SHADEWATCHER: Recommendation-guided Cyber Threat Analysis using System Audit Records

Jun Zeng[†] Xiang Wang[‡*] Jiahao Liu[†] Yinfang Chen[§] Zhenkai Liang[†*] Tat-Seng Chua[†] Zheng Leong Chua[¶]
[†]National University of Singapore [‡]University of Science and Technology of China [§]UIUC [¶]Independent Researcher
{junzeng, jiahao99, liangzk, chuats}@comp.nus.edu.sg xiangwang@ustc.edu.cn

2022 IEEE Symposium on Security and Privacy (SP) | 978-1-6654-1316-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/SP46214.2022.9833669

***Abstract*—System auditing provides a low-level view into cyber threats by monitoring system entity interactions. In response to advanced cyber-attacks, one prevalent solution is to apply data provenance analysis on audit records to search for anomalies (anomalous behaviors) or specifications of known attacks. However, existing approaches suffer from several limitations: 1) generating high volumes of false alarms, 2) relying on expert knowledge, or 3) producing coarse-grained detection signals.**

In this paper, we recognize the structural similarity between threat detection in cybersecurity and recommendation in information retrieval. By mapping security concepts of system entity interactions to recommendation concepts of user-item interactions, we identify cyber threats by predicting the preferences of a system entity on its interactive entities. Furthermore, inspired by the recent advances in modeling high-order connectivity via item side information in the recommendation, we transfer the insight to cyber threat analysis and customize an automated detection system, SHADEWATCHER. It fulfills the potential of high-order information in audit records via graph neural networks to improve detection effectiveness. Besides, we equip SHADEWATCHER with dynamic updates towards better generalization to false alarms. In our evaluation against both real-life and simulated cyber-attack scenarios, SHADEWATCHER shows its advantage in identifying threats with high precision and recall rates. Moreover, SHADEWATCHER is capable of pinpointing threats from nearly a million system entity interactions within seconds.

I. INTRODUCTION

There has been a rapid escalation of high-profile security threats that intentionally target large enterprises, such as the Equifax breach that resulted in a record number of user data stolen [1] and the Solarwinds hack whose scope and audacity are claimed unprecedented [2]. Often termed as *Advanced Persistent Threats* (APTs), they are carried out by skilled attackers with sophisticated cyber capabilities. To combat these threats, SIEM (Security Information and Event Management) system [3]–[5] is pervasively deployed. Such software monitors interactions among system entities (i.e., processes, files, and sockets) on end hosts as audit records, which collects evidence of attack footprints. Unfortunately, due to the ever-expanding scale of modern IT infrastructures, the volume of audit data is always overwhelming. For example, a typical commercial bank with 200,000 hosts can produce almost 70 PB audit records per year [6].

To facilitate attack investigation in large-scale host records, the research community proposes *data provenance* techniques to navigate audit records through a provenance graph that describes the history of a system's execution [7]–[9]. The rich contexts in provenance data enable analysts to perform causal analysis of system activities to detect intrusions, trace dependencies, and reason about security incidents. Threat detection is the typical starting step of an attack investigation. Based on how audit records are used, existing provenance-based detectors fall into three categories: 1) *Statistics-based detection* quantifies the suspiciousness degree of audit records through their rareness in provenance graphs [10]–[12]; 2) *Specification-based detection* matches audit records against a knowledge base of security policies associated with known attack patterns [13]–[19]; 3) *Learning-based detection* extends machine learning techniques to model benign behaviors and detect deviations from them [20]–[24].

Although existing solutions have shown promising detection performance, they bear several inherent drawbacks. Statistics-based detection is prone to high volumes of false alarms for rare yet normal system activities. This is because statistical analysis considers only direct (causal) connections in provenance graphs rather than subtle (semantic) relationships among system entities, which can be critical for threat analysis, especially when facing evolving behaviors. While specification-based detection can maintain low false-positive rates by defining *attack semantics* as security policies, such heuristics are time-consuming and error-prone to develop. According to a recent survey, policies tied to COTS SIEM products cover only 16% of public TTPs knowledge base [25]. Additionally, learning-based detection aims to comprehensively incorporate both the causality and semantics of audit records into threat analysis. Despite the high detection accuracy, current learning approaches produce detection signals at a coarse-grained level, e.g., behavior level. Consequently, tedious manual labor is required to review all audit records in individual behavior to determine if they indicate an actual attack [26].

The fundamental challenge in analyzing cyber threats is to conduct *comprehensive yet fine-grained* reasoning in a large number of audit records. Security analysts need to discern not only whether a certain audit record is the result of an attack but how it is associated with the malicious behavior. Threat actors typically induce unwanted behaviors with unintended system entity interactions deemed suspicious to analysts. Intuitively, intended interactions form the norm of behaviors and constitute the most "likely" behaviors to be observed. On the contrary, unintended behaviors deviate from the norm and contain "unlikely" interactions to be observed. As such, *cyber threats can be revealed by determining how likely a system*

[*]Corresponding authors; research done in National University of Singapore.

entity would interact with another entity. This likelihood of interactions can be estimated by exploiting causal connectivity in provenance graphs. Such connectivity, however, does not capture the hidden semantic meanings behind system entities that are necessary to uncover their relationships. To illustrate, consider the Linux */proc* file system. As */proc/25/stat* and */proc/27/stat* belong to different processes, they are typically disconnected in provenance graphs, being treated causally irrelevant by direct connections. However, both of them present status information about processes, which can be reflected when considering the border contexts around them.

