly score with the number of gadgets. Lower anomaly scores indicate the reduced confidence of a PIDS to flag a malicious node, and thus, a greater likelihood of evasion under imperfectly specified detection thresholds. Fig. 5 shows anomaly scores for different malicious nodes (circles) as gadgets are added to each node. The definition of anomaly score for three PIDSes and our results are detailed below.

Flash. As Flash adopts several GNN models (Section 3.4), we define two variants of anomaly score as follows: 1) We define $\text{AS}_{\text{Flash\_vote}}$ as $1 - F_{match}$, where F_{match} is the fraction of models that predict the node's true type. 2) We define $\text{AS}_{\text{Flash\_prob}}$ as $1 - P_{\text{max}}$, where P_{max} is the highest probability assigned by any of the models to the node's true type. Fig. 5a and Fig. 5b show $\text{AS}_{\text{Flash\_vote}}$ for Cadets and Theia, respectively. The threshold is set to one, as Flash flags a node when no model predicts its true type. In Cadets, Contorter reduces $\text{AS}_{\text{Flash\_vote}}$ to zero by adding only two edges per malicious node, as most of such nodes originally have very few edges, and thus a small number of added edges can sufficiently shift their embedding. Fig. 5c and Fig. 5d show that $\text{AS}_{\text{Flash\_prob}}$ decreases as the number of added edges grows for both datasets. In Cadets, changes of $\text{AS}_{\text{Flash\_prob}}$ is similar for different malicious nodes (indi-



(a) Flash (Cadets, $AS_{\text{Flash\_vote}}$) (b) Flash (Theia, $AS_{\text{Flash\_vote}}$) (c) Flash (Cadets, $AS_{\text{Flash\_prob}}$) (d) Flash (Theia, $AS_{\text{Flash\_prob}}$)

(e) Magic (Cadets) (f) Magic (Theia) (g) NodLink (Cadets) (h) NodLink (Theia)

Figure 5: The effect of evasion on the detection confidence (lower anomaly scores indicate greater impact of Contorter). Red and green circles represent the anomaly scores of malicious nodes flagged and not flagged by the PIDS, respectively. Darker points correspond to overlapping circles.

cated by many overlapping circles), as malicious nodes have similar semantics in this dataset.

Magic. As a smaller Euclidean distance from the benign cluster indicates the lower confidence of Magic in flagging a malicious node, we consider this distance as the anomaly score AS_{Magic} for each node. Fig. 5e and Fig. 5f show AS_{Magic} for Cadets and Theia, respectively. AS_{Magic} is decreased to below the detection threshold for most (99.8%) malicious nodes by adding fewer edges compared to Flash. This is due to the more fine-grained classification in Flash (six classes of system entities) compared with Magic (one cluster of benign nodes). Moreover, higher average AS_{Magic} values in Theia is due to the larger initial distance of malicious nodes from the benign cluster than in Cadets.

NodLink. As a smaller reconstruction error indicates the lower confidence of NodLink in flagging a malicious node, we consider this error as anomaly score $AS_{NodLink}$. Fig. 5g and Fig. 5h show $AS_{NodLink}$ for Cadets and Theia, respectively. $AS_{NodLink}$ falls below the detection threshold for all malicious nodes in Theia with only 15 added edges per node, while the highest evasion level for Cadets requires about 100 added edges per malicious node. One reason is that malicious process nodes (on which NodLink focuses) in Cadets have around six times more edges than those in Theia, so more gadgets are needed to sufficiently shift the embeddings of malicious nodes in Cadets.

4.4. Ablation Study

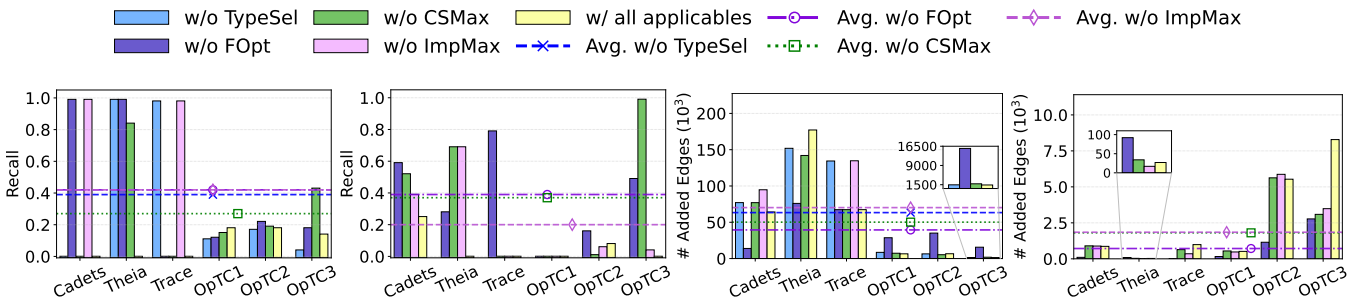
To answer RQ4, we remove one module at a time and measure its impact on recall and the number of added edges. Fig. 6 illustrates the ablation results when evading Flash and NodLink, and the impact of each module is discussed below.

