Sometimes, You Aren't What You Do: Mimicry Attacks against Provenance Graph Host Intrusion Detection Systems

Akul Goyal
University of Illinois at Urbana-Champaign
akulg2@illinois.edu

Xueyuan Han
Wake Forest University
vanbasm@wfu.edu

Gang Wang
University of Illinois at Urbana-Champaign
gangw@illinois.edu

Adam Bates
University of Illinois at Urbana-Champaign
batesa@illinois.edu

***Abstract*—Reliable methods for host-layer intrusion detection remained an open problem within computer security. Recent research has recast intrusion detection as a *provenance graph anomaly detection* problem thanks to concurrent advancements in machine learning and causal graph auditing. While these approaches show promise, their robustness against an adaptive adversary has yet to be proven. In particular, it is unclear if *mimicry attacks*, which plagued past approaches to host intrusion detection, have a similar effect on modern graph-based methods.**

In this work, we reveal that systematic design choices have allowed mimicry attacks to continue to abound in provenance graph host intrusion detection systems (Prov-HIDS). Against a corpus of exemplar Prov-HIDS, we develop evasion tactics that allow attackers to hide within benign process behaviors. Evaluating against public datasets, we demonstrate that an attacker can consistently evade detection (100% success rate) without modifying the underlying attack behaviors. We go on to show that our approach is feasible in live attack scenarios and outperforms domain-general adversarial sample techniques. Through open sourcing our code and datasets, this work will serve as a benchmark for the evaluation of future Prov-HIDS.

I. Introduction

Intrusion detection systems (IDS) are an important reactive security measure that identifies possible ongoing attacks on a host system. With its market size expected to exceed 170 billion dollars [1], cybersecurity is an enormous industry in which intrusion detection is a highly coveted service. Further, IDS provide the initial indicators of compromise used to initiate threat hunting and incident response activities [2, 3, 4, 5, 6, 7, 8]. Due to the difficulty of outright preventing attackers from breaching organizational defenses (e.g., [9, 10]), IDS are a cornerstone of real-world security today.

While IDS typically take two forms, anomaly-based and heuristic rule-based systems, we focus on anomaly-based approaches. In host-based anomaly detection, sequences of low-level events are analyzed to define patterns of typical host activity. For example, in Forrest et al.'s pioneering work [11], each process' behavioral pattern is defined over a sliding window of system calls of some small length k (e.g., k=6 [12]). At runtime, if processes deviate from the set of system call sequences observed when the application was profiled, it is considered an anomaly and an alert is raised. While subsequent IDS would refine this technique and incorporate additional host context (e.g., argument dataflows [13]), the Forrest IDS codifies the general strategy for host-based anomaly detection: monitor a stream of audit events to differentiate typical behaviors from anomalous (potentially malicious) activity.

However, are malicious acts innately distinct from regular activity? This question was raised by Wagner and Soto in their introduction of the *mimicry attack* [14] – due to various imperfections in the IDS' representation of system normality, it is possible for attackers to pattern their actions such that they are indistinguishable from benign processes. For example, a malicious process could defeat the Forrest IDS by executing no-op system calls, preserving the semantics of the attack while simultaneously adhering to system call sequences of approved processes. This finding was repeatedly replicated and extended [11, 12, 15, 16], casting doubt on the practicality of an entire generation of anomaly-based detection systems.

This early intrusion detection work was the forebearer to the explosive popularity of machine learning research seen in the security literature today. In fact, recent advancements in machine learning and system auditing have led researchers to reconceptualize host intrusion detection as a graph learning problem. These "Prov-HIDS" analyze *provenance graphs* – causal dependency graphs that describe the history of system execution – in an attempt to delineate typical from anomalous activity. Through identifying connections between current and historical events in the system, these techniques show promise as a new primitive for intrusion detection. However, at present it is unclear *how*, or even *if*, such approaches are successful in the face of determined, resourceful, and adaptive adversaries.

In this work, we resurrect the foundational challenge of IDS evasion in the context of modern Prov-HIDS. We select five exemplar Prov-HIDS – StreamSpot [17], Unicorn [18], ProvDetector [19],[1] Pagoda [20], and a Full Graph Autoencoder [21] – providing a representative sample of the field. We dissect these systems' inner workings to arrive at an understanding of how each approach sacrifices the historical context of complete provenance graphs to produce efficient and generalizable classification models. We then develop a corpus of candidate *mimicry gadgets* for crafting evasion attacks against these systems.

To evaluate our procedurally-generated mimicry attacks, we make use of the publicly-released StreamSpot [22] and DARPA Transparent Computing [23] datasets. Our findings

[1]Our title is a playful jab at the ProvDetector paper title, "You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis" [19].

Network and Distributed System Security (NDSS) Symposium 2023
27 February - 3 March 2023, San Diego, CA, USA
ISBN 1-891562-83-5
https://dx.doi.org/10.14722/ndss.2023.24207
www.ndss-symposium.org

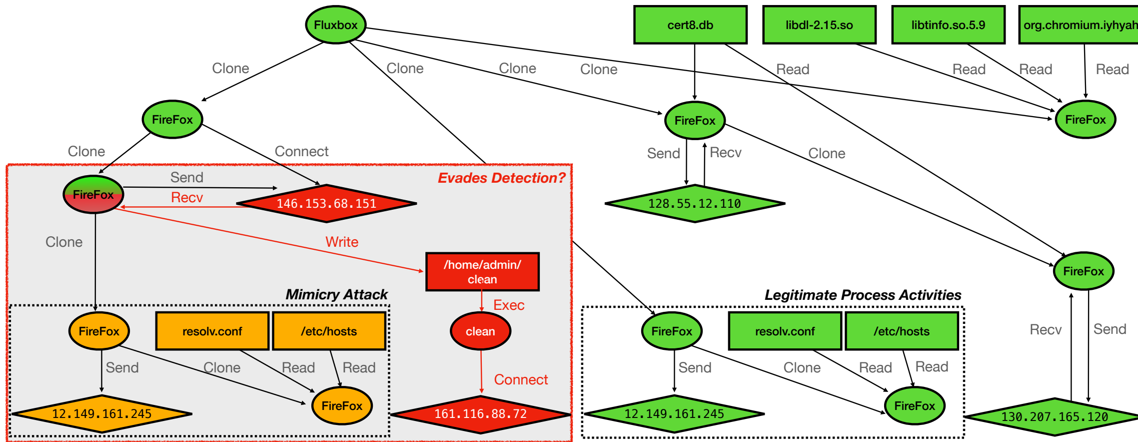


Fig. 1: A provenance graph describing a Firefox backdoor attack using Drakon from a DARPA Transparent Computing engagement. Circles indicate processes, rectangles files, and diamonds network connections. Benign system behaviors are shaded green, while the attack subgraph is red. This work considers Prov-HIDS' resilience to *mimicry attacks* that embed substructures from *legitimate process activities* into the attack subgraph.

show that *out of over 100 attack graph samples, we are able to successfully force misclassification 100% of the time, regardless of the Prov-HIDS under test.* We go on to characterize the cost of evasion in terms of the complexity of the graph transformation, demonstrate the feasibility of our approach by adapting our mimicry strategies to live attack demonstrations, and show that evasion is possible even when multiple Prov-HIDS are employed in an ensemble. Finally, we empirically demonstrate the superiority of our approach over domain-general adversarial sample generation techniques by evaluating RL-S2V [24], a reinforcement learning system: *After seven days of execution in a variety of configurations, RL-S2V failed to produce a single successful evasion sample against the (relatively) simplistic StreamSpot IDS.* In contrast, our methods are highly efficient and reflect testable hypotheses that provide insight into the failure conditions of Prov-HIDS.

Our contributions can be summarized as follows:

- *Sometimes, You Aren't What You Do.* We bring mimicry attacks into the modern era. Surveying five state-of-the-art Prov-HIDS to understand how provenance graphs are encoded for anomaly detection, we propose a battery of novel mimicry attack methodologies.
- *Independent Evaluation of Prov-HIDS.* We conduct an independent evaluation of five exemplar Prov-HIDS. We uncover *systemic vulnerability* to evasion, reaching a 100% success rate against all systems.
- *Open-source Benchmark for Mimicry Attacks.* Our code and data is publicly available [2] as a benchmark suite. Future Prov-HIDS research can leverage our adversarial samples to evaluate system resilience to evasion.

II. Motivating Attack Scenario

To motivate our work, consider the provenance of an intrusion attempt from DARPA Transparent Computing's third adversarial engagement, shown in Fig. 1, which depicts a Firefox backdoor attack. The red subgraph is a simplified version of the original attack. A victim machine running a vulnerable Firefox 54.0.1 unknowingly establishes a connection with a malicious ad server located at 146.153.68.151. The server exploits a backdoor in Firefox and injects a binary executable "Drakon" in its process memory. Drakon subsequently spawns a new process (`/home/admin/clean`) with root privileges that connects to the attacker's server at 161.116.88.72, giving the attacker full access to the victim machine.

As we will soon demonstrate, Prov-HIDS can reliably detect this attack. By learning a representation of the typical behavior of a system (i.e., the green subgraph), Prov-HIDS are able to detect that the attack behavior (the red subgraph) deviates significantly and is thus an anomaly. For example, Han et al.'s Unicorn system [18] may learn that Firefox's child processes are expected to connect to different servers and read from system files containing DNS information (*legitimate process activities*). When the attacker instead writes and executes the file `/home/admin/clean`, the resulting graph structures are highly anomalous, making the attack simple to detect.

But what if the attacker is aware that a Prov-HIDS is monitoring the system? The adversary could then modify their attack subgraph using a *mimicry attack* (the orange subgraph) – while the actual attack logic remains unchanged, the attacker could fork additional processes that resemble legitimate process activities. These activities may be sufficient to confuse the classifier, resulting in a misclassification of the attack. In practice, the simplified mimicry attack depicted in Fig. 1 would not be sufficient to fool state-of-the-art systems. However, the attacker is free to perform an unbounded number of actions within process spaces under their control. They could conceivably integrate many events into their attack behavior, sampling normal activities from the victim system. *Our goal is to determine, through principled analysis, the feasibility and cost of launching mimicry attacks on advanced Prov-HIDS.*

III. Provenance-Based Host Intrusion Detection

A. Provenance Graphs

We define a provenance graph as $G = (V, E)$, where $V = \{v_j\}_{j=1}^{|V|}$ is a set of vertices and $E = \{e_j\}_{j=1}^{|E|}$ is a set of edges. Each vertex in V refers to a concrete system entity that was accessed during the course of a system's execution,

[2] https://bitbucket.org/sts-lab/mimicry-provenance-generator/src/master/

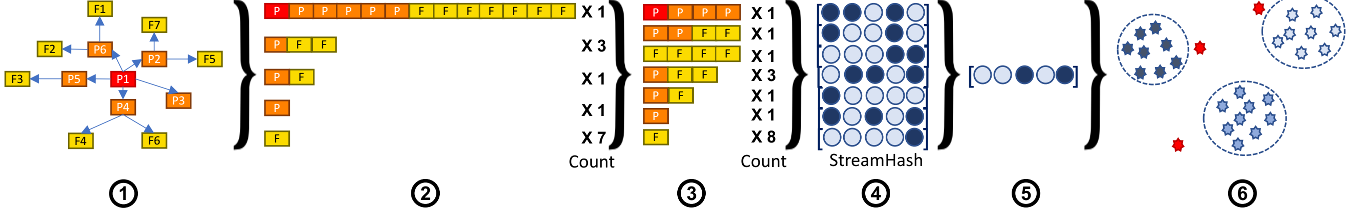


Fig. 2: Overview of the StreamSpot Prov-HIDS. StreamSpot ① takes as input a provenance graph, ② traverses and ③ chunks the neighborhoods of the graph, and ④ encodes each chunk using StreamHash to ⑤ build a final embedding of binary features. StreamSpot ⑥ fits the final embedding into a trained model consisting of benign graph embeddings.

such as a file, a process, a network socket, or inter-process communication. Each edge represents a specific system event that was observed, corresponding to a single system call in the audit log from which the graph is built. A *path* in G is $P(v_s, v_d) = v_s(rel_{si})v_i, \cdots, v_j(rel_{jd})v_d$, where each edge $v_{src}(rel)v_{dst} \in E$ and edges are causally ordered. The length of a path $L(P(v_s, v_d))$ is the number of edges between v_s and v_d along the path. We describe a set of provenance graphs as $\mathcal{G} = \{G_i\}_{i=1}^N$, where $N = |\mathcal{G}|$. An individual graph is therefore $G_i = (V_i, E_i)$, an edge in E_i is e_j^i, and so on.

B. Generic Prov-HIDS Definition

Let each graph G_i in a dataset $\mathcal{G}$ be associated with a label $y_i \in \{0, 1\}$, where 0 is benign and 1 is malicious, such that $\mathcal{G} = \{G_i, y_i\}_{i=1}^{|\mathcal{G}|}$. The objective of a Prov-HIDS $f(G_i)$ is to minimize the loss function $\mathcal{L}$:

$$\mathcal{L} = 1 - \frac{1}{|\mathcal{G}|} \sum_{i=1}^{|\mathcal{G}|} \mathbb{1}(f(G_i) = y_i) \tag{1}$$

where $\mathbb{1}(x) = 1$ if x is true; otherwise, $\mathbb{1}(x) = 0$.

The Prov-HIDS $f(G_i)$ makes a classification decision by encoding *substructures* within the graph G_i and then comparing them to the substructures of a pre-encoded set of benign graphs. An anomaly is raised if G_i's substructures deviate significantly from the known benign substructures:

$$f(G_i) = \mathbb{1}(\mathcal{F}^{\delta}(\mathcal{E}_{\lambda}^{\kappa}(\mathcal{N}_{\beta}^{\gamma}(G_i))) \geq \alpha) \tag{2}$$

$\mathcal{N}_{\beta}^{\gamma}(G_i)$ is a deconstruction function, parameterized by γ and β, that returns a set of substructures $z_j^i \in Z_i$ such that $z_j^i = \mathcal{N}_{\beta}^{\gamma}(v_j^i), \forall v_j^i \in V_i$. [3] γ is a branching factor that determines the size of v_j^i's neighborhood, and β is a depth factor describing the max distance between a node in the substructure and v_j^i.