We notice that a similar problem has been explored in the recommendation domain, where the primary goal is to *predict how likely a user would consume items*. Earlier recommendation systems [27], [28] assume that behaviorally similar users would share preferences on items so that they comprehend user preferences by finding similar users through historical user-item interactions. However, direct connections between users and items, termed as *first-order connectivity*, are insufficient to compare semantic similarities among different items. To address this problem, researchers further take into account *side information* of items, e.g., the genre of movies, for capturing item semantics. At its core is that side information can form *high-order connectivity* to link similar items disconnected in user-item interactions [29], [30]. Based on this advancement in recommendations, we aim to incorporate side information of system entities to interpret their interactions comprehensively. Although such auxiliary information is not explicitly encoded in provenance graphs, a recent study has demonstrated that the semantics of system entities can be revealed from the contexts in which they are used [31]. As such, we leverage contextual information as the underlying side information to profile system entities. Thereafter, system entities with similar contexts would be semantically correlated, despite being considered irrelevant in causal analysis.

By mapping cybersecurity concepts of system entity interactions and entity contextual information to recommendation concepts of user-item interactions and item side information, we can formulate cyber threat detection as a recommendation task. Especially, we observe that *semantically similar system entities would exhibit similar preferences on interactions*. For example, sensitive files (e.g., */etc/passwd* and */etc/shadow*) normally do not interact with public networks, which otherwise indicates data exfiltration [32]. Based on this observation, *threat detection can be further specified as predicting how likely a system entity would not "prefer" its interactive entities.* Note that in contrast to typical recommendation scenarios studying user preferences, threat detection targets interactions that system entities unlikely prefer, as such interactions are commonly strong attack indicators.

In this paper, we present SHADEWATCHER, the first system that analyzes cyber threats through recommendations on system entity interactions. SHADEWATCHER extracts *side information* of system entities using a *context-aware embedding model* [33] that unfolds the semantics of entities by their usage in a running system. To unveil system entity intents on interactions, SHADEWATCHER employs a recommendation model built upon *graph neural networks* [34] that exploits high-order connectivity by recursively propagating information from neighboring entities. Furthermore, SHADEWATCHER dynamically updates its models with analyst feedback on detection signals (i.e., potentially malicious interactions). This allows SHADEWATCHER to integrate false recommendations as additional supervision to improve its detection capabilities. As a semi-supervised approach, SHADEWATCHER is trained on a combination of unlabeled benign system entity interactions with labeled analyst feedback on false alarms. In a nutshell, SHADEWATCHER's novel utilization of recommendations makes it advantageous to existing detectors in that: 1) instead of counting historical frequency as a metric to estimate degrees of suspicion, SHADEWATCHER infers intrinsic semantics of system entities to discover anomalous interactions; 2) SHADEWATCHER provides an end-to-end solution to detect threats without prior knowledge of attacks; 3) SHADEWATCHER produces fine-grained detection signals that highlight the key indicator of an attack.

We implement SHADEWATCHER and evaluate its effectiveness and efficiency on four APT attacks generated by the TRACE team in the DARPA Transparent Computing (TC) program [35] and six cyber-attacks from recent literature [6], [31], [36] simulated in a testbed environment. Experimental results show that SHADEWATCHER effectively differentiates benign and malicious system entity interactions and detects cyber threats with high precision and recall rates. We also demonstrate that SHADEWATCHER is efficient enough to scale to an enterprise environment.

In summary, we make the following contributions:

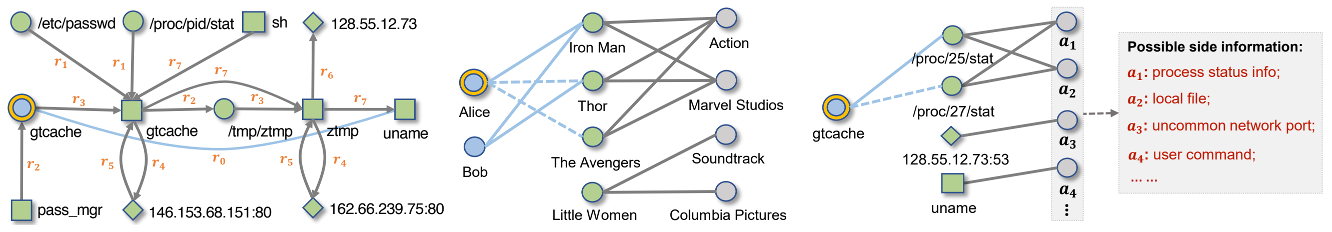
- We identify the task similarity between threat detection and recommendation. Establishing a mapping between these two domains is the key novelty of our approach, which opens up a new space for effective solutions in cyber threat analysis. Moreover, we recognize the concept of high-order information in audit records for modeling preferences of system entities on their interactive entities.
- We present SHADEWATCHER, the first system designed after the principle of recommendations, to analyze cyber threats using audit records in an automated and adaptive manner while providing appropriate abstractions highlighting key attack indicators.
- We implement SHADEWATCHER and conduct a systematic evaluation against both real-life and simulated attack scenarios. The results show that SHADEWATCHER achieves high effectiveness and efficiency in cyber threat analysis.

II. BACKGROUND & MOTIVATION

In this section, we use an APT attack to introduce challenges in existing threat detection solutions. Then, we present our insights of using recommendations to guide threat analysis.