TypeSel. Removing TypeSel increases the average recall by 39%, as its removal may lead to making the context of a malicious node similar to that of a candidate with a different type (e.g., making the context of a process similar to a file), which is anomalous. TypeSel has a greater impact on Theia and Trace as their smaller inter-cluster distances (0.36 and 0.44 compared to 0.52 in Cadets) may cause CSMax to select a node of a different type without TypeSel. For NodLink, TypeSel is not applied, as NodLink embeds only process nodes, which restricts the candidates to processes.

FOpt. Removing FOpt increases the average recall by 0.42 and 0.34 for Flash and NodLink, respectively. This is mainly due to not adding a sufficient number of edges to malicious nodes when FOpt is omitted. Specifically, as malicious nodes typically have few edges, they have a higher cosine similarity to candidates with smaller number of edges. Therefore, removing FOpt causes CSMax to select such candidates, and subsequently, leads to replicating fewer edges around the malicious nodes, which is not sufficient for shifting their embedding toward the benign clusters.

CSMax. Removing CSMax increases the average recall to 0.27 and 0.37 for Flash and NodLink, respectively. Additionally, ignoring contextual similarity may lead to selecting candidates with more edges, which causes a larger number of gadgets (e.g., 13,000 more edges for Cadets in Flash). In contrast, CSMax favors candidates with fewer edges, as their embeddings are more similar to those of malicious nodes, which typically have a small number of edges.

ImpMax. In NodLink, removing ImpMax increases the average recall by 0.15, as without ImpMax, most selected candidates have a high reconstruction error. Therefore, making the context of a malicious node similar to such candidates does not sufficiently reduce its reconstruction error. In Flash, ImpMax is critical for evasion on Cadets and Trace, where clusters of different node types are more distant, and without ImpMax, Contorter may select candidates near the boundary (rather than closer to the center) of the cluster corresponding to the true type of the malicious node. As a result, making the malicious node's context similar to such candidates fails to shift its embedding sufficiently toward that cluster.



(a) Flash, Recall (b) NodLink, Recall (c) Flash, # of edges (d) NodLink, # of edges

Figure 6: Ablation results (larger values for the removed modules indicate their greater impact on evasion success). For Flash, ImpMax is applied only to Cadets and Trace. TypeSel is omitted for NodLink as it focuses only on process nodes.

4.5. Attack Demonstration

To answer RQ5, we investigate whether we can successfully trigger the gadgets obtained by Contorter for OpTC Malicious Upgrade attack (Section 2.1) on our Windows testbed as the victim host. To account for cases where attackers may not be able to train the model based on the logs collected from the victim host, in this experiment, Contorter identifies the contextually similar candidate nodes based on the PIDS model [13] trained on OpTC public dataset and obtains the candidate's connected edges as gadgets. Next, to improve plausibility, OccVer verifies that gadgets are observed in historical logs, which discards many that are not semantically meaningful in practice. Finally, the events corresponding to the verified gadgets are triggered. As indicated by [3], the attacker can synthesize and trigger the gadgets directly (not through the execution path of any application). Accordingly, we simulate the evasion using a Python script that triggers the gadgets on our testbed. More specifically, our script processes gadgets provided in the form of triplets *(actor, action, object)*, executes them as corresponding system-level operations (e.g., file read/write, process execution, or library loading), and logs any failed gadgets.

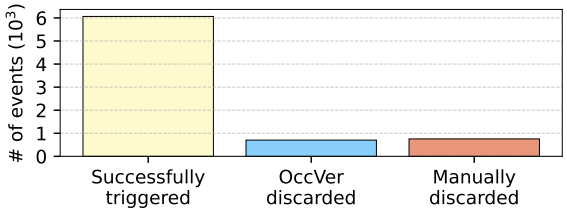
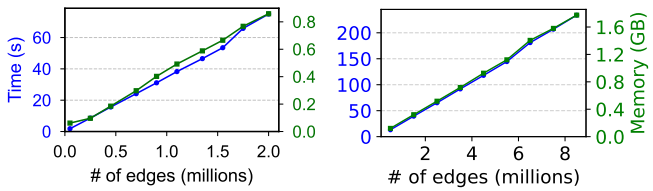


Figure 7: Evaluating the execution success of gadgets.

Fig. 7 shows that 80.6% of the gadgets generated based on the selected candidates were successfully executed on our testbed, which demonstrates the effectiveness of our context-aware design in producing plausible gadgets. About 10% of gadgets are discarded by OccVer as their referenced file paths were missing in logs of the victim system (i.e., our testbed), which is expected since the gadgets were obtained from logs collected on different operating systems and host configurations (e.g., `PSMACHINE.DLL` did not exist on our testbed). The remaining 10% of gadgets correspond to network communications with IP addresses that no longer exist or are reachable, and are thus manually discarded.