$\mathcal{E}_{\lambda}^{\kappa}(Z_i)$ is an encoding function that summarizes Z_i into an L-dimensional vector $\mathbf{V}_i$. κ specifies the size of a subset of Z_i used to represent G_i. λ is the embedding function used.

$\mathcal{F}^{\delta}(\mathbf{V}_i)$ is a distance function that compares $\mathbf{V}_i$ against a set of learned graph encodings $\delta = \{\mathbf{V}_{(1)}^p, \cdots, \mathbf{V}_{(n)}^p\}$, which returns the smallest distance between a graph in δ and $\mathbf{V}_i$. α is a distance threshold, under which G_i is considered benign.

System	Function	Method	Learning Task	Code?
StreamSpot [17]	Detect.	Unsup.	Neigh.-based Whole Graph	✔
FRAPpuccino [25]	Detect.	Unsup.	Neigh.-based Whole Graph	✔
Unicorn [18]	Detect.	Unsup.	Neigh.-based Whole Graph	✔
Pagoda [20]	Detect.	Unsup.	Path-based Whole Graph	✔
P-Gaussian [26]	Detect.	Unsup.	Path-based Whole Graph	✖
ProvDetector [19]	Detect.	Unsup.	Path-based Subgraph	✖
PIDAS [27]	Detect.	Unsup.	Path-based Subgraph	✖
SIGL [21]	Detect.	Unsup.	Whole Graph Autoencoder	✖
Hercule [28]	Invest.	Sup.	Log Community Detection	✖
ATLAS [29]	Invest.	Sup.	Log Sequence Modeling	✔
NoDoze [30]	Invest.	Unsup.	Historic Event Analysis	✖
Holmes [31]	Invest.	Unsup.	Historic Event Analysis	✖

TABLE I: Summary of graph learning mechanisms in provenance-based security tools: *Function* refers to the high-level security function of the learning mechanism (*Detect*ion or *Invest*igation); *Method* indicates whether the mechanism is *Unsup*ervised or *Sup*ervised; *Learning Task* describes the classification task being performed by the mechanism; and *Code?* indicates whether the authors have publicly released an implementation of the prototype of their system. Examplar systems surveyed in our paper are shaded in yellow. Note that we focus only on the learning components of these systems, not their end-to-end functionality. We discuss related work more broadly in §VIII.

C. Exemplar System Selection

We ground the evasion techniques we develop in this work based on an in-depth analysis of a set of exemplar Prov-HIDS. Our survey of graph learning mechanisms in provenance-based security tools is shown in Table I. For completeness, we include in the table investigation-oriented tools that are not Prov-HIDS but feature related learning mechanisms. As we can see, the majority of Prov-HIDS decompose the graph into either *neighborhood-based* or *path-based substructures* prior to vectorization, and then attempt to define normal behavior with respect to either *whole graph* or *subgraph* granularities.

In selecting our exemplar systems, we wish to ensure coverage of different approaches. While supervised learning approaches have been used for non-detection tasks, they have not seen use for intrusion detection. This is appropriate, as the use of supervised intrusion detection intrinsically makes assumptions about attacker behavior, which is considered to be an unsound practice [32]. Therefore, we focus on the more prevalent class of unsupervised intrusion detection systems. We ultimately choose five such systems to be our exemplars. Due to space constraints, we describe in detail three of the exemplar systems (i.e., StreamSpot, Unicorn, and ProvDetector) below; brief descriptions of two other systems, Pagoda and a SIGL-

[3] We slightly abuse the notation here for clarity, since in our prior definition, $\mathcal{N}_{\beta}^{\gamma}(\cdot)$ takes a graph G_i as a parameter, not an individual vertex v_j^i.

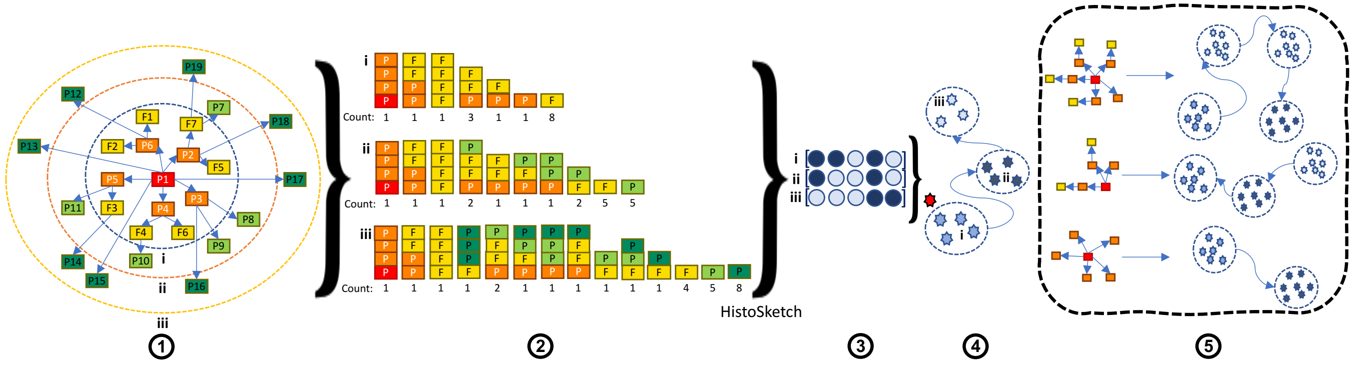


Fig. 3: Overview of the Unicorn Prov-HIDS. ① For a streaming provenance graph, at each time step (marked by a dashed circle), Unicorn ② traverses and chunks graph neighborhoods to construct a histogram representation, ③ encodes each histogram using HistoSketch, and ④ builds a model from all the embeddings. ⑤ The final model is a collection of models from all training graphs. At runtime, Unicorn detects an anomaly in a test graph if it does not fit into any model.

like graph autoencoder, can be found in §VI.

StreamSpot. Fig. 2 shows an overview of StreamSpot [17]. StreamSpot's deconstruction function $\mathcal{N}_\beta^\gamma$ deconstructs G into substructures of (mostly) equal number of vertices. In ① → ②, for each vertex v_j, StreamSpot visits all of its neighboring nodes that are at most K hops away, where $K > 0$ is a configurable parameter. Thus, branching factor $\gamma = deg_{out}(v_j)$ and depth factor $\beta = K$. In ② → ③, StreamSpot subsequently "chunks" substructures into smaller substructures, each of which contains at most J vertices. As such, $\mathcal{N}_\beta^\gamma$ returns a set of substructures Z of equal size J, mixed with substructures of smaller size that cannot be chunked. Fig. 2 illustrates the deconstruction process with $K = 3$ and $J = 4$.[4]

StreamSpot considers the entire chunk set Z when summarizing G (i.e., $\kappa = |Z|$). In ③ → ④, StreamSpot uses StreamHash [17] as the embedding function λ, which preserves the cosine similarity between two graph embeddings. λ embeds each substructure into an L-dimensional *binary* (either -1 or 1) vector. $\mathcal{E}_\lambda^\kappa$ sums the embeddings of all substructures in Z into an integer-value vector I. In ④ → ⑤, it reduces this vector to again a binary vector B as the final graph embedding, assigning each value to 0 or 1 based on the sign in I.

$\mathcal{F}^\delta$ uses the cosine similarity to compare different graph embeddings. In the training phase, StreamSpot performs clustering on each observed graph embedding B. At runtime, it classifies the embedding of each observed (test) graph according to whether or not its embedding can be clustered into the existing model. Specifically, for each cluster, a decision boundary is calculated by finding the cluster's centroid C and standard deviation σ^C. As shown in ⑥, a test graph G^T must satisfy $\mathcal{F}^\delta(C, B^T) \leq 3\sigma^C$ for some cluster (where B^T is G^T's graph embedding), or else be flagged as anomalous.

Unicorn. Fig. 3 shows an overview of Unicorn [18]. Like StreamSpot, in ① → ②, Unicorn deconstructs a *streaming* graph G with $\mathcal{N}_\beta^\gamma$, where $\gamma = deg_{out}(v_j)$, $\beta = K$, and each substructure is chunked into smaller equal sizes of J. However, Unicorn's graph deconstruction also takes *time* into consideration, where a time step t represents a fixed number of edges streamed to G. At each time step, Unicorn revisits vertices whose K-hop neighborhood has changed to update the substructure set Z_t, as well as adding new substructures from latest streamed vertices. Fig. 3 illustrates this process with 3 different time steps using $K = 3$ and $J = 4$.

As shown in ② → ③, Unicorn uses a histogram H_t to represent Z_t at each time step t, such that each bin in the histogram represents a unique substructure and is associated with the frequency of the substructure in Z_t (i.e., $\kappa = |Z_t|$). Unicorn then encodes H_t into an L-dimensional real-value vector R_t, using HistoSketch [33] as the embedding function λ. HistoSketch preserves the Jaccard similarity between two graphs by comparing the *identities* and the *distribution* of histogram bins that summarize those graphs.

Correspondingly, Unicorn's distance function $\mathcal{F}^\delta$ uses the Jaccard similarity to compare between different histogram embeddings. In the training phase, Unicorn builds a model M *for each training graph* by clustering $R = \{R_t\}, t = 1 \ldots T$, as shown in ③ → ④. The model specifies for each cluster a decision boundary (i.e., C and σ^C as in StreamSpot), as well as the temporal relationships between clusters based on the time steps. In ④ → ⑤, Unicorn uses all the models M^D from the set of the training graphs D to detect anomalies at runtime. It generates a histogram H_t^T of an evolving test graph G^T at each time step and compares its embedding R_t^T with every model. Unicorn classifies G^T as anomalous if 1) R_t^T cannot be clustered into any model, *or* 2) the temporal order of R_t^T is different from that in any model.

ProvDetector. Fig. 4 shows an overview of ProvDetector [19]. In ① → ②, ProvDetector deconstructs a graph G into a set of overlapping and non-branching substructures Z. To do so, ProvDetector creates a pseudo source node v_{ps} (which is connected to all vertices in G that have no incoming edges) and a pseudo destination node v_{pd} (which is connected to all vertices in G that have no outgoing edges) and finds *all* paths $P(v_{ps}, v_{pd})$ in G up to a maximum length of 10, at which point longer paths are broken into multiple paths. In other words, in $\mathcal{N}_\beta^\gamma$, $\gamma = 1$ and $\beta = Max(|L(P(v_{ps}, v_{pd})|, 10)$.

In ② → ③, ProvDetector selects $\mathcal{K}$ paths in Z to represent G (i.e., $\kappa = \mathcal{K}$). Paths are selected based on their *regularity scores*, which are computed by the frequencies of the edges

[4] Although it is not documented in [17], StreamSpot down-samples vertex labels from specific system entity names (e.g., `/etc/shadow`, `/bin/bash`) to coarse-grained object types, e.g., `F` (File) and `P` (Process). We make note of this between Step ① and ② in Fig. 2.

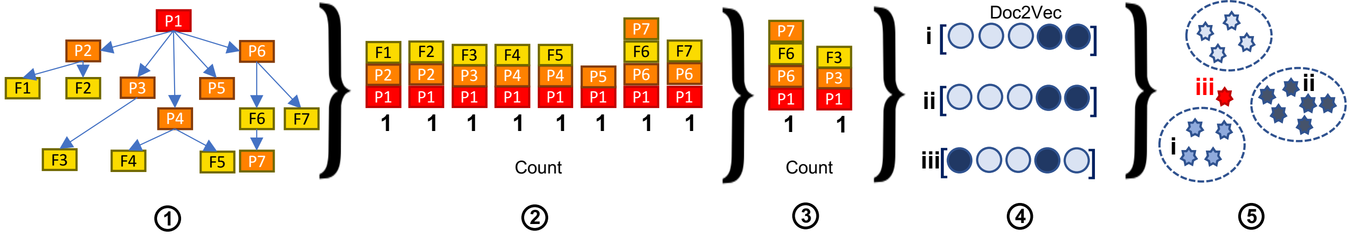


Fig. 4: Overview of the ProvDetector Prov-HIDS. ① Given a provenance graph, ProvDetector ② extracts all paths from the graph, ③ selects and ④ encodes using doc2vec the top $\mathcal{K}$ paths, and ⑤ compares the encoded paths to the trained model.

in the path and the in- and out-degrees of their vertices. We refer interested readers to Hassan et al. [30] for a detailed explanation. ProvDetector selects top $\mathcal{K}$ paths with the lowest regularity scores, then in ③ → ④, encodes each path into an L-dimensional real-value vector using doc2vec [34] as the embedding function λ. Fig. 4 illustrates this process using $\mathcal{K} = 3$.

ProvDetector uses the Euclidean distance in $\mathcal{F}^\delta$ to compare between two path embeddings. In the training phase, ProvDetector builds a Local Outlier Factor model [35] by clustering top-$\mathcal{K}$ path embeddings from all training graphs. At runtime, ProvDetector classifies a test graph as anomalous if at least N out of its $\mathcal{K}$ path embeddings cannot be clustered into the model, as shown in ⑤ (with $N = 1$). Thus, N controls the sensitivity of classification.