A. Motivating Example

Browser extension with drakon dropper (Extension Backdoor for short) is an APT attack from the DARPA TC program



(a) System Provenance Graph (b) Movie Recommendation (c) High-order Information in Audit Records

Fig. 1: (a) A simplified provenance graph of Extension Backdoor, where r_1, r_2, r_3, r_4, r_5, r_6, and r_7 denote *read*, *write*, *load*, *send*, *recv*, *connect*, and *exec*, and r_0 reflects system entity interaction. (b) An example of recommending movies for *Alice*, where blue, green, and gray nodes stand for users, items, and side information. Solid/dashed blue edges show historical/recommended user-item interactions. (c) An illustration of system entity interactions with side information to form high-order connectivities.

red team vs. blue team adversarial engagement [35]. The attack begins with compromising a vulnerable password manager extension *pass_mgr* in *Firefox* while visiting malicious websites. The compromised extension then drops a malicious program *gtcache* on disk. During execution, the program exfiltrates user information */etc/passwd* to a public network. Meanwhile, it implants *ztmp* to collect system configurations and perform a port scan of target networks for internal reconnaissance.

Figure 1a shows a simplified provenance graph constructed from audit records in Extension Backdoor. Nodes in the graph represent system entities, where rectangles, ovals, and diamonds indicate processes, files, and sockets, respectively. Gray edges denote system calls orientated in the direction of information flows. To reduce clutter, we use two individual nodes in lieu of */proc/pid/stat* with different process IDs and *128.55.12.73* with different network ports. The provenance graph provides a promising representation to navigate large-scale audit records. It enables security analysts to perform backward and forward information flow tracking to discover root causes of security incidents and their ramifications [10].

B. Challenge to Existing Solutions

Provenance-based detection excels at extracting potentially malicious behaviors from provenance graphs. However, we identify several inherent limitations of existing approaches.

- *Statistics-based detection:* Recent studies observe that security incidents in attack campaigns are usually uncommon system activities [10]–[12]. Therefore, they quantify audit records' degrees of suspicion by their historical frequency. Though simple and effective, a primary concern is the number of false positives generated using statistical analysis. For example, when *gtcache* first reads */proc/27/stat* in Figure 1a, an alarm is raised as this activity has never been seen before, although it represents a perfectly normal process status retrieval. From this example, it is easy to see the key shortfall of statistics-based methods, which is identifying audit records rare in the history of system execution but fails to differentiate normal records from *previously unobserved yet semantically relevant* activities.
- *Specification-based detection:* Specification-based detectors hunt down cyber threats by matching audit records against a knowledge base of security policies that describe attack semantics. While such detection can maintain a low false-positive rate [13], developing security policies is time-consuming and inevitably requires domain expertise. Regarding our motivating example, RapSheet [14] develops over ten hand-crafted TTPs (Tactics, Techniques, and Procedures) queries for *kill chain* search; Morse [17] initializes confidentiality and integrity tags of six million system entities for tag propagation; Poirot [19] extracts query graphs from a six-page cyber threat intelligence report [35] for graph alignment. We also note that the quality of resultant policies can be highly varied due to factors such as the expert's subjective interpretation of attacks, different levels of competencies, and even just plain human mistakes.
- *Learning-based detection:* Current learning-based detectors mostly train a model of benign behaviors and detect deviations as cyber-attacks [20]–[23]. While these approaches can achieve high detection accuracy by incorporating the semantics of audit records into threat analysis, no learning solutions to date provide explicable results or insights on the key indicators of an attack, undermining the usefulness in practice. Specifically, extensive manual efforts are required to review audit records in behavior to locate particular system activities triggering a detection signal. For example, as Unicorn [21] analyzes APT attacks upon a long duration of system execution, analysts need to sift through thousands of audit records to identify and validate the indicator of a single detection signal.

C. Threat Detection as Recommendation

System entity interaction serves as the bedrock in our approach, where we utilize the observation that interaction with low likelihood can be naturally identified as a potential cyber threat. For example, executables downloaded by browsers (e.g., *gtcache*) commonly do not run sensitive user commands (e.g., *uname*), which otherwise signifies malicious execution [37]. Provenance-based solutions typically use causal connections in provenance graphs to interpret system entity interactions. However, such causality is insufficient to reveal the semantic relationship between two system entities. In particular, causally disconnected entities (e.g., */proc/pid/stat* with different *pid*) are not necessarily semantically irrelevant.

We discover that a similar problem has been explored in the recommendation domain. Figure 1b illustrates a movie recommendation scenario, where *Alice* is the target user to provide recommendations for. The user-item interactions, *Alice*→*Iron Man* and *Bob*→*Iron Man*, indicate the behavioral similarity between *Alice* and *Bob*. Based on the assumption that *behaviorally similar users would share preferences on items* [28], earlier recommendation systems predict that *Alice* is in favor of *Thor* as *Bob* likes it. However, considering the recommendations of relevant items to a particular user, user-item interactions are insufficient as they cannot compare item semantic similarity. To address this issue, the recently proposed methods [29], [30] leverage item side information, such as movie genre and studio, to form *high-order connectivities* that link semantically similar items. For example, the two-order connections, *Iron Man*→*Action*→*The Avengers* and *Iron Man*→*Marvel Studios*→*The Avengers*, suggest that *Alice* may prefer *The Avengers* as its side information is identical to that of *Iron Man*.

Similarly, an intuitive way to better understand system entity interactions is to identify side information of system entities to form high-order connectivity. For example, if */proc/27/stat* were correlated with */proc/25/stat* through side information (e.g., *process status info* in Figure 1c), we would determine that they share the probability of interacting with other system entities (e.g., *gtcache*). However, side information is not explicitly encoded in original provenance graphs. To extract such knowledge, we take inspiration from a recent study [31], which infers the semantics of system entities from their usage contexts. More specifically, we regard contextual information as auxiliary knowledge to profile system entities. As such, causalities of system entities form user-item interactions as in the recommendation, while system contexts provide side information to form high-order connectivity. Note that since contextual information of system entities is reflected in their neighbors in provenance graphs, we capture *high-order connectivity as a multi-hop path correlating neighboring entities.*

Therefore, we can formulate cyber threat detection as a recommendation problem, which models the likelihood of interactions between two system entities, predicting interactions that entities normally do not like as cyber threats. For example, detecting the attack of Extension Backdoor in Figure 1a becomes recommending system entities with which *gtcache* unlikely interacts in Figure 1c. Motivated by this insight, we are able to bridge the threat detection and recommendation domains and leverage the advances in recommendation methods to help comprehend system entity interactions.