4.6. Scalability

To answer RQ6, this section evaluates the scalability of Contorter on provenance graphs with larger number of edges and malicious nodes. To this end, we apply Contorter on subsets of the Cadets and Theia datasets with an increasing number of events, where each subset includes all preceding ones. We conduct this experiment for evading [13] on Cadets, which requires all modules including ImpMax, and on Theia, which has the largest number of edges. The processing time represents the total elapsed duration for each subset including computation and I/O operations. We also measure the maximum GPU memory usage of our approach.



(a) Cadets (b) Theia

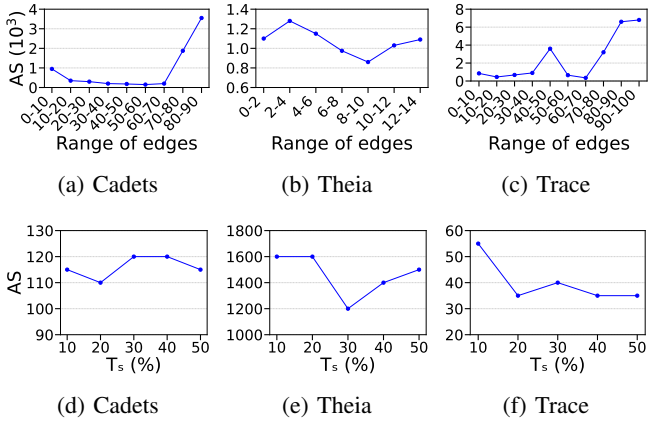
Figure 8: The processing time of Contorter (blue) and its memory usage (green) for Cadets and Theia.

Fig. 8 shows our results. While the size of the provenance graph increases from 50,000 and 500,000 to around two and eight million edges in Cadets and Theia, respectively, the processing time of Contorter remains under four minutes in both datasets. Upon examining our results, we observe that over 94% and 96% of the total processing time in Cadets and Theia, respectively, corresponds to constructing the curated provenance graph and evaluating evasion as performed by the target PIDS. This implies that the overhead

for evasion may largely depend on the graph representation learning technique used by the PIDS itself. For ImpMax (applicable to Cadets), our implementation is optimized to load the provenance graph and compute the confidence scores once, and reuse the results, instead of repeating these steps for obfuscating each malicious node. The results show that the memory consumption of Contorter grows almost linearly with the size of the provenance graph, and it remains around 0.8 and 1.6 gigabyte in the largest subset of Cadets and Theia, respectively.

4.7. Optimal Parameters for Candidate Selection

This section answers RQ7 by identifying the optimal values for two parameters of our approach (i.e., acceptable range for the number of candidates' edges, and the percentage of candidates provided by CSMax to ImpMax (T_s). We conduct this experiment based on three datasets in evading NodLink, as it applies ImpMax to all three datasets.



(a) Cadets (b) Theia (c) Trace

(d) Cadets (e) Theia (f) Trace

Figure 9: Effectiveness of Contorter across various ranges for edges in FOpt (top) and percentages of candidates provided to ImpMax (bottom). *AS* means anomaly score.

Figs. 9a, 9b, and 9c present our results on the three datasets across different acceptable ranges for the number of edges per candidate, with $T_s = 10\%$. The anomaly score decreases with the number of added edges, but its reduction eventually stops after a certain point, since excessive connectivity makes the node appear anomalous. This shows that the careful selection of candidates (and their gadgets) can eliminate the need for adding numerous edges, which would be anomalous and more noticeable to the analyst. Figs. 9d, 9e, and 9f show that the anomaly score reduces as the value of T_s grows, while it starts to increase after providing around 25% of the top ranked candidates to ImpMax. This further highlights the importance of focusing on candidates that are contextually similar to malicious nodes, as they provide the most effective edges for embedding modifications.

5. Related Work

Early intrusion detection systems typically monitor sequences of system calls to detect anomalous behavior [1], [2], [38], [39]. However, such IDSes can be evaded by attackers obfuscating malicious activities among benign-looking gadget system calls [3], [4], [40]. Provenance-based IDSes have emerged as a promising approach offering enhanced robustness and have gained significant attention in the literature [41], [42]. Rule-based PIDSes are suitable for detecting known malicious activities [43], [44], [45], [46], [47], and they are prone to high rates of false positives [11]. Many anomaly-based PIDSes focus on flagging suspicious subgraphs or paths [8], [9], [10], [48], [49], while the coarse granularity of detection makes attack reconstruction challenging and can be more vulnerable to evasion attacks [11]. Recent PIDSes provide more fine-grained detection by flagging suspicious nodes or edges (as summarized in Table 8). Edge-level PIDSes (e.g., [19], [22], [30], [50]) provide more fine-grained detection though with lower efficiency [13]. Node-level PIDSes apply natural language processing techniques to embed causal relationships and entity type of nodes (e.g., [14], [17], [51]) or entity path/name information (e.g., [13], [15], [16], [22]) and attempt to detect the deviation of nodes' embeddings through various graph representation learning approaches or variational autoencoder. The fine-grained and enriched embedding of these PIDSes enhances their detection performance and robustness. Nevertheless, Contorter enables evading even such advanced PIDSes by leveraging the contextual similarities as a guide for generating gadgets.