IV. Threat Model

Attacker Capabilities. We consider a sophisticated adversary that has gained access to a system. The adversary represents an advanced persistent threat (APT) [36, 37, 38], whose techniques and objectives are consistent with those observed in the MITRE ATT&CK knowledge base [39]:

- *Masquerading (T1036)* [40]. Adversaries commonly manipulate features of their own artifacts to appear legitimate to security tools. Our attacker therefore will use any available procedures to evade detection by Prov-HIDS. Restated formally, given the *unmodified* attack footprint $G^A = (V^A, E^A)$ that would otherwise be detected by Prov-HIDS as anomalous, i.e., $f(G^A) = 1$, the attacker's objective is to apply a transformation function $\mathcal{T}(G^A) = \tilde{G^A} = (\tilde{V^A}, \tilde{E^A})$ such that $f(G^A) \neq f(\tilde{G^A})$.
- *Gather Victim Host Information (T1592)* [41]. Adversaries will use this reconnaissance technique to aid other attack operations such as masquerading. We assume our attacker has access to procedures that enable them to access or infer the contents of audit logs, an especially valuable source of host information. For example, the attacker may escalate privileges to read directly from the logs (`/var/log/audit` in Linux). If privilege escalation procedures are not available, the attacker can instead infer log contents through profiling other processes on the system with `ps`, `netstat`, etc., or even through system side channels (e.g., [42]). For simplicity, we assume the attacker can read directly from system logs.

We place restrictions on the transformation function $\mathcal{T}$ used by the attacker. $\mathcal{T}$ will always take the form of an addition, i.e., $\tilde{E^A} = E^A + \Theta$, where Θ cannot introduce disconnected edges to $\tilde{G^A}$. $\tilde{V^A}$ comprises the union of V^A and the vertices referenced in Θ; that is, Θ can contain nodes not in the original graph such that $V^A \subseteq \tilde{V^A}$. While evading detection, the attacker must still perform the attack and succeed. This means that $\mathcal{T}$ cannot transform G^A such that the semantics of the original attack become invalid. Additionally, the attacker has control over their attack subgraphs only, i.e., $\mathcal{T}$ must be applied exclusively to subgraphs in G^A that describe attack processes. This more conservative attack model reflects the requirement for $\mathcal{T}$ to produce a *realizable* attack strategy given the constraints of the target system; lifting these restrictions would further empower the attacker and facilitate evasion.

Assumptions & Trust Model. We make the following assumptions about the operating environment. Our trusted computing base (TCB) is comprised of the operating system, auditing frameworks, and provenance analysis tools including the Prov-HIDS. We assume the integrity of the components in the TCB at the time of installation and throughout the incursion. Platform and audit log integrity are widely assumed in the auditing literature [43, 44, 45, 46, 47, 48]. Finally, we *do not assume* that the attacker has knowledge of the systems under test. Specifically, we consider whether our attack strategies can evade multiple Prov-HIDS in §VI-I and their robustness to changes in model parameters in §VI-J.

V. Prov-HIDS Evasion Tactics

Following our in-depth analysis of the exemplar systems and bearing in mind the capabilities of our attacker, we now introduce a series of tactics for evading Prov-HIDS. Rather than targeting individual systems one-by-one, each tactic describes a generic strategy that can be recruited by an adversary to evade one or more Prov-HIDS. Because they are inter-operable and can be used in concert, we thus consider each tactic to be a "mimicry gadget" for us in an end-to-end evasion attempt.

A. Preliminary Observations

Before presenting our mimicry gadgets, we make the following observations about the Prov-HIDS detection scenario.

- *Adversaries can exert influence over an attack's embedding.* Prov-HIDS take as input a provenance graph that describes the totality of system execution. Trivially, an attacker that is able to engage in attack behaviors on a system is also able to engage in additional behaviors beyond those strictly required by the attack. These additional behaviors will appear

Alg. 1: The idealized Prov-HIDS evasion strategy. AddSubStructs follows naturally from our observations in §V-A, while the PickSubStructs function is concretized by our mimicry gadgets (see §V).

```
Inputs : Attack Graph G^A, Benign Graph G^N
Output: Evasion Graph G^E
1  G^E = G^A                       // initialize evasion graph
2  Z_N = N_β^γ(G^N)                // deconstruct the benign graph
3  V_N = E_λ^κ(Z_N)                // encode benign substructures
   /* repeat step 2 and 3 for evasion graph          */
4  V_E = E_λ^κ(N_β^γ(G^E))
   /* compute the distance to evasion                */
5  Dist = F^δ(V_N, V_E)
   /* repeat until G^E is misclassified              */
6  while Dist ≥ threshold do
       /* select substructures from Z_N              */
7      P_N = PickSubStructs(G^E, Z_N, V_N)
       /* add substructures to G^E                   */
8      G^E = AddSubStructs(G^E, P_N)
       /* re-encode G^E                              */
9      V_E = E_λ^κ(N_β^γ(G^E))
       /* re-compute the distance to evasion         */
10     Dist = F^δ(V_N, V_E)
11 end
12 return G^E
```

in the embedding of the attack subgraph, injecting confusion into the Prov-HIDS' classification task. This observation is analogous to Wagner and Soto's observation that inserting "no-op" system calls into a malicious process could evade the Forrest IDS without affecting attack semantics [14].

- *Adversarial additions to an attack's embedding can be made indistinguishable from benign behavior.* While it is unsurprising that an attacker can add arbitrary behaviors, what is surprising is that these additions can be made indistinguishable from the system's legitimate behavior. In a provenance graph, any behavior that an attacker engages can be causally linked back to their point of entry into the system, seemingly indicating that any extraneous behavior will still appear suspicious in the eyes of the Prov-HIDS. Unfortunately, in the process of vectorizing the provenance graph into a fixed-length representation, Prov-HIDS' deconstruction of the graph disassociates graph neighborhoods from one another through bounded branching (γ) and depth (β). For example, the Unicorn authors consider a maximum depth of three [18]; while the ProvDetector authors construct paths of maximum depth of 10 [19]. To the best of our knowledge, the same issue arises in all Prov-HIDS in the literature. Thus, even if an injected behavior has a malicious *ancestry*, the behavior's embedded representation will map to a benign behavior if it is more than β hops away from the root of the attack graph.

B. Mimicry Gadgets

Based on these observations, the basic premise of an Prov-HIDS evasion attempt emerges. An idealized version of this strategy is given in Alg. 1. Starting with the attack graph (G^A), the attacker engages in behaviors found in G^N that add seemingly benign graph substructures into the evasive attack graph G^E. This process continues until the classification of G^E crosses the Prov-HIDS' decision boundary. We next describe the construction of the key function, PickSubStructs.

1) Abusing Unweighted Graph Encoding: When a Prov-HIDS summarizes a graph in such a way that every substructure is equally weighted, an attacker could change the graph's embedding simply by adding additional activity. Consider an L-dimensional embedding that is derived from summing over the set of substructures output by $\mathcal{N}_\beta^\gamma(G_i)$. An attacker that engages in spurious system activities creates additional substructures into G_i, changing the output of $\mathcal{N}_\beta^\gamma(G_i)$ and consequently the L-dimensional embedding. In fact, because the L-dimensional embedding is a fixed length feature vector that describes a graph whose size is under the attacker's control, we predict that for unweighted Prov-HIDS, there will always exist a set of substructures that can transform the attack graph into any embedding within the L-dimensional space.

In our first mimicry gadget, the attacker starts by profiling the target system to identify a large number of graph substructures associated with benign activity (see §IV). They then select a batch of benign substructures of parameterizable size and replicate the system activities that produce those structures. In our implementation, substructures are selected chronologically in the order that they appeared in the logs. *We predict that, given a batch of benign substructures of sufficient size,*[5] *this evasion gadget will consistently force misclassification against unweighted Prov-HIDS.*

Consider one of our exemplar Prov-HIDS, StreamSpot, which uses a bag-of-words approach to embed the input graph. The hashing function sums over every hash of all the substructures within the graph to create the final embedding. Under this unweighted graph encoding approach, StreamSpot can differentiate typical from anomalous behavior if and only if there is a significant difference in the set of the substructures that existed from each behavior. However, by adding benign substructures to the attack graph, the significance of the anomalous substructures can be arbitrarily reduced, to the point that the attack graph would fall inside of the decision boundary of a benign cluster. Further, as StreamSpot does not represent any temporal properties within its $\mathcal{N}_\beta^\gamma(G_i)$ function, the order in which substructures are added does not affect the embedding, further increasing the likelihood of the mimicry gadget's success. While we will evaluate this mimicry gadget against StreamSpot and Pagoda, we note that other Prov-HIDS in the literature, including FRAPpuccino [25] and PIDAS [27], utilize similar graph encoding mechanisms.

2) Abusing Distributional Graph Encoding: Graph embedding techniques that focus on summarizing the substructure distribution preserve the global graph structure at the expense of highlighting anomalous activity. Unlike the previous summarization approaches that weigh all substructures equally, techniques preserving the underlying distribution weigh each unique substructure according to its prevalence

[5] We note that this approach is heuristic, but in theory the attacker could solve a system of linear equations to determine the minimally sufficient substructure set required to force misclassification. We opt for the heuristic approach, because it is computationally less complex and does not assume that the attacker has perfect knowledge of the Prov-HIDS.

within $\mathcal{N}_\beta^\gamma(G_i)$. Prov-HIDS that utilize these encoding techniques assume that the distribution of substructures in a malicious graph differs significantly from that in a benign graph. However, the distribution of the substructures in a malicious graph is (at least in part) under the attacker's control. Further, to account for the fact that a provenance graph grows continuously over time, these Prov-HIDS must normalize substructure distributions to compare past (i.e., training data) and present embeddings (i.e., test data). Thus, we predict that there will always exist a set of substructures that can transform an attack graph into any embedding within the L-dimensional space.

In our second mimicry gadget, the attacker profiles the target system to identify the relative frequency of each observed substructure associated with benign activity. They then select a batch of benign substructures that preserves this distribution and replicate the system activities that produce those substructures. The batch size is again parameterizable; in this gadget, this parameter must be set such that the normalized representation of any malicious substructures accounts for a vanishingly small proportion of the embedding. This gadget implementation also considers Prov-HIDS that monitor temporal features of the graph by selecting substructures that are consistent with the distribution of the activities at the time of the initial compromise and updating its representation of the substructure distributions as the graph evolves. *We predict that, given the knowledge of a benign substructure distribution and a transformation of sufficient size, this gadget will consistently force misclassification against distributional Prov-HIDS.*

We return to our exemplar Prov-HIDS, Unicorn, which represents the underlying substructure distribution in a histogram that captures the normalized frequency of each unique substructure within $\mathcal{N}_\beta^\gamma(G_i)$. This histogram is embedded to create an L-dimensional vector such that embeddings that are close together share similar substructure frequencies. However, by adding benign substructures in a manner that mirrors the frequencies of an existing cluster, the attack graph can be moved closer to a benign cluster to the point that it falls within the decision boundary. This is because the malicious substructures in the attack graph's embedding, while easily identifiable by Unicorn in the original graph, will have a diminishing contribution to the classifier's decision as a result of normalization. While we will evaluate this mimicry gadget against Unicorn, Pagoda, and SIGL, we note that other Prov-HIDS in the literature, such as P-Gaussian [26], utilize similar graph summarization mechanisms.

3) Abusing Downsampled Graph Encoding: Thus far, our mimicry gadgets have called attention only to the risks of encoding the entire provenance graph when a portion of it is under the attacker's control. An alternate approach is to perform a *downsampling* procedure prior to graph embedding, which may improve the efficiency of training and generalizability of the model. It also stands to reason that a downsampled graph embedding may be more difficult to insert mimicry substructures because they must first bypass the downsampling function. Consider a Prov-HIDS that first passes the target graph into a downsampling function, i.e., $DS(G_i) = G'_i$, before decomposing its substructures ($\mathcal{N}_\beta^\gamma(G'_i)$). We observe an issue with such an approach; namely, that by making decisions about which subgraphs are relevant to the classification decision, $DS(\cdot)$ effectively replicates the functionality of the Prov-HIDS' core decision function, $\mathcal{F}^\delta(\cdot)$. There is a disconnect between these two notions of anomaly – after all, one is defined over a graph while the other is defined within the embedding space. We predict that there must exist benign substructures that can bypass $DS(\cdot)$ while still affecting the attack's representation within the L-dimension space.

In our final mimicry gadget, the attacker first profiles the target system while monitoring the behavior of $DS(\cdot)$ on different observed sequences. They then select a batch of benign substructure sequences of parameterizable size that bypass the downsampling operation. As before, the batch size parameter is based on what is necessary to affect the final representation of the attack graph in the embedding space. *We predict that, given the knowledge of the downsampling function and a sufficient transformation size, this gadget will consistently force misclassification against downsampling Prov-HIDS.*

Our exemplar Prov-HIDS, ProvDetector, attempts to classify the full test graph by downsampling it to only the top $\mathcal{K}$ paths with the lowest regularity scores. Regularity scores are calculated by observing the frequencies of individual events in the system and then using a diffusion algorithm to aggregate these scores along a path. Subsequently, in the embedding space, ProvDetector uses the Euclidean distance between path vectors to build a clustering model. Unfortunately, because ProvDetector's clustering model only trains on the paths that have the lowest regularity scores, its notion of normality in the embedding space is distorted. In particular, if there exists at least one low-regularity path that occurs frequently enough in training to form a benign cluster, the attacker can inject copies of this path $\mathcal{K}$ (or fewer) times such that their actual attack behaviors are entirely removed from the embedding.