III. Problem Definition

We first formally define several basic concepts required to understand how ShadeWatcher provides recommendations. Then, we introduce the problem statement and threat model.

A. Basic Concept

Provenance Graph: Audit records are a set of log entries that describe the history of a system's execution. Each record represents an activity at the granularity of system calls. Typically, it is formulated as a 3-tuple (src, rel, dst), where $src, dst \in \mathcal{E} = \{process, file, socket\}$, and $rel \in \mathcal{R} = \{clone, write, ...\}$. For example, a network service scanning activity in Figure 1a is defined as (*ztmp*, *connect*, *128.55.12.73:54*). Data provenance organizes audit records in the form of a provenance graph, a directed acyclic graph composed of (src, rel, dst) tuples. Formally, we define a provenance graph as $\mathcal{G}_P = \{(src, rel, dst) |\ src, dst \in \mathcal{E}, rel \in \mathcal{R}\}$.

System Entity Interaction: Causal connections in a provenance graph reflect interactions among system entities. For example, a chain of edges connecting *gtcache* and *uname* in Figure 1a indicate that they are interactive. In recommendation scenarios, user-item interactions are commonly presented as a bipartite graph to preserve the collaborative filtering signal [38]. We thus also define system entity interactions as a bipartite graph, $\mathcal{G}_B = \{(e, y_{ee'}, e') | e, e' \in \mathcal{E})\}$. The link $y_{ee'} = 1$ shows that entity e has interacted with entity e', while $y_{ee'} = 0$ the opposite. Note that a system entity interaction represents not only explicit data dependency but also implicit control dependency. For example, the aforementioned interaction between *gtcache* and *uname* shows a control dependency where *gtcache* manipulates *ztmp* to execute *uname*.

Order of Connectivity: Here we define the concept of a knowledge graph (KG) that encodes system entity contexts and interactions into a unified relational graph. More specifically, we convert a valid interaction (i.e., $y_{ee'} = 1$) in the $\mathcal{G}_B$ into a 3-tuple $(e, interact, e')$, where $interact$ stands for an additional relationship beyond system calls. As both $\mathcal{G}_P$ and $\mathcal{G}_B$ are now defined as entity-relation-entity sets, we can unify them as a KG, $\mathcal{G}_K = \{(h, r, t) | h, t \in \mathcal{E}, r \in \mathcal{R}'\}$, where $\mathcal{R}' = \mathcal{R} \cup \{interact\}$. With a KG representing both system entity contexts and interactions, we formally define *first-order connectivities* as one-hop connections (e.g., */etc/passwd*$\xrightarrow{r_1}$*gtcache*) in the KG and *high-order connectivities* as multi-hop paths (e.g., */etc/passwd*$\xrightarrow{r_1}$*gtcache*$\xrightarrow{r_4}$*146.153.68.151:80*).

B. Problem Statement

Given system entity interactions from audit records, we aim to learn a recommendation model whose objective is to predict the probability $\hat{y}_{ht}$ that a system entity h would not interact with another entity t. Note that $\hat{y}_{ht}$ also indicates the likelihood of an interaction to be adversarial, which forms the basis for ShadeWatcher to analyze cyber threats in audit records.

Threat Model: This work considers an attacker who aims to manipulate or exfiltrate information present in a system. For example, the attacker may install malware or insert a backdoor to steal sensitive data. Similar to previous studies on system auditing [11], [14], [17], [21], [39], we assume an OS and kernel-space auditing frameworks to be our trusted computing base. Additionally, we do not consider hardware trojans or side-channel attacks that are invisible in system auditing.

Note that during the APT lifecycle [13], attackers may escalate privileges to corrupt system auditing, at which point audit data are no longer reliable for cyber threat analysis.

However, we can ensure the integrity of historical audit records by employing secure provenance storage systems [8], [40] or managing audit data in a remote analysis server. As such, attackers have no way of manipulating previous audit records that have tracked the evidence of privilege escalation for SHADEWATCHER to detect malicious system entity interactions. By further integrating tamper-evident auditing techniques [41], [42], we can locate when attackers tamper with audit records to hide their footprints. Finally, system hardening techniques (e.g., Linux IMA) can be adopted to increase the complexity of compromising system auditing.

IV. SHADEWATCHER OVERVIEW

Figure 2 presents an overview of the SHADEWATCHER architecture, which receives audit records collected by the commodity auditing framework (i.e., Linux Audit [43]) and produces signals for adversarial system entity interactions. SHADEWATCHER consists of four major phases: building a knowledge graph (KG), generating a recommendation model, detecting cyber threats, and adapting the model.

Our KG builder first converts system audit records into a provenance graph (PG) and extracts system entity interactions as a bipartite graph (BG). Then, the PG and BG are combined into a KG, which is subsequently used to learn a recommendation model whose objective is to predict the preferences of system entities on their interactive entities.