Recent literature studies the evasion of provenance-based solutions [11], [12], [20], [52] (as summarized in Table 9). Goyal et al. [11] show that attaching frequent benign events can evade graph-level PIDSes by reducing the graph anomaly score. As demonstrated by [13] and [14], capturing the semantic and causal dependencies makes advanced node-level PIDSes robust against such evasion approaches. Additionally, the authors in [20] propose attacks that mislead provenance-based investigation solutions into analyzing benign activity instead of the real attack. The authors in [12] propose ProvNinja, which replaces rare malicious events with gadgets and relies on the attacker to confirm both the success of the original attack (despite the replacement) and the contextual plausibility of gadgets. However, depending solely on human verification is error-prone and may incur significant delay [12], during which the system may transition to new states and thus undermine the relevance of the verification. In contrast, Contorter targets PIDSes that perform detection at a lower granularity by leveraging nodes' contextual information for generating gadgets without breaking the attack sequence. This design enhances evasion and reduces the reliance on manual verification of stealth and success of the attack.

6. Discussion

Applicability and Extensions. First, our approach can potentially be applied to i) other node-level PIDSes with an initial effort of adapting to their provenance construction and embedding mechanisms for measuring vector distances. ii) edge-level PIDSes by extending our candidate selection

approach to identify edges with the same type of (*source, event, destination*) as the malicious edge and a similar embedding. iii) graph-based PIDSes and sequence-based IDSes by curating audit logs based on the embeddings obtained from node-level PIDSes. Second, although we have focused on evading embedding-based detections, our approach can be potentially extended through integration with other evasion approaches to handle PIDSes that leverage multiple detection criteria. For instance, to evade Kairos [22] leveraging both embeddings and raw frequency of nodes, we can integrate our approach with [12] to obfuscate suspicious edges and replace rare nodes with frequent ones. Third, while leveraging the obtained fine-grained embeddings is the key strengths of Contorter, it also means a unique reliance on the quality of those embeddings [53]. For instance, Contorter performs better where clusters of different node types are formed farther from each other, which allows selecting a candidate that can sufficiently shift the embedding of the malicious node toward the intended area in the vector space. Fourth, to improve the effectiveness of Contorter in black-box settings, where the target PIDS model is unknown, an interesting future direction is using ensemble learning for candidate selection. Fifth, extending Contorter to automatically identify optimal ranges based on attack characteristics (e.g., edge count and entity type) and typical behavior of the system is another future direction.

Threats to Validity. Similar to prior works [11], [12], our experiments mostly assume that attackers can obtain or infer the logs and the target PIDS, e.g., through common reconnaissance techniques (Section 2.3). Though, our black-box evaluation (Section 4.1.1 and 4.1.2) and attack demonstration (Section 4.5) show that even without complete knowledge about the PIDS, evasion can achieve partial success. Finally, Contorter focuses on reducing reliance on attackers' expertise and manual effort rather than completely replacing them. Considering contextual relevance for generating gadgets (via TypeSel and CSMax) and discarding previously unseen ones (via OccVer) reduce the likelihood of gadgets that are not executable in practice, while executability can be further ensured through manual inspection or by leveraging expert-provided criteria (e.g., excluding gadgets requiring privilege escalation proposed in [12]). Incorporating a refinement stage after OccVer to further filter gadgets based on these criteria is a future direction.

7. Conclusion

We presented Contorter, an evasion framework that handles node-level PIDSes. We described the methodology and application of Contorter to different node-level PIDSes. Contorter identifies a candidate node that is contextually similar to each malicious node and is labeled benign with high confidence by the target PIDS. It then replicates the selected candidate's edges around the malicious node to further align their contexts and hide the malicious node from the PIDS. Our implementation and evaluation results based on four PIDSes and public datasets demonstrated the efficacy of Contorter in evading advanced PIDSes.

8. Acknowledgment

We thank the anonymous shepherd and reviewers for their valuable comments. This work was supported by the Natural Sciences and Engineering Research Council of Canada Discovery Grant and University of Manitoba.

9. Ethics Considerations

We have considered the risks and benefits of our approach: similar to other studies on evasion attacks [11], [12], [20], our approach focuses on uncovering the blind spots of provenance-based solutions discussed in the academic literature with the goal of informing efforts on improving robustness. To ensure protecting sensitive information and ethical data utilization, all datasets adopted in our research are publicly available. Finally, all of our experiments are conducted on local controlled resources.