VI. Evaluation

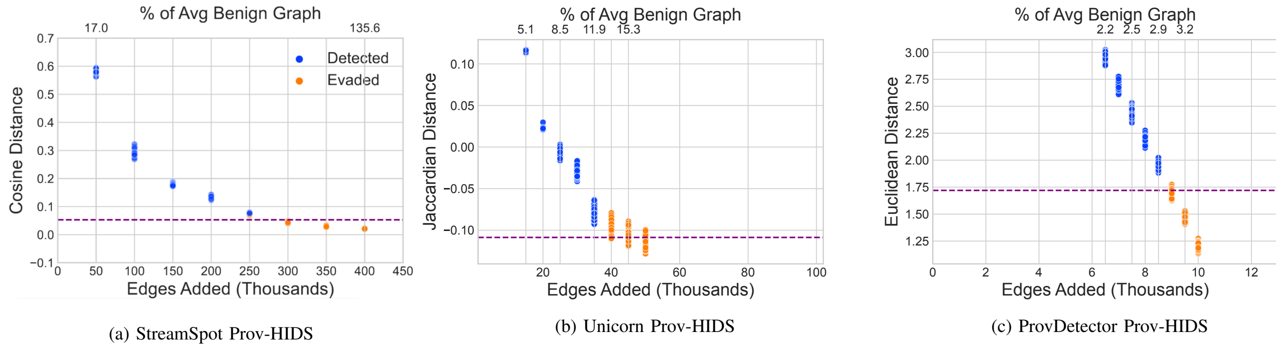
We performed a number of experiments to evaluate the efficacy and efficiency of our evasion strategies. Our evaluation focuses on addressing the following research questions:

- How effective are our mimicry gadgets against the five state-of-the-art, exemplar Prov-HIDS, StreamSpot (§VI-D), Unicorn (§VI-E), ProvDetector (§VI-F), Pagoda (§VI-G), and a SIGL-like full graph autoencoder (§VI-H)?
- Are our mimicry gadgets interoperable and generally applicable to other detection mechanisms (§VI-I)? Relatedly, how much knowledge does the attacker need when using our gadgets to successfully evade detection (§VI-J)?
- Can the attacker practically deploy our mimicry gadgets in the real world? (§VI-K)
- How does our evasion strategies compare to previous, domain-general graph evasion techniques (§VI-L)?
- What is the runtime performance of our approach (§VI-M)?

A. Experimental Setup

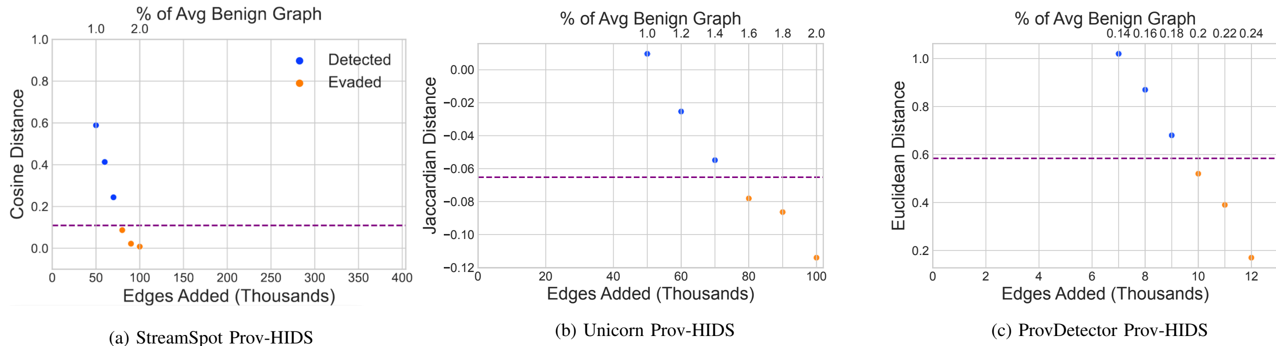
All experiments were run on a modestly provisioned 20-core Intel Xeon(R) server with 64GB of RAM and an NVIDIA GeForce GTX 1080 Ti. Our exemplar systems were implemented and configured as follows:

- *StreamSpot*. We use its open-source implementation [22] with recommended parameters: depth factor $K = 1$, chunk size $J = 50$, and the embedding dimensionality $L = 1,000$.



(a) StreamSpot Prov-HIDS (b) Unicorn Prov-HIDS (c) ProvDetector Prov-HIDS

Fig. 5: Evasion results against the StreamSpot dataset. For each value on the x-axis, there are 100 points representing the 100 unique attack samples included in the dataset. The purple line represents the average decision threshold across the benign graphs. The y-axis varies by Prov-HIDS: StreamSpot uses the cosine distance from the nearest benign cluster, Unicorn measures the normalized Jaccardian distance, and ProvDetector uses cluster density and the Euclidean distance.



(a) StreamSpot Prov-HIDS (b) Unicorn Prov-HIDS (c) ProvDetector Prov-HIDS

Fig. 6: Evasion results against the DARPA Transparent Computing dataset. See Fig. 5's caption for an explanation of the different plot components. Because this dataset describes only one intrusion, there is only one point per value on the x-axis.

- *Unicorn*. We use its open-source implementation [49] with the best parameters in [18]: $K = 3$, $J = 50$, and $L = 2,000$. At each time step, we stream 500 edges to the graph.
- *ProvDetector*. While ProvDetector is closed-source, we reimplemented the system based on its description in the original paper [19] and through extended dialogues with the authors. The authors recommended the following configuration: depth factor $\beta = 10$, the dimensionality of path embedding $L = 100$, the number of paths to represent a graph $\kappa = 20$, and the sensitivity of classification $N = 3$.

Configurations of the other two exemplar systems, Pagoda and a SIGL-like full graph autoencoder, are provided in §VI-G and §VI-H, respectively, along with their descriptions.

B. Datasets

We make use of two datasets, the StreamSpot dataset [22] and DARPA Transparent Computing (TC) Engagement 3. The StreamSpot dataset, generated from SystemTap [50], was used in the original evaluations of StreamSpot and Unicorn. The dataset was generated in 2016 and contains one drive-by-download attack scenario and five benign host activity scenarios (watching YouTube, browsing CNN, downloading files, checking emails, and playing a game). The attack describes a Firefox vulnerability in which a victim clicking on a malicious URL accidentally triggers a bug in Flash that enables the attacker to obtain root access. Each behavior was recorded 100 times, resulting in 600 provenance graphs. In our experiments, we exclusively trained on the browsing CNN scenario, which advantaged the classifier by allowing it to define tighter bounds around normality. When we evaluate the complexity of our graph transformations, we note that each benign graph in the dataset averages about $295K$ edges, while the attack graphs average about $28K$ edges. StreamSpot does not contain behaviors representative of real-world systems as all of the benign graphs contain system events from a single activity. More importantly, attack graphs contain only attack behavior.

DARPA's Transparent Computing program released multiple attack engagement datasets that describe a professional red team's attempts to penetrate a small network of hosts. We evaluate our evasion tactics on the Engagement 3 data that was generated by the THEIA team, which includes a single provenance graph broken up into 25 different time periods. While the dataset includes other smaller unsuccessful intrusion attempts, we make use of the successful and more complex Drakon intrusion that was described in §II. When evaluating the complexity of our graph transformations, we note that the total activity in this dataset is approximately $4.8M$ edges. THEIA represents more realistic system activity and features attack behavior occurring alongside normal system activity. Table II provides a summary for each dataset.

Dataset	# of Nodes	# of Edges	# of Graphs	# of Attacks
SS	822,998	27,792,491	600	1
THEIA	3,721,210	46,303,154	25	4

TABLE II: Dataset statistics for StreamSpot (SS) and DARPA TC3 THEIA. THEIA has one provenance graph split into 25 provenance subgraphs representing different time periods. There are four different attacks conducted in THEIA, while StreamSpot had only one attack executed for 100 times.

C. Gadget Implementation and Evasion Procedure

Each of our mimicry gadgets was implemented in Python. Each gadget takes as input (1) the attack graph to be modified, (2) a provenance graph of benign activity, (3) the number of edges/substructures to add, and (4) a point of insertion. We follow the steps below to create an evasion graph:

1) *Load the pre-attack graph* from our experimental datasets. The *pre-attack* graph describes the state of the victim's system immediately before the attacker infiltrates the system. We create the pre-attack graph from the original attack graph by identifying the first system process (chronologically) in the attack footprint according to the ground truth and then removing all system activity (including both malicious and benign background activity) after this process.
2) *Load benign graphs* used to train the Prov-HIDS.
3) *Find the insertion point* corresponding to the set of edges within the attack graph that describes the attacker connecting to and gaining control over a process on the victim's machine. For example, in Fig. 1, the point of insertion is the edges associated with the green/red Firefox process.
4) *Inject benign substructures* into the attack graph (§V). For StreamSpot, we deconstruct a benign graph from the training dataset into its K-hop substructures and then add all substructures to the attack graph multiple times. For Unicorn, we calculate the normalized count for each K-hop substructure within a benign graph and then add to the attack graph the set of the K-hop substructures by their normalized counts multiple times. For ProvDetector, we generate a frequency database (where ProvDetector stores the frequencies of different edges, see Fig. III-C) from a benign graph. We then extract all paths from the graph and identify the most abnormal path based on the frequency database. We sample another benign graph if the most abnormal path has a regularity score less than the attack paths.[6] We introduce the path N times into the attack graph. We describe the procedures to inject benign substructures for Pagoda and the graph autoencoder in §VI-G and §VI-H, respectively. Note that in our experiments, we evaluate the number of edges/substructures to be added to successfully evade detection successfully. Further, for each of our attacks, to ensure that the additional innocuous behavior does not get associated with any malicious edges, we (as the evasive attacker) fork the initial compromised process up to a depth factor β to disassociate the mimicry behavior.
5) *Insert the attack payload's substructures* to the pre-attack graph. The attack substructures contain all edges that an attacker-controlled process can reach *after* the attacker had infiltrated the system. Finally, the resulting evasion graph is evaluated against the Prov-HIDS.

[6]We assume the ability to estimate each path's regularity score by running the attack locally and using the frequency database to calculate the score.

D. Evading StreamSpot

We make use of our first mimicry gadget, *abusing unweighted graph encoding* (§V-B1), to evade the StreamSpot Prov-HIDS. Using this gadget, we iteratively add batches of benign substructures to the attack graph until misclassification occurs. We continue to add more substructures to ensure that each attack *remains* undetected after the initial false negative.

Our results are shown in Fig. 5a and Fig. 6a for the StreamSpot and DARPA datasets, respectively. The primary x-axis indicates the number of *edges* added, instead of substructures, to facilitate comparison across different Prov-HIDS. The secondary x-axis shows the edges added as as a *percentage* of the (average) size of a benign graph in the dataset. The y-axis shows the cosine distance between an attack graph and the centroid C of the nearest cluster in the model. The purple line indicates the average decision boundary among all clusters in the model, providing a rough estimate of where misclassification occurs. Against the StreamSpot dataset, some of the attack graphs begin to evade detection after adding $250K$ edges,[7] and all of the attacks become undetectable after $300K$ edges. In the DARPA dataset, the attack evades detection after $80K$ edges. For both datasets, our evasion strategy had 100% success.

Remarks. As we can see, distance to a benign cluster reliably decreases as more benign substructures (edges) are added to the attack graph. At first glance, it may seem unreasonable to add hundreds of thousands of edges to the attack graph; surely, such an approach is far from convert. However, in reality, this transformation simply scales the attack graph to the same order of complexity as the benign graph samples. In the StreamSpot dataset, the benign graph averaged $295K$ edges while the attack graphs averaged $28K$ edges. The necessary transformation added at least $250K$ edges, 85% of the benign graph size. This rules out the possibility of using a trivial event counting mechanism to detect this evasion attempt. The high cost of evasion on the StreamSpot dataset ($250K$, or 85%) as compared to DARPA ($80K$, or 1.5%) may be the result of lack of diversity of benign activity in the former; while StreamSpot's training data describes a single browsing behavior, DARPA describes system-wide activities. We further attribute the high evasion cost for the StreamSpot dataset to the relative naïveté of the first mimicry gadget. In §VI-I, we demonstrate that our second mimicry gadget can evade the StreamSpot IDS at a much lower cost ($40K$ edges).

E. Evading Unicorn

Against Unicorn, we make use of our second mimicry gadget, *abusing distributional graph encoding* (§V-B2). Recall that the Unicorn Prov-HIDS encodes the temporal properties of a provenance graph, re-embedding the graph to train its model after every t new edges. To account for this, for each batch of t edges in the graph that contains an attack edge, we parameterize the gadget to select t edges whose substructures match the distribution of the unmodified training graph.

Our results are shown in Fig. 5b and Fig. 6b for the StreamSpot and DARPA datasets, respectively. The y-axis

[7]The careful reader might also notice that a few successful evasions in Fig. 5a are above the purple line. This is because some cluster boundaries were more permissive than the *average* boundary.

indicates the Jaccardian distance, Unicorn's distance function. Note also the change in scale of the x-axis. Against the StreamSpot dataset, we successfully evade Unicorn in as few as $40K$ edges, or 13.6% of the average size of a benign StreamSpot graph. Against the DARPA data, we again achieve evasion in $80K$ edges, 1.6% of the size of the benign graph. Our evasion strategy evaded Unicorn 100% of the time.

Remarks. When comparing the performance of our first two exemplar systems on the StreamSpot dataset, the more sophisticated Unicorn seems to perform more poorly – $350K$ edges required to evade the StreamSpot Prov-HIDS, as compared to just $40K$ edges to evade the Unicorn Prov-HIDS. Initially, we attributed this to the fact that the StreamSpot authors had the opportunity to finely tune their chosen parameters against this dataset. However, after testing the distributional gadget against the StreamSpot Prov-HIDS (see §VI-I), we discover that it is also able to evade the StreamSpot IDS at the lower cost of $40K$ edges. This indicates that selecting substructures distributionally, as opposed to chronologically, is simply a better approach to attacking StreamSpot's graph encoding technique. In contrast, against the DARPA dataset we evade the two Prov-HIDS with the same transformation of $80K$ edges. The lower overall attack cost might reflect the increased complexity of system-wide activities in the DARPA data, leading to looser bounds around normal behavior clusters. Conversely, the similar performance of the two gadgets indicates that the chronological ordering of events in the dataset happens to better reflect the distribution of events.

F. Evading ProvDetector

Against ProvDetector, we make use of our third mimicry gadget, *abusing downsampled graph encoding* (§V-B3). Recall that this mimicry gadget exploits the disconnect between the metrics used for graph summarization and the metrics used for classification; in both datasets, we indeed identified a benign path that exhibited lower regularity than all attack paths. As a result, our implementation repeatedly injects the substructures associated with this low-regularity benign path. ProvDetector classifies a graph as benign so long as more than $\mathcal{K} - N$ of its substructures fall within a known cluster, where recommended parameterization of $\mathcal{K}$ is 20 and N is 3. In our experiment, we opt to insert 14 to 20 instances of the low-regularity paths.