The key idea behind our recommendation is to *use different-order connectivities in a KG to model the likelihood of system entity interactions, identifying anomalies in system execution as cyber threats*. Full details will be given in § VI, but we outline the workflow here. To explicitly exploit first- and high-order information, we parameterize system entities as embeddings (i.e., vectors) via *a context-aware embedding module* and then iteratively propagate embeddings from neighbors via *a graph neural network*. By aggregating embeddings from all propagation iterations, SHADEWATCHER determines the probability of entity-entity interactions to be adversarial.

When system behavior changes, SHADEWATCHER may raise false alarms for unobserved yet benign system entity interactions. To keep up with evolving interactions patterns, SHADEWATCHER provides an option to dynamically update its recommendation model by adapting to analyst feedback on false-positive interactions.

In summary, SHADEWATCHER's functionalities can be separated into two stages, i.e., training and detection. To perform anomaly-based detection, we use attack-free audit records to train the recommendation model. For the newly arrived audit stream, SHADEWATCHER first extracts system entity interactions and feeds them to the recommendation model obtained from the training stage. Then, SHADEWATCHER detects interactions as potential threats if their probabilities of being adversarial are larger than a pre-defined threshold. Note that SHADEWATCHER is currently designed and implemented to perform offline cyber threat analysis. We discuss the potential of adapting SHADEWATCHER to an online approach and the corresponding challenge in Appendix A.

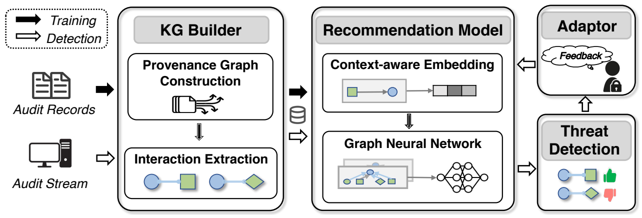


Fig. 2: Overview of SHADEWATCHER architecture.

V. KNOWLEDGE GRAPH BUILDER

In this section, we present how to parse audit records into a knowledge graph (KG), which preserves both first- and high-order information in audit records.

A. Provenance Graph Construction

Given audit records on end hosts, SHADEWATCHER transforms them into a graph structure called a provenance graph (PG). Nodes in the graph denote system entities with a set of attributes[1], and edges describe causal dependencies among system entities and the timestamp of record occurrence. As a common representation of audit records [8], [16], [44], [45], the PG facilitates reasoning over long-lived attacks as causal records are closely correlated, albeit temporally distant.

Notice that most audit records are not strictly necessary in the causal analysis of cyber threats [46]. Even worse, adversarial activities may be crowded out in the noise of normal and complicated audit records. Accordingly, we implement several noise reduction strategies from recent work [6], [46], [47] (explained in Appendix B) to reduce auditing complexity while preserving attack-relevant information.

B. Interaction Extraction

SHADEWATCHER identifies cyber threats on the basis of system entity interactions. A naïve approach to extracting interactions is to pair every two system entities in a PG and explore their causal dependency. In particular, a pair of causally connected entities represents a valid interaction. Unfortunately, traversing all system entity pairs in a PG is infeasible in practice due to the large size that most PGs have. Case in point, the PG built upon the DARPA TRACE dataset [48] consists of over six million system entities, forming 18 trillion pairs. However, only a tiny portion (much less than 0.01%) of system entity pairs exhibit valid interactions.

Abstracting behaviors from audit records has been shown effective in reducing analysis workloads [31]. Specifically, each behavior is represented as a provenance subgraph with a sequence of causal records rooted at a data object (e.g., file). Figure 1a illustrates an example of the behavior associated with Extension Backdoor. The *gtcache* highlighted with double circles denotes the root data object. Working on the level of behaviors can substantially reduce the scope of interaction analysis because causally disconnected system entities have been separated into different behaviors. As such, we decide to partition a PG into multiple subgraphs, each describing a

[1]Process attributes: *pid*, *ppid*, *exe*, and *args*; file attributes: *name*, *path*, and *version*; socket attributes: *ip* and *port*.

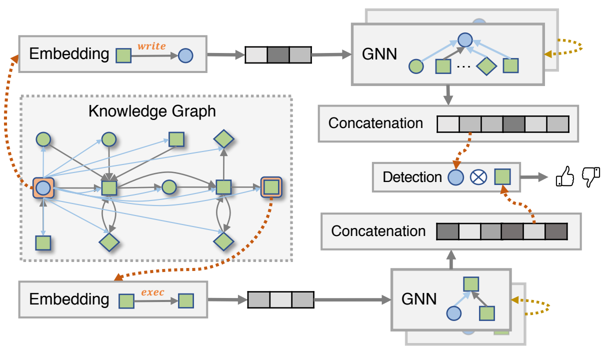


Fig. 3: An illustration of SHADEWATCHER's recommendation.

behavior, before extracting system entity interactions. To do so, we first identify all data objects in a PG, then perform a forward depth-first search on individual data objects to extract subgraphs (see [31] for more details), and finally merge two subgraphs if one subgraph is a subset of the other.

Intuitively, a behavior summarizes a series of interactions between a data object and its interactive entities, which separately indicate the initiator and targets of interactions. For example, given the interactions *gtcache*→*uname* and *gtcache*→*162.66.239.75:53*, we observe that an executable attempts to collect system configurations and scan network services. Based on this intuition, SHADEWATCHER converts interactions in behaviors into a bipartite graph (BG), where two disjoint node sets are data objects and system entities, and edges connecting two sets reflect interactions.

Combining Provenance Graph and Bipartite Graph. Considering system entity interaction as a relation beyond system calls, both PG and BG are formulated as sets of entity-relation-entity tuples. We thus align system entities to merge them into a KG as defined in § III-A. SHADEWATCHER provides the capability to store a KG into databases (PostgreSQL [49] or Elasticsearch [50]) so that the KG can be queried without being built from scratch for downstream cyber threat analysis. We also integrate Kibana [51] into SHADEWATCHER as the visualization front-end to facilitate attack investigation[2].