References

[1] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for unix processes," in *Proceedings 1996 IEEE symposium on security and privacy*. IEEE, 1996, pp. 120–128.

[2] N. Habra, B. L. Charlier, A. Mounji, and I. Mathieu, "Asax: Software architecture and rule-based language for universal audit trail analysis," in *European Symposium on Research in Computer Security*. Springer, 1992, pp. 435–450.

[3] D. Wagner and P. Soto, "Mimicry attacks on host-based intrusion detection systems," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002, pp. 255–264.

[4] K. M. Tan, K. S. Killourhy, and R. A. Maxion, "Undermining an anomaly-based intrusion detection system using common exploits," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2002, pp. 54–73.

[5] H. G. Kayacik and A. N. Zincir-Heywood, "Mimicry attacks demystified: What can attackers do to evade detection?" in *2008 Sixth Annual Conference on Privacy, Security and Trust*. IEEE, 2008, pp. 213–223.

[6] X. Han, T. Pasquier, and M. Seltzer, "Provenance-based intrusion detection: opportunities and challenges," in *10th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2018)*, 2018.

[7] T. Pasquier, X. Han, M. Goldstein, T. Moyer, D. Eyers, M. Seltzer, and J. Bacon, "Practical whole-system provenance capture," in *Proceedings of the 2017 symposium on cloud computing*, 2017, pp. 405–418.

[8] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Chen, W. Cheng, C. A. Gunter *et al.*, "You are what you do: Hunting stealthy malware via data provenance analysis." in *NDSS*, 2020.

[9] E. Manzoor, S. M. Milajerdi, and L. Akoglu, "Fast memory-efficient anomaly detection in streaming heterogeneous graphs," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1035–1044.

[10] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "Unicorn: Runtime provenance-based detector for advanced persistent threats," in *Network and Distributed System Security Symposium (NDSS)*, 2020.

[11] A. Goyal, X. Han, G. Wang, and A. Bates, "Sometimes, you aren't what you do: Mimicry attacks against provenance graph host intrusion detection systems," in *30th Network and Distributed System Security Symposium*, 2023.

[12] K. Mukherjee, J. Wiedemeier, T. Wang, J. Wei, F. Chen, M. Kim, M. Kantarcioglu, and K. Jee, "Evading Provenance-Based ML detectors with adversarial system actions," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 1199–1216.

[13] M. U. Rehman, H. Ahmadi, and W. U. Hassan, "Flash: A comprehensive approach to intrusion detection via provenance graph representation learning," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 3552–3570.

[14] Z. Jia, Y. Xiong, Y. Nan, Y. Zhang, J. Zhao, and M. Wen, "MAGIC: Detecting advanced persistent threats via masked graph representation learning," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 5197–5214.

[15] S. Li, F. Dong, X. Xiao, H. Wang, F. Shao, J. Chen, Y. Guo, X. Chen, and D. Li, "Nodlink: An online system for fine-grained apt attack detection and investigation," in *Network and Distributed System Security Symposium (NDSS)*, 2023.

[16] A. Goyal, G. Wang, and A. Bates, "R-caid: Embedding root cause analysis within provenance-based intrusion detection," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 3515–3532.

[17] S. Wang, Z. Wang, T. Zhou, H. Sun, X. Yin, D. Han, H. Zhang, X. Shi, and J. Yang, "Threatrace: Detecting and tracing host-based threats in node level through provenance graph learning," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 3972–3987, 2022.

[18] J. Ren and R. Geng, "Provenance-based APT Campaigns Detection via Masked Graph Representation Learning," *Computers & Security*, vol. 148, p. 104159, 2025.

[19] T. Bilot, B. Jiang, Z. Li, N. El Madhoun, K. Al Agha, A. Zouaoui, and T. Pasquier, "Sometimes Simpler is Better: A Comprehensive Analysis of State-of-the-Art Provenance-Based Intrusion Detection Systems," in *34th USENIX Security Symposium (USENIX Security 25)*, 2025, pp. 7193–7212.

[20] A. Sang, Y. Wang, L. Yang, J. Jia, and L. Zhou, "Obfuscating provenance-based forensic investigations with mapping system meta-behavior," in *Proceedings of the 27th International Symposium on Research in Attacks, Intrusions and Defenses*, 2024, pp. 248–262.

[21] Z. Li, Y. Wei, X. Shen, L. Wang, Y. Chen, H. Xu, S. Ji, F. Zhang, L. Hou, W. Liu *et al.*, "Marlin: Knowledge-driven analysis of provenance graphs for efficient and robust detection of cyber attacks," *arXiv preprint arXiv:2403.12541*, 2024.

[22] Z. Cheng, Q. Lv, J. Liang, Y. Wang, D. Sun, T. Pasquier, and X. Han, "Kairos: Practical intrusion detection and investigation using whole-system provenance," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 3533–3551.