The results are shown in Fig. 5c and Fig. 6c. Against the StreamSpot dataset, inserting $9K$ edges was sufficient to achieve a 100% success rate. These $9K$ edges translate to 18 of the low-regularity benign path, or 3% of the average size of the benign graph. This is expected, as the sensitivity parameter used for ProvDetector (N) is set to 3, meaning that 18 substructures are needed to fall within a cluster. Against the DARPA dataset, misclassification first occurs at $10K$ edges (0.2% of the benign graph), a difference of $1K$ edges between datasets. The reason for this is that the low-regularity benign path could be immediately inserted in the StreamSpot attack graph (it is just another Firefox activity), while the low-regularity benign path in the DARPA data is not related to the attack graph. As a result, our gadget had to first perform a preliminary transformation of roughly $1K$ edges to create a graft point for the low-regularity benign path that did not itself create an additional suspicious low-regularity path. The evasion strategy achieved 100% success against ProvDetector.

Remarks. Aside from the extra $1K$ transformation in the DARPA dataset, the cost of attacking ProvDetector was identical between datasets. This is intuitive, because ProvDetector embeds a fixed number of fixed-length paths, placing a low ceiling on the cost of attacking the system as compared to StreamSpot and Unicorn. In fact, even against larger provenance graphs describing months or years of activity, we predict that the cost of evading ProvDetector will remain constant due to this down-sampling. Conversely, StreamSpot and Unicorn place less weight on any single substructure within their graphs, requiring a larger transformation to achieve evasion.

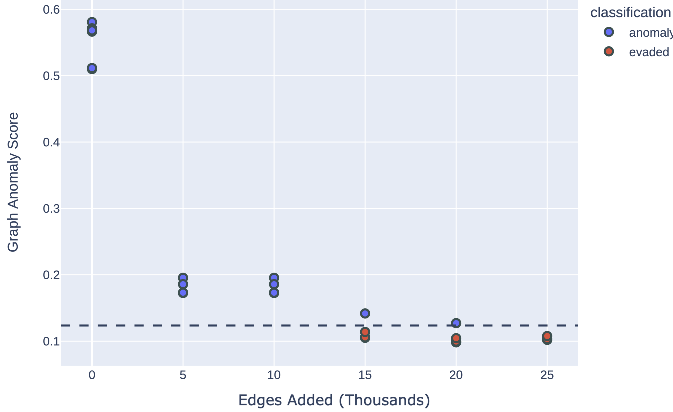
G. Evading Pagoda

Pagoda [20] is a path-based, whole-graph Prov-HIDS. Similar to ProvDetector, it makes use of an event frequency database to assign a *rarity score* to individual edges. If an edge in the path is not part of the frequency database, it is assigned a score of one; otherwise, the edge receives a score of zero.

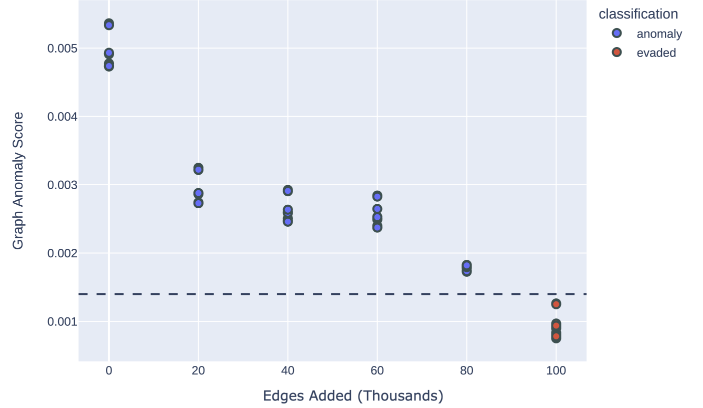
Pagoda flags anomalous graphs in two ways, both based on *path anomaly scores*. A path's anomaly score is the average of all its edges' rarity scores. At the path level, if *any* path in the graph has an anomaly score greater than a configurable path-level threshold, the graph is considered to be anomalous. At the graph level, Pagoda additionally assigns a *graph anomaly score* to the entire graph, based on a weighted summation of all paths' anomaly scores. A path's weight is its normalized path length based on the lengths of all paths in the graph. Pagoda considers a graph to be abnormal, if its *graph anomaly score* is greater than a predetermined graph-level threshold.

The original Pagoda source code is publicly-available; however, after sustained correspondence with the authors, we ultimately decided to re-implement the system. Pagoda is built on top of the 2006 Provenance-Aware Storage System (PASS) [51], the first provenance-aware operating system, with deep integration into PASS' userspace utilities that were last updated in 2010. After several attempts, we were unable to recreate a build environment in which these tools could run. Instead, we referenced the author's original code and pseudocode from the paper [20] to reproduce the system. We configured the system as follows: If an event in the training dataset appears more than twice, we add the event to the frequency database. The path-level threshold is set to be the highest anomaly score among all the paths in the training dataset. Similarly, the graph-level threshold is the highest anomaly score of all the graphs in the training dataset. Within a graph, a path starts from a root node and ends at a leaf node.

To evade Pagoda, we utilized the *abusing unweighted graph encoding* (§V-B1) and *abusing distributional graph encoding* ((§V-B2) mimicry gadgets. Because Pagoda normalizes each path's length to compare graphs of different sizes, an attacker could insert long but benign paths to reduce the weights of shorter attack paths, thus lowering the graph anomaly score of the attack graph. In our experiment, we used the training dataset to identify long benign paths with low anomalous scores to lower the graph anomaly score. For each attack path, the attacker could insert edges from the frequency database to decrease the path anomaly score of the attack path, preventing it from raising detection. Therefore, in our



(a) Pagoda



(b) FGA

Fig. 7: Evasion results against the StreamSpot dataset. For each value on the x-axis, there are 100 points representing the 100 unique evasion graphs based on the attack samples included in the dataset. The y-axis varies by Prov-HIDS: Pagoda uses a graph anomaly score represented as a weighted sum of all the paths within the graph, while FGA uses the Euclidian distance from the nearest benign embedding.

experiment, we inserted edges in the frequency database into attack paths to lower their anomaly scores. The results are shown in Fig. 7a. Our re-implementation can correctly classify StreamSpot attack graphs against a non-evasive attacker but is susceptible to our evasion strategies 100% of the time.

H. Evading a Full Graph Autoencoder (FGA)

SIGL [21] is a Prov-HIDS that utilizes a graph autoencoder to alert security analysts to malicious software installations. SIGL learns by training a graph autoencoder using reconstruction losses on SIGs representing clean software installs. While SIGL is a Prov-HIDS, its focus on software installation means that it only considers small subgraphs of targeted process behaviors. It stands to reason that performing anomaly detection over these smaller, well-formed procedures is a significantly different graph learning task than highly variable system-wide anomaly detection. Evidence for this claim can be found in [21] where larger SIGs (i.e., more variations) incurred the highest reconstruction losses. To more fairly compare other Prov-HIDS explored in this paper, we implement a full graph autoencoder (FGA) to detect intrusion, for mimicry analysis.

Graph autoencoders are machine learning models that consist of two separate neural networks, an encoder and a decoder. For an input graph, the encoder produces a D-dimensional vector; the decoder then takes as input the D-dimensional vector and tries to reconstruct the original input graph. By minimizing the difference, also known as the *reconstruction loss*, between the input and reconstructed graph, the autoencoder can learn good representations of graphs within the training distribution. As SIGL is a proprietary closed-source system, we implemented an FGA using PyTorch Geometric [52] to learn normal behavior from benign provenance graphs through reconstruction loss measurements. Our FGA uses a graph convolutional network (GCN) [53] to embed each node and an inner product decoder [54] to reconstruct the node's neighborhood. We represent a graph's embedding as the average of the node embeddings from all the nodes within the graph. At test time, a graph is labeled as malicious if the distance from its embedding to the nearest training graph is greater than some predefined threshold. We configured the threshold in FGA using a validation set, which is set to be the tightest bound we can find to correctly classify all validation graphs as benign. We see from Fig. 7b that FGA can perfectly classify attack graphs in the StreamSpot dataset.

Evasion Strategy	E.A.	StreamSpot	Unicorn	ProvDetector
Gadget 1 (§V-B1)	$80K$	✔	✖	✖
Gadget 2 (§V-B2)	$80K$	✔	✔	✖
Gadget 3 (§V-B3)	$10K$	✖	✖	✔
Gadget 2 $\rightarrow$ Gadget 3	—	✖	✖	✔
Gadget 3 $\rightarrow$ Gadget 2	—	✔	✔	✔

TABLE III: Cross-comparison of evasion gadgets. ✔'s mark successful evasion, while ✖'s denote failure. E.A. refers to *Edges Added*.

To evade FGA, we utilized the second mimicry gadget, *abusing distributional graph encoding* (§V-B2), to match the distribution of node neighborhoods of an attack graph to be the same as that in benign provenance graphs. Because FGA uses a GCN to embed the input graph, each node is encoded in terms of its "ancestral" k-hop neighborhood. GCN uses message passing to aggregate feature vectors from a node's parents to generate the D-dimensional vector for the node. Adding the same "ancestral" k-hop neighborhood found in the training dataset to a given attack graph introduces nodes with the same embedding as nodes associated to benign activity. This allows the average embedding of all the nodes in the attack graph to be closer to that of a benign graph, thus successfully evading the Prov-HIDS, as we can see in Fig. 7b.

I. Gadget Composability Analysis

As discussed in §V, because our mimicry gadgets target general design features of Prov-HIDS, they are interoperable with one another and can be deployed in concert to evade more complex detection systems. As a proxy for such a system, we now consider whether our evasion techniques can succeed when three of our exemplar Prov-HIDS (StreamSpot, Unicorn,

and ProvDetector) are deployed as an ensemble model. To do so, we make use of the DARPA dataset. The Prov-HIDS are all configured identically to the prior experiments. For each test, we use the evasion strategy to add the number of edges that was first observed to cause misclassification of the target system. Based on these findings, we then test whether a composition of evasion strategies can be used to evade all three systems.

Our results are shown in Table III. As expected, gadgets designed for one graph encoding strategy were not effective when tested against another. The one exception to this rule is the *abusing distributional graph encoding* gadget (Gadget 2), which also succeeded against Prov-HIDS that use unweighted graph encoding. We then attempted to use different compositions of gadgets to see if it was possible to simultaneously evade all Prov-HIDS. Because the Gadget 2 was already demonstrated to be sufficient to evade the StreamSpot Prov-HIDS, Gadget 1 was removed from consideration. We first applied Gadget 2 *and then* Gadget 3, but this was detected by StreamSpot and Unicorn, because the $\mathcal{K}$ paths added by Gadget 3 for ProvDetector created unexpected substructures in the L-dimensional embeddings. We then applied the transformations in the opposite order, first injecting $\mathcal{K}$ paths using Gadget 3 and then concealing any anomalous changes to the substructure distribution using Gadget 2. This composition of evasion strategies was sufficient to evade all three Prov-HIDS.

Remarks. This analysis highlights two important considerations regarding the deployability of our evasion attacks. First, the results indicate that we might be able to relax our assumptions of the attacker's knowledge of Prov-HIDS on the target system – if the attacker can create a generally applicable evasion transformation, it might be possible to simultaneously evade many detection mechanisms and many parameterizations of those mechanisms (which we evaluate in §VI-J). Second, these results underscore the value of our approach as compared to domain-general methods of adversarial sample generation. Techniques for misleading machine learning classifiers abound (and are discussed at greater length in §VI-L); however, the existence of an adversarial sample does not illuminate a general strategy for evasion, provide interoperability with attacks against related systems, or highlight the design flaws that led to its feasibility. Our approach satisfies all these requirements.

J. Evasion Performance Under Incorrect Parameters

Setting the parameters in an intrusion detection system can often be unique to the deployment, to reflect the nature of system activity within that environment and the tolerance of the security analyst for false positives. To assess the practicality of our evasion strategy, we consider an attacker with imperfect knowledge of the intrusion detection system. To do so, we selected a key parameter for each of our three exemplar systems: StreamSpot's chunk size, Unicorn's sketch size, and ProvDetector's path length. We selected these parameters, because they play a roughly analogous role in each system. Using the StreamSpot dataset for training and testing, we then slowly increased and decreased each parameter's value until classification accuracy against the unperturbed test split began to degrade. We did not test parameters that would cause more severe degradation (e.g., 100% false positive rate) as such classifiers would be impractical to use. Finally, we re-tested the original evasion samples on each parameterization.

System	Parameter	Value	TPR	TNR	Evasion Rate
StreamSpot	Chunk Size	40	0.86	1.00	1.00
StreamSpot	Chunk Size	45	0.91	1.00	1.00
StreamSpot	Chunk Size	50	1.00	1.00	1.00
StreamSpot	Chunk Size	55	1.00	0.95	1.00
StreamSpot	Chunk Size	60	1.00	0.90	1.00
Unicorn	HistoSketch Size	500	0.99	0.94	1.00
Unicorn	HistoSketch Size	1000	1.00	0.92	1.00
Unicorn	HistoSketch Size	2000	1.00	0.93	1.00
Unicorn	HistoSketch Size	4000	0.95	0.96	1.00
Unicorn	HistoSketch Size	8000	0.29	0.96	1.00
ProvDetector	Path Length	4	0.0	1.00	1.00
ProvDetector	Path Length	6	0.21	1.00	1.00
ProvDetector	Path Length	8	1.00	1.00	1.00
ProvDetector	Path Length	10	1.00	1.00	1.00
ProvDetector	Path Length	20	1.00	0.92	0.86

TABLE IV: Success of evasion attack when a key parameter is manipulated. TPR and TNR denote the true positive and true negative rates of the model parameterization prior to evasion. The reference parameter value used in our prior experiments, and here (incorrectly) assumed by the attacker to be fixed, is shaded. Our evasion methodology continues to be effective even as the baseline performance of the model degrades.