VI. RECOMMENDATION MODEL

Figure 3 illustrates the workflow of SHADEWATCHER's recommendation model. It mainly consists of three components: 1) modeling the first-order information, which parameterizes system entities as embeddings (i.e., vectorized representations) through their usage contexts; 2) modeling the higher-order information, which updates system entity representations by recursively propagating information from multi-hop neighboring entities; 3) learning to detect threats, which predicts an interaction's probability of being adversarial on the top of two system entity representations.

A. Modeling the First-order Information

Having coupled security concepts of system entity interactions and entity contexts with recommendation concepts of user-item interactions and item properties, we are aware that direct connections in a knowledge graph (KG) present both behavioral and semantic similarities among system entities. To model such first-order connectivity, we use KG embedding methods to parameterize system entities as vectors, where the distance between two vectorized entities captures their similarity. TransE [52] is a widely-used method. At its core is the translation principle: if a tuple (h, r, t) holds in a KG, the embeddings of system entities h, t and their relation r are learned by satisfying $\mathbf{e}_h + \mathbf{e}_r \approx \mathbf{e}_t$, where $\mathbf{e}_h$, $\mathbf{e}_r$, $\mathbf{e}_t \in \mathbb{R}^d$. This principle perfectly matches our intuitive understanding of system entities. Take (*ztmp*, *connect*, *128.55.12.73:53*) and (*ztmp*, *connect*, *128.55.12.73:54*) in Figure 1a as examples. As both network sockets are represented as $\mathbf{e}_{ztmp} + \mathbf{e}_{connect}$, they are embedded nearby in the vector space, indicating similar semantics, which mirrors our domain knowledge of labeling both of them as network scanning targets. Despite its effectiveness, a significant limitation exists in TransE — a single relation type may correspond to multiple entities, causing 1-to-N, N-to-1, or N-to-N problems [53]. We demonstrate how this limitation affects cyber threat analysis in § VIII-C.

To overcome this issue, we employ TransR [33], which learns a separate representation for a system entity conditioned on different relations. It allows us to assign distinctive semantics to the same entity under different relation contexts. More formally, TransR maps system entities h, t and relation r into embeddings $\mathbf{e}_h, \mathbf{e}_t \in \mathbb{R}^d, \mathbf{e}_r \in \mathbb{R}^k$. For each relation r, it specifies a projection matrix $\mathbf{W}_r \in \mathbb{R}^{d\times k}$ to transform system entities from a d-dimensional entity space to a k-dimensional relation space, i.e., $\mathbf{e}_h^r = \mathbf{e}_h \mathbf{W}_r$, $\mathbf{e}_t^r = \mathbf{e}_t \mathbf{W}_r$. Afterward, TransR measures the plausibility score of a given tuple (h, r, t) as follows:

$$f(h, r, t) = \|\mathbf{e}_h^r + \mathbf{e}_r - \mathbf{e}_t^r\|,$$

where $\|\cdot\|$ denotes the L1-norm distance function. A lower score of $f(h, r, t)$ suggests that the tuple is more likely to be observed in a KG and otherwise not.

To optimize the representation learning of TransR, we resort to a margin-based pairwise ranking loss, which enforces the plausibility score of a valid (observed in a KG) tuple to be lower than that of a corrupted (unobserved) tuple:

$$\mathcal{L}_{first} = \sum_{(h,r,t)\in\mathcal{G}_K} \sum_{(h',r',t')\notin\mathcal{G}_K} \sigma(f(h, r, t) - f(h', r', t') + \gamma),$$

where (h, r, t) holds in a KG, while (h', r', t') does not exist in the KG; γ is the hyper-parameter that controls the margin between valid and corrupted tuples; $\sigma(x)$ is the softplus activation function. Following mainstream recommendation systems [54], [55], we generate corrupted tuples by replacing one system entity in a valid tuple with a random entity. In summary, minimizing this loss of the first-order modeling allows us to encode the semantic and behavioral similarities into system entity representations.

B. Modeling the Higher-order Information

Beyond direct (first-hop) connections, multi-hop paths are inherent in a KG. Such higher-order connectivities not only

[2]A demonstration can be found at https://youtu.be/Kd8t0YnPAvY.

supplement similarities among system entities but also exhibit how system entities influence each other. For example, a two-hop path */proc/25/stat*$\xrightarrow{r_0}$*gtcache*$\xrightarrow{r_0}$*/proc/27/stat* shows the similarity between */proc/25(27)/stat* as they both interact with *gtcache*; the */etc/passwd*$\xrightarrow{r_1}$*gtcache*$\xrightarrow{r_4}$*146.153.68.151:80* describes how sensitive user information is transmitted out of an enterprise. Clearly, modeling higher-order connectivity can help us localize potential adversaries by revealing system entity relationships. However, solely using TransR is unable to characterize such high-order information.