[23] DARPA, "DARPA Engagement 3 (E3) - Transparent Computing," https://github.com/darpa-i2o/Transparent-Computing/blob/master/README-E3.md, 2017, accessed: 2025-11-13.

[24] D. Jatnika, M. A. Bijaksana, and A. A. Suryani, "Word2vec model analysis for semantic similarities in english words," *Procedia Computer Science*, vol. 157, pp. 160–167, 2019.

[25] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[26] G. Guo, H. Wang, D. Bell, Y. Bi, and K. Greer, "KNN Model-based Approach in Classification," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2003, pp. 986–996.

[27] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the association for computational linguistics*, vol. 5, pp. 135–146, 2017.

[28] MITRE Corporation, "MITRE ATT&CK," https://attack.mitre.org/techniques/T1592/, 2025, accessed: 2025-09-15.

[29] S. Jana and V. Shmatikov, "Memento: Learning secrets from process footprints," in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 143–157.

[30] B. Jiang, T. Bilot, N. El Madhoun, K. Al Agha, A. Zouaoui, S. Iqbal, X. Han, and T. Pasquier, "Orthrus: Achieving high quality of attribution in provenance-based intrusion detection systems," in *Security Symposium (USENIX Sec'25). USENIX*, 2025.

[31] A. Bates, D. J. Tian, K. R. Butler, and T. Moyer, "Trustworthy Whole-System Provenance for the Linux Kernel," in *24th USENIX Security Symposium (USENIX Security)*, 2015, pp. 319–334.

[32] R. Paccagnella, P. Datta, W. U. Hassan, A. Bates, C. Fletcher, A. Miller, and D. Tian, "Custos: Practical Tamper-evident Auditing of Operating Systems Using Trusted Execution," in *Network and distributed system security symposium*, 2020.

[33] R. Paccagnella, K. Liao, D. Tian, and A. Bates, "Logging to the danger zone: Race condition attacks and defenses on system audit frameworks," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1551–1574.

[34] T. Mikolov, W.-t. Yih, and G. Zweig, "Linguistic Regularities in Continuous Space Word Representations," in *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, 2013, pp. 746–751.

[35] DARPA and Five Directions, "OpTC Dataset - Operational Transparent Computing," https://github.com/FiveDirections/OpTC-data, 2022, accessed: 2025-04-18.

[36] L. Wang, X. Shen, W. Li, Z. Li, R. Sekar, H. Liu, and Y. Chen, "Incorporating gradients to rules: Towards lightweight, adaptive provenance-based intrusion detection," *arXiv preprint arXiv:2404.14720*, 2024.

[37] B. J. Wilson and A. M. Schakel, "Controlled experiments for word embeddings," *arXiv preprint arXiv:1510.02675*, 2015.

[38] H. S. Vaccaro, "Detection of anomalous computer session activity," Los Alamos National Laboratory (LANL), Los Alamos, NM (United States), Tech. Rep., 1988.

[39] A. Ghosh, A. Schwartzbard, and M. Schatz, "Learning program behavior profiles for intrusion detection," in *1st Workshop on Intrusion Detection and Network Monitoring (ID 99)*, 1999.

[40] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna, "Automating mimicry attacks using static binary analysis," in *USENIX Security Symposium*, vol. 14, 2005, pp. 11–11.

[41] M. Zipperle, F. Gottwalt, E. Chang, and T. Dillon, "Provenance-based Intrusion Detection Systems: A Survey," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–36, 2022.

[42] W. Cheng, T. Zhu, C. Xiong, H. Sun, Z. Wang, S. Jing, M. Lv, and Y. Chen, "SoK: Knowledge is All You Need: Accelerating Last Mile Delivery for Automated Provenance-based Intrusion Detection with LLMs," *arXiv preprint arXiv:2503.03108*, 2025.

[43] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "Holmes: real-time apt detection through correlation of suspicious information flows," in *2019 IEEE symposium on security and privacy (SP)*. IEEE, 2019, pp. 1137–1152.

[44] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, "Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 1795–1812.

[45] P. Gao, X. Xiao, Z. Li, F. Xu, S. R. Kulkarni, and P. Mittal, "AIQL: Enabling Efficient Attack Investigation from System Monitoring Data," in *2018 USENIX Annual Technical Conference (USENIX ATC)*, 2018, pp. 113–126.

[46] P. Gao, X. Xiao, D. Li, Z. Li, K. Jee, Z. Wu, C. H. Kim, S. R. Kulkarni, and P. Mittal, "SAQL: A stream-based query system for Real-Time abnormal system behavior detection," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 639–656.

[47] W. U. Hassan, A. Bates, and D. Marino, "Tactical provenance analysis for endpoint detection and response systems," in *2020 IEEE symposium on security and privacy (SP)*. IEEE, 2020, pp. 1172–1189.