Evasion Strategy	StreamSpot	Unicorn	ProvDetector
Full Knowledge	✔	✔	✔
Limited Knowledge	✔	✔	○

TABLE V: Live demonstration results for limited knowledge and full knowledge attack scenarios: ✔'s mark successful evasion, and ○'s represent partial success (see details in §VI-K).

Table IV reports the results of those experiments, with the shaded parameter value denoting the original experimental setup that the attacker assumes (incorrectly) to be fixed. As we can see from the True Positive Rate (TPR) and True Negative Rate (TNR) columns, each parameter value was manipulated until it reached a clearly suboptimal value for the unperturbed test data. In spite of this, the adversarial samples we generated for a different parameterization continue to be effective. Our attack strategy generates each evasion graph by adding a set of benign substructures rather than individual edges to a given graph, meaning that individual perturbation steps are not strongly linked to parameter values. As a result, there is slack in each evasion attempt that allows for it to remain successful despite Prov-HIDS' parameter changes.

K. Live Attack Demonstration

While we have demonstrated the ability to produce mimicry attacks in the feature space, previous work [55] has noted that side effects can occur when one attempts to invert samples from the feature space to the problem space. In this section, we demonstrate that these strategies can be operationalized on a live system by an attacker. We first replicate an attack scenario similar to the StreamSpot dataset in which benign activity was generated by opening the Firefox browser and visiting `youtube.com` and `cnn.com`. This activity was used to train the classifier.

In the attack scenario, the victim clicks a malicious link that exploits a vulnerability in Firefox, causing a reverse shell to open and creating an attacker-controlled process. Before initiating the remainder of the attack, the attacker attempts to evade the Prov-HIDS by injecting innocuous behavior patterns.

These innocuous behaviors are elicited by the attacker using a simple Python script to visit legitimate websites. After eliciting these behaviors, the attacker then injects its payload behavior by using a JavaScript vulnerability to gain root privilege and extract sensitive information to a remote server. This attack sequence was used as test data. We simulated the victim's and the attacker's systems using an Ubuntu and Kali VM on Virtualbox, respectively. Metasploit was used to conduct the intrusion and privilege escalation.

Prior to the beginning of the attack, the attacker will use their knowledge of the victim system to both select innocuous behaviors to inject as well as create a surrogate model of the target system's Prov-HIDS for preliminary testing. We consider two viable methods of injecting innocuous behaviors: a *Full Knowledge (FK)* attack and a *Limited Knowledge (LK)* attack. In the FK scenario, the attacker can read directly from the system logs in the training data, allowing them to identify the websites visited by Firefox. In the LK scenario, the attacker can infer that Firefox is running but cannot directly access the logs to determine which websites were visited. Instead, the LK attacker browses popular websites on their own machine and inspects the log output to determine plausible low-level system call patterns. The attacker then visits these other websites (but *not* `youtube.com` and `cnn.com`) during the attack. These scenarios reflect different levels of ability to gather victim host information (T1592) [41], which is explained at length in §IV.

Our results are summarized in Table V. In the FK scenario, the attacker is able to evade detection against our three exemplar systems. In the LK scenario, the attacker completely evades detection against StreamSpot and Unicorn. Against ProvDetector, an anomaly is not detected and the paths selected for embedding do not contain malicious behavior; however, some of the attacker's innocuous website visits are included in the top 20 most anomalous paths. As a result, we consider this to be a qualified success; because an alarm is not raised and the most anomalous paths appear to be legitimate activities, it is unlikely that the attack would be detected.

The results of this experiment demonstrate that our attacker does not need a general solution to feature space inversion to launch a successful evasion attempt. Instead, they can use the adversarial sample generated by our methodology as a guide, and then approximate those behaviors on the target system by eliciting higher-level events such as page loads. Further, the results from the limited knowledge scenario demonstrate that this approach is potentially viable even with only coarse-grained knowledge of the target system.

L. Comparison to Domain-General Attacks

The above results indicate that Prov-HIDS are broadly susceptible to evasion, but it is not yet clear whether our specialized attack strategies are necessary. We now consider the applicability of domain-general graph evasion strategies.

Domain-General Graph Evasion. A survey of graph evasion methods is given in Table VI, following the taxonomy presented by Sun et al. [67]. The majority of graph evasion systems target node classifiers [56, 57, 58, 59, 60, 61, 62, 63]. As we are not aware of any Prov-HIDS that perform node classification, we rule out these techniques. Amongst whole-graph classification strategies, two are gradient-based attacks

Attack Method	Cls Tsk	Ptrb Typ	Mdl Acs	Grdnt Bsd?	Grph Typ	Prov-HIDS Compatible?
InfMax [56]	N	GA	C	No	Any	✖
EDA [57]	N	E	O	No	Any	✖
GF-Attack [58]	N	E	C	No	Any	✖
ReWatt [59]	N	E	C	No	Any	✖
EpoAtk [60]	N	E	O	Yes	Any	✖
AGA-GAN [61]	N	E	C	No	Any	✖
SRL [62]	N	E	C	No	Any	✖
CD-ATTACK [63]	N	E	C	No	Any	✖
Tang et al. [64]	G	GA	C	Yes	Any	✖
Xu et al. [65]	G	N+E	O	Yes	Any	✖
TNI [66]	G	N+E	O	No	Spl	✖
RL-S2V [24]	G	E	C	No	Any	?
Our Approach	G	N+E	O	No	Any	✔

TABLE VI: **Related Graph Evasion Attacks**— *Classification Task (Cls Tsk)* is either on nodes (N) or on the entire graph (G); *Perturbation Type (Ptrb Typ)* describes the graph properties transformed: Edge (E), Node (N), and Graph Attributes (GA); *Model Access (Mdl Acs)* denotes whether an open (O)- or closed (C)-knowledge model is assumed; *Gradient Based (Grdnt Bsd?)* indicates whether a graph neural network is assumed; and *Graph Type (Grph Typ)* designates whether a model can work on any (*Any*) graph type or only against a special (*Spl*) type.

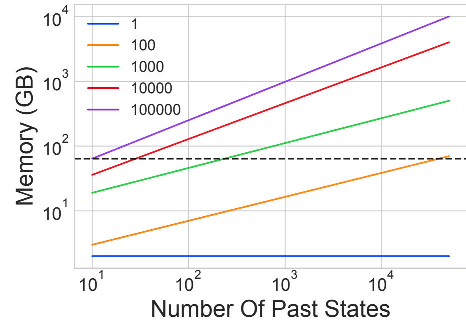


Fig. 8: Memory overheads incurred by RL-S2V as the number of saved past states (x-axis) and edge additions (line color) increases. The black dashed line denotes the available memory on our test server. RL-S2V can retain a large number of past states for small graph transformations; however, its per-state memory requirements become unwieldy when attempting transformations of the size required to evade Prov-HIDS.

against graph neural networks; these approaches may work against SIGL [21], but not the seven other Prov-HIDS in Table I. Among the two remaining systems, the Targeted Noise Injection (TNI) [66] "edge mirroring" technique is thematically similar to our methodology. However, it is designed for a bipartite graph classification task; provenance graphs are not bipartite and encode significantly more complex structures, leading us to believe that TNI would require significant modification to work on Prov-HIDS. Of the surveyed approaches, only Dai et al.'s reinforcement learning based RL-S2V [24] appears immediately applicable to Prov-HIDS.

RL-S2V. The objective of RL-S2V is to learn a policy that can successfully modify any input drawn from the same distribution. Given a state s representing graph G with $m-1$ modifications, RL-S2V learns the expectations over the probabilities of possible edge additions on G. After m modifications, RL-S2V updates its parameters based on the response from the target classifier. RL-S2V cycles through millions of possible modifications to learn an optimal policy.

Experimental Setup. We attempt to apply RL-S2V to the Prov-HIDS evasion problem using a minimally modified ver-

m	Execution Time Budget	Past State Budget	Training Iterations	Time Per Iteration (sec)
1	7 days	50000	3,024,000	0.2
100	7 days	5000	201,600	3
1,000	7 days	500	20,160	30
10,000	7 days	100	2,240	270
100,000	7 days	10	604	1000

TABLE VII: Training performance of RL-S2V as the number of edge additions (m) increases. Each configuration was run for the same amount of time and the past state budget was modified to prevent it from hitting a memory wall.

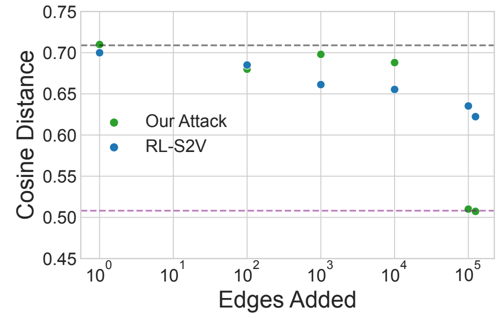


Fig. 9: Performance between our attack technique and RL-S2V. A summary of notation is given in Fig. 5. Briefly, the attack attempt must reach the purple dashed line to evade detection. The RL-S2V system was unable to successfully evade detection, w hile our strategy is successful above $125,000$ edges.

sion of the authors' source [68]. To advantage RL-S2V as best as we could, we tested against the StreamSpot Prov-HIDS, which evaluated samples much faster than the other systems, in the static graph classification setting. We also made use of the modified version of the relatively small StreamSpot dataset, training on all benign behaviors and testing against a single attack graph. While our original experiments used a subset of the benign behaviors for training (§VI-B), by including all behaviors during training, we made it *easier* for RL-S2V by increasing the variance among benign activity. RL-S2V is not designed for property graphs that carry edge attributes, so we down-sampled the StreamSpot dataset such that edges denoted attribute-free information flows. For example, any system call from a process to a file (e.g., `write`, `writev`, etc.) was mapped to an unlabeled edge between those entities.

In our initial experiments, we observed a halt in the training routine due to hitting a memory wall – as we scaled up RL-S2V to explore the very large graph transformations needed to evade Prov-HIDS, the algorithm's retention of past states became prohibitively costly. This trade-off between transformation size, past states, and memory cost is visualized in Fig. 8. To explore state changes of $100,000$ edges, RL-S2V's default configuration of retaining $50,000$ past states requires $10,000$ GB of RAM. Because it is necessary to explore edge modifications of this size according to our own results, it was necessary to modify RL-S2V's past state "budget" based on the size of the transformation being explored. For each transformation size, we used the analysis in Fig. 8 to determine the past state budget that could be supported by 64 GB of system RAM. Further, each configuration of RL-S2V was given seven days to train. As result of that, the number of training iterations varied based on the speed with which RL-S2V could complete a single iteration for a given transformation size. This information is summarized in Table VII.

Experimental Results. Fig. 9 reports on RL-S2V's best evasion attempt at the conclusion of training. Although the cosine distance to the nearest cluster decreases nominally with larger transformations, *RL-S2V is unable to efficiently learn a discernible pattern from the data and thus cannot successfully evade the StreamSpot Prov-HIDS.* While evasion may eventually become possible with larger graph transformations or more training time, the prohibitive cost of training RL-S2V makes this far less practical than our approach. Similarly, adapting RL-S2V to consider additional perturbation types like nodes or attributes would vastly expand the search space, imposing even more training overhead. Following our experiments, we reached out to the RL-S2V authors to see if they could suggest an alternative configuration. They explained that the experiments conducted in [24] considered graph transformations of at most $m = 1$; thus, they found it unlikely that RL-S2V would succeed at our task.

Remarks. These results underscore the difficulties of applying domain-general evasion attacks to provenance graph classification. Due to their size, complexity, and rich property space, provenance graph classification is a fundamentally different task than many other graph classification challenges. Of course, owing to the widespread success of adversarial examples on graph models, we fully anticipate their use in future attacks against Prov-HIDS. However, blindly applying existing adversarial models to Prov-HIDS, even if successful, does not elucidate the structural weaknesses of the systems under test. Our work looks to identify the pitfalls in current Prov-HIDS designs to inspire further work in the area.

Evasion Step	Time (sec)
Load the Pre-attack Graph	0.4334
Load Benign Graphs	0.0007
Find the Insertion Point	0.5490
Inject Benign Substructures	0.8325
Insert the Attack Payload's Substructures	0.0010
Total	**1.8166**

TABLE VIII: Runtime performance for each step in our evasion strategy.

M. Runtime Performance

Table VIII reports the time required to perform a single attempt to evade the StreamSpot Prov-HIDS on the StreamSpot dataset. *Load the Pre-attack* Graph and *Load Benign Graphs* denote the time required for the attacker to load and parse the initial attack graph and the benign training set, respectively. The initial attack graph describes how the attacker takes control of a process on the victim's system, but does not include the remainder of the attack. *Find the Insertion Point* returns the edge/node in the initial attack graph that represents the process under the attacker's control. *Inject Benign Substructures* corresponds to the injection of substructures, starting at the insertion point, that simulate benign activity. These substructures are sampled from the benign training set. These times are linearly dependent to the number of edges inserted. In this example, we inserted $300K$ edges into the attack graph. *Insert the Attack Payload's Substructures* is the final step in the evasion where the attacker conducts the rest of the attack. These steps are discussed in more detail in §VI-C.

Compared to domain-general adversarial sample approaches, our attack is much more time and space efficient. Consider the previously reported evasion attempts by RL-S2V, shown in Table VII. A single *unsuccessful* evasion attempt that is roughly the necessary transformation size, $m = 100,000$, requires 16.67 minutes to generate and test. In contrast, our methodology is able to generate a sufficient evasion attempt in less than 2 seconds. While other (non-black box) adversarial sample tools may offer stronger performance than RL-S2V, this result demonstrates the efficiency of our technique relative to commodity off-the-shelf tools.

VII. Discussion

A. Applicability to Other Prov-HIDS

While we evaluate five exemplar Prov-HIDS, we briefly remark on the feasibility of our methods on other systems. FRAPpuccino [25] and PIDAS [27] both weigh events equally within an inspection window, i.e., using unweighted encoding (§V-B1). Thus, using our first gadget to introduce benign events/substructures to the inspection window will likely force misclassification. Regarding our attack on distributional encoding (§V-B2), P-Gaussian [26] normalizes its graph-wide anomaly score across all paths in the graph. Therefore, injecting benign substructures will create more low-score paths, reduce the average score, and eventually force misclassification. We also suspect that either of our first two gadgets would frustrate attack reconstruction for Hercule's [28] log correlation mechanism; this is because Hercule assumes that edges between malicious and benign communities of events are infrequent, but this assumption is confounded if the attacker injects a large number of seemingly benign substructures.