To capture high-order connectivity, we adopt graph neural network (GNN) [34] to integrate multi-hop paths into system entity representations. Specifically, given a system entity h, one GNN module recursively updates its representation by propagating and aggregating messages from neighbors:

$$\mathbf{z}_h^{(l)} = g(\mathbf{z}_h^{(l-1)}, \mathbf{z}_{\mathcal{N}_h}^{(l-1)}),$$

where $\mathbf{z}_h^{(l)} \in \mathbb{R}^{d_l}$ is the d_l-dimensional representation of h at the l-th propagation layer, $\mathbf{z}_h^{(l-1)} \in \mathbb{R}^{d_{l-1}}$ is that of previous layer, and $\mathbf{z}_h^{(0)} = \mathbf{e}_h^r$ is initialized by embeddings derived from TransR; $\mathcal{N}_h$ is h' one-hop neighbors (aka ego-network [56]), and $\mathbf{z}_{\mathcal{N}_h}^{(l-1)} \in \mathbb{R}^{d_{l-1}}$ memorizes the information being propagated from h's $(l-1)$-hop neighbors; $g(\cdot)$ is the aggregation function which combines the representation of an entity with the information propagated from its neighbors. As we can see, both information propagation and aggregation functions play essential roles in a GNN module.

In terms of information propagation, as different neighboring entities should contribute unequally to the ego entity h, we devise an attention mechanism [29] to discriminate the importance of system entity neighbors:

$$\mathbf{z}_{\mathcal{N}_h}^{(l-1)} = \sum_{(h,r,t)\in\mathcal{N}_h} \alpha(h,r,t)\mathbf{z}_t^{(l-1)},$$

where $\alpha(h,r,t)$ is the attention function to control how much information is propagated from t to h conditioned on a certain relation r. We design it as follows:

$$\alpha(h,r,t) = \mathbf{e}_t^{r\top}\tanh(\mathbf{e}_h^r + \mathbf{e}_r),$$

where $\mathbf{e}_t^r$, $\mathbf{e}_h^r$, and $\mathbf{e}_r$ are system entity embeddings obtained from TransR. The attention scores across all neighboring entities are further normalized by the softmax function. Through this attentive information propagation, we are able to highlight informative signals from relevant entities and filter out uninformative signals from irrelevant entities.

In terms of information aggregation, we adopt the GraphSage Aggregator [57] to update system entity representations:

$$g(\mathbf{z}_h^{(l-1)}, \mathbf{z}_{\mathcal{N}_h}^{(l-1)}) = \mathrm{LeakyReLU}((\mathbf{z}_h^{(l-1)}||\mathbf{z}_{\mathcal{N}_h}^{(l-1)})\mathbf{W}^{(l)}),$$

where $\cdot||\cdot$ is the concatenation operator between two vectors; $\mathbf{W}^{(l)} \in \mathbb{R}^{2d^{(l-1)}\times d^{(l)}}$ is a transformation matrix at the l-th propagation layer to distill useful information. As such, we can integrate the messages of multi-hop neighbors into an entity's original representation $\mathbf{z}_h^{(0)}$ to form a new representation $\mathbf{z}_h^{(l)}$. Specifically, the number of hops in integrating neighboring entities is determined by the number of propagation layers L.

C. Learning to Detect Threats

Having obtained the representations of system entities, we move on to threat detection — learning to classify system entity interactions into normal and adversarial.

After L iterations of information propagation and aggregation, we obtain a series of representations for entity h, $\{\mathbf{z}_h^{(0)}, \cdots, \mathbf{z}_h^{(L)}\}$, which encode different-order information in a KG. Here we employ a simple concatenation operator to merge them into the final representation:

$$\mathbf{z}_h^* = \mathbf{z}^{(0)}||\cdots||\mathbf{z}_h^{(L)}.$$

The concatenation introduces no additional parameters to optimize and preserves information pertinent to different propagation layers, which has achieved promising performance in recent recommendation systems [57], [58].

Given any interaction $(h, interact, t)$, we apply the inner product on system entity representations to predict how likely system entity h would **not** interact with another entity t:

$$\hat{y}_{ht} = \mathbf{z}_h^{*\top}\mathbf{z}_t^*.$$

If the probability $\hat{y}_{ht}$ is larger than a pre-defined threshold, we further flag the interaction as a potential cyber threat. To meet this principle, we learn parameters in the GNN module by optimizing a widely-used pairwise loss [59]:

$$\mathcal{L}_{higher} = \sum_{(h,r_0,t)\in\mathcal{G}_K} \sum_{(h',r_0,t')\notin\mathcal{G}_K} \sigma(\hat{y}_{ht} - \hat{y}_{h't'}),$$

where r_0 denotes the *interact* relation; the interactions observed in a KG built upon normal audit records are viewed as negative (benign) instances; meanwhile, we randomly sample interactions unobserved in the KG as positive (potentially malicious) instances. Note that our sampled interactions do not necessarily reflect cyber threats. We further explain their impacts on threat detection in Appendix C.

By combining the losses of the first-order modeling and the higher-order modeling, we minimize the following objective function to learn parameters in our recommendation model:

$$\mathcal{L} = \mathcal{L}_{first} + \mathcal{L}_{higher} + \lambda\left\|\Theta\right\|,$$

where $\Theta = \{\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t, \mathbf{W}_r, \mathbf{W}^{(l)}|h,t \in \mathcal{E}, r \in \mathcal{R}', l \in \{1,\cdots,L\}\}$ is the set of trainable model parameters; λ is the hyper-parameter that controls the L_2 regularization term to address the over-fitting problem [60].

D. Model Adaption

As system behavior changes, SHADEWATCHER may raise false alarms for benign system entity interactions unobserved at the training stage. Consequently, it is necessary to keep up with the evolution of interactions. In practice, analysts in security operations centers would continuously sift through threat alarms to filter out false positives. Therefore, a natural way to generalize SHADEWATCHER to evolving interactions is to include analysts in the loop for dynamic updates.