[48] X. Han, T. Pasquier, T. Ranjan, M. Goldstein, and M. Seltzer, "FRAPpuccino: Fault-detection through runtime analysis of provenance," in *9th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 17)*, 2017.

[49] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates, "Nodoze: Combatting threat alert fatigue with automated provenance triage," in *network and distributed systems security symposium*, 2019.

[50] J. Zengy, X. Wang, J. Liu, Y. Chen, Z. Liang, T.-S. Chua, and Z. L. Chua, "Shadewatcher: Recommendation-guided cyber threat analysis using system audit records," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 489–506.

[51] F. Yang, J. Xu, C. Xiong, Z. Li, and K. Zhang, "PROGRAPHER: An anomaly detection system based on provenance graph embedding," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 4355–4372.

[52] Z. Li, R. Yang, Q. A. Chen, and Y. Chen, "Mimic the whole attack chain: A first look at evasion against provenance graph based detection," in *Annual Computer Security Applications Conference (ACSAC)*, 2020.

[53] K. R. Shahapure and C. Nicholas, "Cluster Quality Analysis Using Silhouette Score," in *2020 IEEE 7th international conference on data science and advanced analytics (DSAA)*. IEEE, 2020, pp. 747–748.

Appendix A. Node-level Detection Visualization

Fig. 10 illustrates how an exemplar node-level PIDS, Flash [13], detects an anomalous process node. Flash captures the semantic and causal dependencies of nodes as their vector representation such that nodes corresponding to the same type of system entity would have similar representations (as illustrated on the left side of Fig. 10). Accordingly, when the semantic and causal dependencies of a process node is such that its embedding is closer to that of socket nodes, its predicted type would be Socket (as illustrated on the right side of Fig. 10). Such a mismatch between predicted and true type (i.e., socket and process, respectively) is flagged by Flash, as the context of the node is anomalous for its node type.

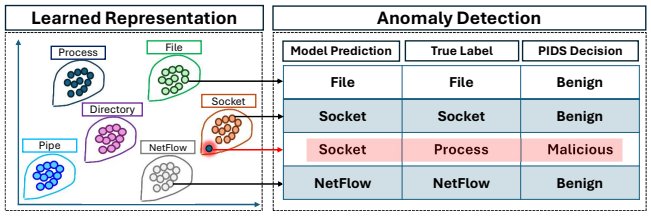


Figure 10: Illustrating the detection of an example anomalous process node by Flash.

Appendix B. Dataset Statistics

Cadets captures system activity on a FreeBSD host, with over 357K nodes and 2M edges. Theia and Trace are Linux-based (Ubuntu 12.04 and 14.04, respectively). Theia contains around 9.4M edges and 344K nodes, and Trace contains about 2M edges and 1M nodes. All datasets include complex multi-stage attack scenarios involving implants like Drakon and MicroAPT, privilege escalations, and lateral movement.

Darpa OpTC includes three attack scenarios. The first scenario, *Plain PowerShell Empire*, involves an initial foothold using PowerShell, privilege escalation, credential dumping via Mimikatz, lateral movement through WMI, and multi-host propagation. The second attack, *Custom PowerShell Empire*, combines phishing-based delivery of macro-less documents, command-and-control pivoting, automated domain enumeration with DeathStar, and data exfiltration via RDP and Netcat across multiple hosts. The third, *Malicious Upgrade*, leverages a compromised software update mechanism to deploy a Meterpreter payload, which allows system-level access, persistence via registry autorun, credential extraction, and timestomping for anti-forensics.

TABLE 6: Statistics of datasets with node-level ground truth.

Dataset	# All Nodes	# Edges	System	# Malicious nodes
Cadets	357,174	2,097,882	FreeBSD	12,852
Theia	344,768	9,426,694	Linux	25,359
Trace	1,211,451	2,255,113	Linux	68,173
Fivedirections	298,333	798,945	Windows	330
OpTC1	83,943	169,555	Windows 10	181
OpTC2	304,387	582,194	Windows 10	410
OpTC3	104,628	206,273	Windows 10	62

TABLE 7: Statistics of datasets with graph-level ground truth.

Dataset	# Graphs	Avg. # Nodes	Avg. # Edges	Anomaly Type
StreamSpot	600	8925	185998	APT
Unicorn	150	18515	53559	Malware

Appendix C. Comparison of Existing Approaches

Table 8 summarizes the characteristics of recent PIDSes focusing on fine-grained detection, and Table 9 shows the characteristics of different approaches evading provenance-based solutions in comparison with Contorter.

TABLE 8: Summary of characteristics of recent PIDSes.