Finally, we note that our exploration of mimicry gadgets is not exhaustive; our general attack methodology suggests other attacks against Prov-HIDS. For example, path-based systems (e.g., [19, 20, 26, 27]) suffer from a common design flaw in which they cannot represent suspicious activity that occurs along branching causal paths. As a result, an adversary could transform their attack logic to distribute its payload across a set of cooperating processes, as suggested by De Gaspari et al.'s Naked Sun attack [69], to create more attack paths with lower anomaly scores. While we expect that such an evasion attempt would be successful against these systems, we pursue different approaches, as Naked Sun requires modifying the core attack semantics whereas our gadgets can be applied transparently.

B. Threats to Validity

Most of our experiments assume that the attacker had access to system logs (i.e., training data). As explained in our threat model (§IV), accessing host information like logs is a common reconnaissance technique that is within the abilities of many intruders. Further, we showed in our live attack demonstration (§VI-K) that even limited coarse-grained knowledge about host activities, e.g., the presence of a certain process, will often be sufficient. Thus, privileged log access is not a strict requirement for our evasion methodology. Knowledge of the Prov-HIDS' decision threshold is also not required, since an attacker can conservatively estimate the sufficient transformation in exchange for a less efficient evasion attack.

C. Potential Mitigation Strategies

As seen in Table I, early Prov-HIDS research has gravitated towards *whole-graph* classification. Systems classifying lower-level graph structures such as nodes, edges, and subgraphs may be robust to our evasion strategy. At such a granularity, the additional substructures added by the attacker would have a difficult time affecting the classification of the anomalous nodes within the attack graph. The detection of finer-grained behaviors would also bring the usage model of Prov-HIDS more in line with commercial endpoint detection products [3].

The inability for Prov-HIDS to account for mimicry attacks is surprising, given the tremendous promise of provenance-based causal analysis. We attribute these vulnerabilities to a disconnect between how provenance has been used in the auditing and detection literature. For system auditing, researchers have leveraged domain knowledge to dramatically improve the space efficiency and precision of provenance analysis. In contrast, at present, many of these Prov-HIDS leverage "out of the box" graph learning solutions that do not account for the unique constraints of system intrusion scenarios. For whole-graph classification systems to become more resilient, security researchers may need to develop bespoke solutions that better leverage the properties of provenance graphs. For example, one path forward might be to integrate into intrusion detection the notions of root cause and impact analysis, which currently are not reflected in state-of-the-art methods of graph learning.

VIII. Related Work

Intrusion detection is among the canonical challenges of computer security. At the host layer, host intrusion detection systems (HIDS) can be classified into three broad categories: signature-based (e.g., antivirus [70, 71, 72]), rule-based (e.g., MITRE ATT&CK [39]), and anomaly-based. While signature-based approaches scan data on the disk for evidence of malware, rule- and anomaly-based systems both examine behavioral activity on the system for evidence of intrusion.

Our focus is on anomaly-based HIDS, which in addition to system calls [11, 13, 15, 73, 74, 75, 76, 77, 78] have also analyzed various host context [77, 79, 80], including call stack information [81], argument dataflows [13], and process configuration and environment [82]. Myriad techniques have been proposed for syscall anomaly detection based on policy specification [83, 84], sequence learning [11, 15, 85], rule induction [73], finite state automaton [76], and hidden Markov models [86]. Other work considers additional factors like the right length for syscall sequences [74, 77], ensemble and randomized classifiers [75, 87, 88], false positive reduction [89], multi-log analysis [90], and alert correlation [78, 91, 92].

The skepticism about HIDS' real-world efficacy has existed for nearly as long as the existence of HIDS themselves. One early concern noted that the length of learned sequences in syscall-based detectors was arbitrarily small and set through anecdotal testing [12]. Wagner and Soto's groundbreaking work on *mimicry attacks* operationalized this fear, demonstrating methods for malware to insert no-op events to avoid detection. Both Wagner and Soto [14] and Tan et al. [93] employ this method to subvert Forrest et al.'s behavior-based IDS [11, 15]. Later efforts semi-automated the process of mimicry attack generation through static binary analysis [16].

System auditing is a topic of resurgent interest in computer security due to the application of *data provenance* concepts to audit logs. Recent work has presented threat detection systems based on provenance graph analysis. In addition to the graph learning systems presented in Table I, we note also that a number of Prov-HIDS make use of heuristic-based detection, including Holmes [31], NoDoze [30] RapSheet [94], and Poirot [95]. Reasoning about evasion vulnerabilities in such systems is more difficult due to their heuristic nature; however, while heuristic detectors cannot be evaded by adding additional events, they are prone to high false positive rates and suffer from threat alert fatigue problems [5, 96]. This can cause detected attacks to go uninvestigated [2], as was the case in the 2013 Target data breach [97]. Thus, it is especially important for evasion vulnerabilities in heuristic Prov-HIDS to be evaluated in the context of organization-scale datasets, which is beyond of the scope of this work.

IX. Conclusion

We demonstrate the feasibility of mimicry attacks on provenance graph host intrusion detection systems based on analysis of common design features. Our experimental results show that our evasion strategies are practical and that successful evasion is consistently possible. We open-source our code and data to serve as a benchmark for future work in this field.

Acknowledgment

We would like to thank the anonymous reviewers for their helpful feedback to improve the paper. This work was supported by NSF under contracts CNS-16-57534, CNS-17-50024 and CNS-20-55127. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of their employers or the sponsors.

References

[1] S. Morgan, "Global Cybersecurity Spending Predicted To Exceed $1 Trillion From 2017-2021," https://cybersecurityventures.com/cybersecurity-market-report/, 2019.

[2] Crowdstrike, "Why Dwell Time Continues to Plague Organizations," https://www.crowdstrike.com/blog/why-dwell-time-continues-to-plague-organizations/, 2019.

[3] Gartner Peer Insights, "Endpoint Detection and Response Solutions Market," https://www.gartner.com/reviews/market/endpoint-detection-and-response-solutions, 2019.

[4] T. Hiroki, S. Yoshiaki, K. Koji, and A. Takayoshi, "Automated Security Intelligence (ASI) with Auto Detection of Unknown Cyber-Attacks," *NEC Technical Journal*, vol. 11, 2016.

[5] Fireeye, "Incident Investigation," https://www.fireeye.com/solutions/incident-investigation.html, 2019.

[6] "Automated Incident Response: Respond to Every Alert," https://swimlane.com/blog/automated-incident-response-respond-every-alert/, 2019.

[7] Malwarebytes Inc., "Malwarebytes," https://www.malwarebytes.com/business/edr, Last accessed April 2022.

[8] Splunk Inc., "splunk," https://www.splunk.com, Last accessed August 2018.

[9] T. S. Bernard, T. Hsu, N. Perlroth, and R. Lieber, "Equifax Says Cyberattack May Have Affected 143 Million in the U.S." https://www.nytimes.com/2017/09/07/business/equifax-cyberattack.html, 2019.

[10] *Target Missed Warnings in Epic Hack of Credit Card Data*. [Online]. Available: https://bloom.bg/2KjElxM.'

[11] 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*, May 1996, pp. 120–128.

[12] K. M. C. Tan and R. A. Maxion, ""why 6?" defining the operational limits of stide, an anomaly-based intrusion detector," in *Proceedings 2002 IEEE Symposium on Security and Privacy*, May 2002, pp. 188–201.

[13] S. Bhatkar, A. Chaturvedi, and R. Sekar, "Dataflow anomaly detection," in *2006 IEEE Symposium on Security and Privacy (S P'06)*, May 2006, pp. 15 pp.–62.

[14] 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*, ser. CCS '02. New York, NY, USA: Association for Computing Machinery, 2002, pp. 255–264. [Online]. Available: https://doi.org/10.1145/586110.586145

[15] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting intrusions using system calls: alternative data models," in *Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No.99CB36344)*, May 1999, pp. 133–145.

[16] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna, "Automating mimicry attacks using static binary analysis," in *Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14*, ser. SSYM'05. USA: USENIX Association, 2005, p. 11.

[17] 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*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 1035–1044. [Online]. Available: https://doi.org/10.1145/2939672.2939783

[18] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "Unicorn: Runtime Provenance-Based Detector for Advanced Persistent Threats," in *27th ISOC Network and Distributed System Security Symposium*, ser. NDSS'20, February 2020.

[19] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Zhen, W. Cheng, C. A. Gunter, and H. chen, "You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis," in *27th ISOC Network and Distributed System Security Symposium*, ser. NDSS'20, February 2020.

[20] Y. Xie, D. Feng, Y. Hu, Y. Li, S. Sample, and D. Long, "Pagoda: A hybrid approach to enable efficient real-time provenance based intrusion detection in big data environments," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 6, pp. 1283–1296, 2020.

[21] X. Han, X. Yu, T. Pasquier, D. Li, J. Rhee, J. Mickens, M. Seltzer, and H. Chen, "SIGL: Securing software installations through deep graph learning," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 2345–2362. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/han-xueyuan

[22] E. Manzoor, S. M. Milajerdi, and L. Akoglu, "StreamSpot: Detecting network anomalies in edge streams (Source Code and Data)," https://sbustreamspot.github.io/, 2016.

[23] D. I2O, "Transparent computing engagement 5 data release," https://github.com/darpa-i2o/Transparent-Computing, 2020.

[24] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song, "Adversarial attack on graph structured data," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 1115–1124. [Online]. Available: http://proceedings.mlr.press/v80/dai18b.html

[25] 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)*. Santa Clara, CA: USENIX Association, Jul. 2017. [Online]. Available: https://www.usenix.org/conference/hotcloud17/program/presentation/han

[26] Y. Xie, Y. Wu, D. Feng, and D. Long, "P-gaussian: Provenance-based gaussian distribution for detecting intrusion behavior variants using high efficient and real time memory databases," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 6, pp. 2658–2674, 2021.

[27] Y. Xie, D. Feng, Z. Tan, and J. Zhou, "Unifying intrusion detection and forensic analysis via provenance awareness," *Future Gener. Comput. Syst.*, vol. 61, no. C, pp. 26–36, Aug. 2016. [Online]. Available: http://dx.doi.org/10.1016/j.future.2016.02.005

[28] K. Pei, Z. Gu, B. Saltaformaggio, S. Ma, F. Wang, Z. Zhang, L. Si, X. Zhang, and D. Xu, "Hercule: Attack story reconstruction via community discovery on correlated log graph," in *Proceedings of the 32Nd Annual Conference on Computer Security Applications*, ser. ACSAC '16. New York, NY, USA: ACM, 2016, pp. 583–595. [Online]. Available: http://doi.acm.org/10.1145/2991079.2991122

[29] A. Alsaheel, Y. Nan, S. Ma, L. Yu, G. Walkup, Z. B. Celik, X. Zhang, and D. Xu, "ATLAS: A sequence-based learning approach for attack investigation," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 3005–3022. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/alsaheel

[30] 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 *26th ISOC Network and Distributed System Security Symposium*, ser. NDSS'19, February 2019.

[31] 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 2019 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2019. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP.2019.00026

[32] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly Detection: A Survey," *ACM Comput. Surv.*, vol. 41, no. 3, jul 2009. [Online]. Available: https://doi.org/10.1145/1541880.1541882

[33] D. Yang, B. Li, L. Rettig, and P. Cudré-Mauroux, "Histosketch: Fast similarity-preserving sketching of streaming histograms with concept drift," in *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2017, pp. 545–554.

[34] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International conference on machine learning*, 2014, pp. 1188–1196.

[35] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000, pp. 93–104.

[36] MITRE, "APT3," https://attack.mitre.org/groups/G0022/, 2019.

[37] ——, "APT29," https://attack.mitre.org/groups/G0016/, 2019.

[38] "Threat-based Defense," https://www.mitre.org/capabilities/cybersecurity/threat-based-defense, 2019.

[39] MITRE Corporation, "MITRE ATT&CK," https://attack.mitre.org, 2019.

[40] MITRE, "MITRE ATT&CK: Masquerading," https://attack.mitre.org/techniques/T1078, 2020.

[41] ——, "MITRE ATT&CK: Gather Victim Host Information," https://attack.mitre.org/techniques/T1592/, 2020.

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

[43] K. H. Lee, X. Zhang, and D. Xu, "LogGC: Garbage Collecting Audit Log," in *Proceedings of the 2013 ACM SIGSAC conference on Computer and Communications Security*, ser. CCS '13. New York, NY, USA: ACM, 2013, pp. 1005–1016. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516731

[44] S. Ma, X. Zhang, and D. Xu, "ProTracer: Towards Practical Provenance Tracing by Alternating Between Logging and Tainting," in *Proceedings of NDSS '16*, Feb. 2016.

[45] S. Ma, J. Zhai, F. Wang, K. H. Lee, X. Zhang, and D. Xu, "MPI: Multiple Perspective Attack Investigation with Semantic Aware Execution Partitioning," in *26th USENIX Security Symposium*, August 2017.

[46] W. U. Hassan, N. Aguse, M. Lemay, T. Moyer, and A. Bates, "Towards Scalable Cluster Auditing through Grammatical Inference over Provenance Graphs," in *Proceedings of the 25th ISOC Network and Distributed System Security Symposium*, ser. NDSS'18, San Diego, CA, USA, February 2018.

[47] Y. Kwon, F. Wang, W. Wang, K. H. Lee, W.-C. Lee, S. Ma, X. Zhang, D. Xu, S. Jha, G. Ciocarlie, A. Gehani, and V. Yegneswaran, "Mci: Modeling-based causality inference in audit logging for attack investigation," in *Proc. of the 25th Network and Distributed System Security Symposium (NDSS'18)*, 2018.

[48] A. Bates, W. U. Hassan, K. R. Butler, A. Dobra, B. Reaves, P. Cable, T. Moyer, and N. Schear, "Transparent Web Service Auditing via Network Provenance Functions," in *26th World Wide Web Conference*, ser. WWW'17, Perth, Australia, April 2017.

[49] X. Han, T. Pasqueir, A. Bates, J. Mickens, and M. Seltzer, "Unicorn: Runtime Provenance-Based Detector for Advanced Persistent Threats (Source Code and Data)," https://github.com/crimson-unicorn, 2018.

[50] B. Jacob, P. Larson, B. Leitao, and S. Da Silva, "Systemtap: instrumenting the linux kernel for analyzing performance and functional problems," *IBM Redbook*, vol. 116, 2008.

[51] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer, "Provenance-aware Storage Systems," in *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, ser. Proceedings of the 2006 Conference on USENIX Annual Technical Conference, Jun. 2006.

[52] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[53] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 4602–4609.

[54] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[55] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing properties of adversarial ml attacks in the problem space," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1332–1349.

[56] J. Ma, J. Deng, and Q. Mei, "Adversarial Attack on Graph Neural Networks as An Influence Maximization Problem," *CoRR*, vol. abs/2106.10785, 2021. [Online]. Available: https://arxiv.org/abs/2106.10785

[57] S. Yu, J. Zheng, J. Chen, Q. Xuan, and Q. Zhang, "Unsupervised Euclidean Distance Attack on Network Embedding," in *2020 IEEE Fifth International Conference on Data Science in Cyberspace (DSC)*, 2020, pp. 71–77.

[58] H. Chang, Y. Rong, T. Xu, W. Huang, H. Zhang, P. Cui, W. Zhu, and J. Huang, "A restricted black-box adversarial framework towards attacking graph embedding models," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, pp. 3389–3396, Apr. 2020. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/5741

[59] Y. Ma, S. Wang, L. Wu, and J. Tang, "Attacking Graph Convolutional Networks via Rewiring," *CoRR*, vol. abs/1906.03750, 2019. [Online]. Available: http://arxiv.org/abs/1906.03750

[60] X. Lin, C. Zhou, H. Yang, J. Wu, H. Wang, Y. Cao, and B. Wang, "Exploratory Adversarial Attacks on Graph Neural Networks," in *2020 IEEE International Conference on Data Mining (ICDM)*, 2020, pp. 1136–1141.

[61] J. Chen, D. Zhang, and X. Lin, "Adaptive Adversarial Attack on Graph Embedding via GAN," in *Security and Privacy in Social Networks and Big Data*, Y. Xiang, Z. Liu, and J. Li, Eds. Singapore: Springer Singapore, 2020, pp. 72–84.

[62] L. Zhang, P. Liu, and Y.-H. Choi, "Semantic-preserving Reinforcement Learning Attack Against Graph Neural Networks for Malware Detection," 2020.

[63] J. Li, H. Zhang, Z. Han, Y. Rong, H. Cheng, and J. Huang, "Adversarial Attack on Community Detection by Hiding Individuals," in *Proceedings of The Web Conference 2020*, ser. WWW '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 917–927. [Online]. Available: https://doi.org/10.1145/3366423.3380171

[64] H. Tang, G. Ma, Y. Chen, L. Guo, W. Wang, B. Zeng, and L. Zhan, "Adversarial Attack on Hierarchical Graph Pooling Neural Networks," *CoRR*, vol. abs/2005.11560, 2020. [Online]. Available: https://arxiv.org/abs/2005.11560

[65] X. Xu, X. Du, and Q. Zeng, "Attacking Graph-Based Classification without Changing Existing Connections," in *Annual Computer Security Applications Conference*, ser. ACSAC '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 951–962. [Online]. Available: https://doi.org/10.1145/3427228.3427245

[66] Y. Chen, Y. Nadji, A. Kountouras, F. Monrose, R. Perdisci, M. Antonakakis, and N. Vasiloglou, "Practical attacks against graph-based clustering," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 1125–1142. [Online]. Available: https://doi.org/10.1145/3133956.3134083

[67] L. Sun, Y. Dou, C. Yang, J. Wang, P. S. Yu, L. He, and B. Li, "Adversarial attack and defense on graph data: A survey," *arXiv preprint arXiv:1812.10528*, 2018.

[68] H. Dai, "Graph adversarial attack," https://github.com/Hanjun-Dai/graph_adversarial_attack, 2021.

[69] F. De Gaspari, D. Hitaj, G. Pagnotta, L. De Carli, and L. V. Mancini, "The naked sun: Malicious cooperation between benign-looking processes," in *Applied Cryptography and Network Security: 18th International Conference, ACNS 2020, Rome, Italy, October 19–22, 2020, Proceedings, Part II*. Berlin, Heidelberg: Springer-Verlag, 2020, pp. 254–274. [Online]. Available: https://doi.org/10.1007/978-3-030-57878-7_13

[70] "ClamAV Anti-Virus," https://www.clamav.net/, 2007.

[71] O. Erdogan and Pei Cao, "Hash-av: fast virus signature scanning by cache-resident filters," in *IEEE Global Telecommunications Conference*, ser. GLOBECOM, vol. 3, 2005.

[72] J. Oberheide, E. Cooke, and F. Jahanian, "Cloudav: N-version antivirus in the network cloud," in *Proceedings of the 17th Conference on Security Symposium*, ser. SS'08. USA: USENIX Association, 2008, pp. 91–106.

[73] W. Lee and S. J. Stolfo, "Data mining approaches for intrusion detection," in *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7*, ser. SSYM'98. USA: USENIX Association, 1998, p. 6.

[74] A. Wespi, M. Dacier, and H. Debar, "Intrusion detection using variable-length audit trail patterns," in *Recent Advances in Intrusion Detection*, H. Debar, L. Mé, and S. F. Wu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 110–129.

[75] T. Bass, "Intrusion detection systems and multisensor data fusion," *Commun. ACM*, vol. 43, no. 4, pp. 99–105, Apr. 2000. [Online]. Available: http://doi.acm.org/10.1145/332051.332079

[76] R. Sekar, M. Bendre, D. Dhurjati, and P. Bollineni, "A fast automaton-based method for detecting anomalous program behaviors," in *Proceedings 2001 IEEE Symposium on Security and Privacy. S P 2001*, May 2001, pp. 144–155.

[77] D. Gao, M. K. Reiter, and D. Song, "On gray-box program tracking for anomaly detection," in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, ser. SSYM'04. USA: USENIX Association, 2004, p. 8.

[78] G. Gu, A. A. Cárdenas, and W. Lee, "Principled reasoning and practical applications of alert fusion in intrusion detection systems," in *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '08. New York, NY, USA: ACM, 2008, pp. 136–147. [Online]. Available: http://doi.acm.org/10.1145/1368310.1368332

[79] F. Liu, Y. Wen, D. Zhang, X. Jiang, X. Xing, and D. Meng, "Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS 19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 1777–1794. [Online]. Available: https://doi.org/10.1145/3319535.3363224

[80] T. van Ede, H. Aghakhani, N. Spahn, R. Bortolameotti, M. Cova, A. Continella, M. van Steen, A. Peter, C. Kruegel, and G. Vigna, "Deepcase: Semi-supervised contextual analysis of security events," in *2022 2022 IEEE Symposium on Security and Privacy (SP) (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2022, pp. 614–631. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP46214.2022.00036

[81] H. H. Feng, O. M. Kolesnikov, P. Fogla, W. Lee, and Weibo Gong, "Anomaly detection using call stack information," in *2003 Symposium on Security and Privacy, 2003.*, May 2003, pp. 62–75.

[82] J. T. Giffin, D. Dagon, S. Jha, W. Lee, and B. P. Miller, "Environment-sensitive intrusion detection," in *Recent Advances in Intrusion Detection*, A. Valdes and D. Zamboni, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 185–206.

[83] C. Ko, G. Fink, and K. Levitt, "Automated detection of vulnerabilities in privileged programs by execution monitoring," in *Tenth Annual Computer Security Applications Conference*, Dec 1994, pp. 134–144.

[84] D. Wagner and R. Dean, "Intrusion detection via static analysis," in *Proceedings 2001 IEEE Symposium on Security and Privacy. S P 2001*, May 2001, pp. 156–168.

[85] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 1285–1298. [Online]. Available: https://doi.org/10.1145/3133956.3134015

[86] K. Xu, K. Tian, D. Yao, and B. G. Ryder, "A sharper sense of self: Probabilistic reasoning of program behaviors for anomaly detection with context sensitivity," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2016, pp. 467–478.

[87] J. E. Tapiador and J. A. Clark, "Masquerade mimicry attack detection: A randomised approach," *Computers & Security*, vol. 30, no. 5, pp. 297–310, 2011.

[88] W. Khreich, S. S. Murtaza, A. Hamou-Lhadj, and C. Talhi, "Combining heterogeneous anomaly detectors for improved software security," *Journal of Systems and Software*, vol. 137, pp. 415 – 429, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121217300420

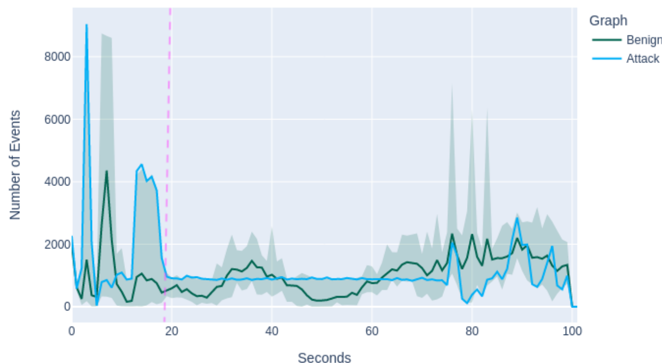


Fig. 10: The average rate of edges added per second for benign and evasion graphs. The green highlights represent the minimum and maximum number of edges per second among all benign graphs in the train set. The purple vertical line represents the attacker's *insertion point*.

	Benign	Evasion
Number of Nodes	57,297	57,395
Number Of Edges	121,107	121,427
Degree Assortativity Coefficient	-0.4202	-0.4218
Edge Density	0.00004	0.00004
Average Degree	2.11	2.12

TABLE IX: Statistical differences between benign and evasion graphs. The degree assortativity coefficient is the tendency of nodes with similar degrees to connect. The edge density is the total number of edges over all edges possible. The average degree is the number of edges over the number of nodes.

[89] G. P. Spathoulas and S. K. Katsikas, "Using a fuzzy inference system to reduce false positives in intrusion detection," in *International Conference on Systems, Signals and Image Processing*, 2009.

[90] T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda, "Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks," in *Proceedings of the 29th Annual Computer Security Applications Conference*, ser. ACSAC '13. New York, NY, USA: ACM, 2013, pp. 199–208. [Online]. Available: http://doi.acm.org/10.1145/2523649.2523670

[91] F. Valeur, G. Vigna, C. Kruegel, and R. A. Kemmerer, "Comprehensive approach to intrusion detection alert correlation," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 3, pp. 146–169, July 2004.

[92] A. Sadighian, J. M. Fernandez, A. Lemay, and S. T. Zargar, *ONTIDS: A Highly Flexible Context-Aware and Ontology-Based Alert Correlation Framework*. Cham: Springer International Publishing, 2014, pp. 161–177. [Online]. Available: https://doi.org/10.1007/978-3-319-05302-8_10

[93] K. M. C. Tan, K. S. Killourhy, and R. A. Maxion, "Undermining an anomaly-based intrusion detection system using common exploits," in *Recent Advances in Intrusion Detection*, A. Wespi, G. Vigna, and L. Deri, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 54–73.

[94] W. U. Hassan, A. Bates, and D. Marino, "Tactical Provenance Analysis for Endpoint Detection and Response Systems," in *41st IEEE Symposium on Security and Privacy (SP)*, ser. Oakland'20, May 2020.

[95] 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*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 1795–1812. [Online]. Available: https://doi.org/10.1145/3319535.3363217

[96] FireEye, Inc., "How Many Alerts is Too Many to Handle?" https://www2.fireeye.com/StopTheNoise-IDC-Numbers-Game-Special-Report.html, 2019.

[97] M. Riley, B. Elgin, D. Lawrence, and C. Matlack, "Target Missed Warnings in Epic Hack of Credit Card Data," https://bloom.bg/2KjElxM, 2019.

Appendix A
Covertness Against Statistical Tests

To successfully evade the exemplar systems, our evasion strategy requires transformations of tens to hundreds of thousands of edges. Doing so raises the question of whether our methodology could be detected through the use of a trivial statistical monitor. Fig. 10 reports the events per second that occur across all benign logs in the StreamSpot dataset, as compared to the events per second of a successful evasion attempt. The early spikes in the attack line (blue) correspond to the events in the *Load the Pre-attack Graph* step (§VI-C), the relatively flat portion corresponds to our insertion of benign substructures, and finally the more variable portion to the right corresponds to the injection of the attack payload's substructures. If we compare the blue attack line to the green shaded areas, which represent the maximum and minimum observed events per second in the benign logs, attack behaviors never extend beyond the bounds of expected normal activity in this dataset. While the flattened portion of the line does not appear naturalistic, the attacker controls these insertion points and could apply traffic shaping measures to create a more realistic curve. Larger and more diverse datasets, such as the DARPA Transparent Computing, are even noisier and therefore more permissive to the adversary.

We also ran several statistical test between the evasion graph and benign graphs for signs of abnormality, which we report in Table IX. Both the benign and evasion graphs have similar statistical properties. There is a negligible difference in the number of nodes and edges. Similarly, the degree assortativity coefficient (the tendency of similar nodes to connect), edge density (the number of edges over all edges possible), and the average degree between both sets are related. These findings reinforce the observation, from Fig. 5 and Fig. 6, that our evasion methodology appeared to be making the attack behavior *more consistent* with the statistical properties of the average benign graph.