	Flash [13]	Magic [14]	Threatrace [17]	NodLink [15]	Kairos [22]	R-Caid [16]	Captain [36]	Prographer [51]	Orthrus [30]
Node-level	✓	✓	✓	✓	✓	✓	✗	✓	✓
Graph-level	✓	✓	✓	✗	✓	✓	✗	✓	✓
Edge-level	✗	✗	✗	✗	✓	✗	✓	✗	✓
Representation Learning	✓	✓	✓	✓	✓	✓	✗	✓	✓
Feature: Entity Type	✓	✓	✓	✗	✓	✓	✗	✓	✓
Feature: Edge Type	✓	✓	✓	✗	✓	✓	✓	✓	✓
Feature: Entity Path	✓	✗	✗	✓	✓	✓	✓	✗	✓
Open-source	✓	✓	✓	✓	✓	✗	✓	✗	✓

TABLE 9: Comparison of evasion approaches with Contorter (ours).

Criterion	PAIOF [20]	Goyal et al. [11]	ProvNinja [12]	Contorter
Preserving Attack Chain	✓	✓	✗	✓
Unified Architecture	✗	✗	✓	✓
Context-Aware Gadget	✗	✗	✗	✓
Applied to Node-level PIDS	✗	✗	✓	✓

Appendix D. Processing Time of Different Modules

Table 10 shows the processing time of each module for three PIDSes on Trace dataset (with the largest number of malicious nodes). We can see that the total processing time even for this dataset is below two minutes for Flash and NodLink. CSMax is computationally expensive for Magic for three reasons as follows: 1) the embedding dimension in Magic is larger than in Flash (64 vs. 30); 2) the average number of candidates per malicious node in Magic is much larger (3,277,340 in Magic vs. 1,139,744 in Flash) as Flash classifies nodes into smaller categories based on their types (rather than the larger category of all benign nodes as in Magic); 3) the ground truth for Magic is much larger than for NodLink, since NodLink captures only malicious processes (which represent only a small fraction of the 68k malicious nodes in Flash and Magic). ImpMax module in Flash has a longer processing time than in other PIDSes, as it involves loading all nodes (1.2 million) and their edges (2.25 million) to 20 GNN models in Flash, based on which ImpMax calculates the classification confidence. The processing time of GadRet is longer in Magic as it requires rebuilding the internal adjacency matrices in Deep Graph Library (DGL).

TABLE 10: Execution time of our different modules.

Module	Flash Time (sec)	Magic Time (sec)	NodLink Time (sec)
TypeSel	0.0808	2.4100	–
FOpt	0.0652	141.56	0.0173
CSMax	7.3264	2060.9	0.5164
GadRet	0.0000	90.751	0.0000
OccVer	0.0000	0.0000	0.0000
ImpMax	0.3712	-	0.0344
EVer	79.103	635.50	0.8100
Total	**86.946**	**2931.1**	**1.3781**

Appendix E. Meta-review

The following meta-review was prepared by the program committee for the 2026 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

E.1. Summary

This paper presents Contorter, an evasion framework for node-level provenance-based intrusion detection systems (PIDSes). The key idea is to modify the local context of malicious nodes so that their learned representations move closer to benign clusters, making them harder for graph-based detectors to identify. The attack selects contextually similar benign nodes and introduces carefully chosen gadget edges that preserve plausibility while shifting embeddings. The paper evaluates the approach against four representative PIDSes on multiple provenance datasets and shows substantial reductions in detection recall, along with comparisons to prior attacks, ablation studies, and an overhead analysis.

E.2. Scientific Contributions

- Creates a New Tool to Enable Future Science.
- Provides a Valuable Step Forward in an Established Field.

E.3. Reasons for Acceptance

1) While evasion of provenance-based intrusion detection has been studied before, this work advances the state-of-the-art by proposing a context-aware embedding-shift strategy. This gives the attack a clearer semantic basis and leads to stronger evasion with fewer added edges.
2) The core technical insight is interesting and well motivated. The paper identifies that node-level PIDSes rely heavily on contextual embeddings, and exploits that dependence by reshaping malicious nodes toward benign neighborhoods of the same type.
3) By organizing the attack into modular stages and evaluating it across several PIDSes, the work helps clarify where current node-level provenance detectors are brittle and what kinds of contextual manipulations are most damaging. That makes the paper useful both as an attack paper and as a stress test for future defenses.

E.4. Noteworthy Concerns

1) Although the framework is designed to generate plausible attack perturbations, the paper provides only partial validation of their practical executability. The paper does not provide a methodology to verify that the gadget edges can be executed under real system constraints, such as permissions, process behavior, and file-access feasibility. Thus, attacks generated by Contorter

and other systems might not be possible, and the current evaluation methodology and metrics may not accurately reflect the attack's performance.

Appendix F. Response to the Meta-Review

We thank the reviewers and the shepherd for their valuable comments. We note that considering the contextual relevance for generating gadgets (via our TypeSel and CS-Max modules) and discarding previously unseen ones (via our OccVer module) reduces the likelihood of gadgets that are not executable in practice (Section 4.5). Executability can be further improved through manual inspection of the gadgets or by leveraging expert-provided criteria (e.g., those proposed in [12]).