Towards this end, we provide an option for analysts to give new labels on false-positive interactions, allowing SHADEWATCHER to use false alarms as additional supervision to

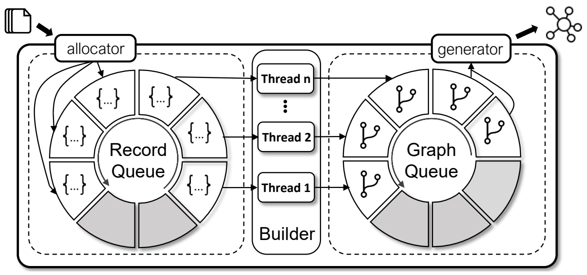


Fig. 4: Parallel Provenance Graph Construction.

revise its recommendation model. For example, suppose (*gt-cache*, *interact*, */proc/27/stat*) has been detected as malicious but later manually verified as a false alarm. To avoid future mistakes for similar interactions, SHADEWATCHER feeds this interaction as a new negative instance to retrain its model.

More specifically, to verify the nature of an alarm, analysts need to reconstruct the attack scenario by tracking the provenance between two system entities in the potentially malicious interaction. The main challenge faced by analysts is to understand previously unseen interactions, e.g., the first time a program loads a configuration file. To facilitate interpretation of such interactions, an intuitive approach is to incorporate additional auxiliary information — e.g., binary analysis, program comprehension, and network monitoring — into the KG so that analysts can reason about new interactions from different aspects, which we leave for future work.

We acknowledge that low-quality (e.g., false) feedback may mislead the recommendation model. However, as SHADEWATCHER provides fine-grained detection signals that highlight key indicators of an attack, analysts have a high chance to correctly differentiate true and false alarms.

VII. IMPLEMENTATION

We develop SHADEWATCHER in 11K lines of C++ code and 3K lines of Python code. We present important technical details in the implementation[3].

System Auditing Collection: To collect whole-system audit records, we make use of the Linux Audit with a ruleset covering 32 types of commonly-used system calls (see Appendix D for details). Once an audit record is generated, it is processed into a JSON format and shipped to SHADEWATCHER through Apache Kafka [61] in a stream fashion.

Parallel Provenance Graph Construction: Our initial prototype shows that the provenance graph (PG) construction is considerably time-consuming, which degrades the overall system performance. To put this into perspective, we present that our prototype spends approximately six hours on just PG construction for the DARPA TRACE dataset, while it only takes several seconds to predict cyber threats.

To address this issue, we implement a parallel pipeline, as shown in Figure 4, to allow concurrent audit record processing. The allocator first loads local audit records in a streaming fashion and inserts them into a record queue batch by batch. Once the builder identifies a new batch in the record queue, it distributes the batch to an idle thread under its control. Afterward, the thread produces a provenance subgraph into a graph queue. Meanwhile, subgraphs in the graph queue are continuously consumed by the generator to construct the final PG. In our current implementation, the parallel pipeline only works on audit records in the Common Data Model format [62] (e.g., DARPA TC dataset format) as it allows independent record processing. The massive dependencies among audit records generated by commodity auditing frameworks bring non-negligible efforts to support concurrent processing [10]. For example, the file descriptor used in a *read* record is always defined in *open* or *socket* records. We believe that the parallel or even distributed PG construction by itself is an interesting research topic, and we leave such an extension to future work.

Recommendation Model Training: We implement our recommendation model using Google Tensorflow [63]. The model is optimized by Adam optimizer [64], where the batch size, margin γ, and normalization λ are fixed at 1024, 1, and 10^{-5}. We train the model for 30 epochs with an early stop strategy — the training will be terminated if the accuracy does not increase on the validation set for five successive epochs. To mitigate the over-fitting problem, we further employ a dropout technique [65] with a dropping ratio of 0.2.

We initialize model parameters Θ with Xavier [66]. For hyper-parameters, we apply a grid search: the learning rate is tuned in $\{0.0001, 0.001, 0.01\}$; the embedding size of system entities is searched in $\{16, 32, 64\}$; the number of propagation layers in GNNs is tuned in $\{1, 2, 3\}$; and the threshold is searched in $\{-1, -0.5, 0, 0.5, 1\}$. In light of the best accuracy, we report results in a setting with the learning rate as 0.001, the embedding size as 32, two propagation layers with hidden dimensions as 32 and 16, and the threshold as -0.5.

VIII. EVALUATION

We evaluate SHADEWATCHER on four aspects: 1) How effective is SHADEWATCHER as a threat detection system? (§ VIII-B) 2) How do first-order and high-order information facilitate detection? (§ VIII-C) 3) What is the capability of the model adaption to reduce false alarms? (§ VIII-D) 4) How efficient is SHADEWATCHER? (§ VIII-E)

All experiments are performed on a server with Intel Xeon E5-2620 v4 CPUs @ 2.10GHz, 64 GB physical memory, and an NVIDIA Tesla V100 GPU. The OS is Ubuntu 16.04.3 LTS.

A. Dataset

In our evaluation, we use both a public DARPA TRACE dataset [48] (henceforth called TRACE dataset) for the reproduction of experimental results, as well as a simulated dataset to explore SHADEWATCHER's efficacy in practice. Table II summarizes the statistics of provenance graphs built upon our experimental datasets. Note that the dataset statistics are not necessarily the same as those of existing studies due to different noise reduction strategies and system entity granularities. For example, BEEP [67] partitions long-running processes into finer-grained execution units, causing a significant increase in the overall volume of system entities.

[3]SHADEWATCHER is avaialbe at https://github.com/jun-zeng/ShadeWatcher

TABLE I: Statistics of attack scenarios in the TRACE and Simulated datasets. All attack scenarios are abstracted as 10 of 153,859 behaviors in Table II for cyber threat analysis. The PG, BG, and KG share the same number of nodes. The detection results show the number of true positives (#TP), true negatives (#TN), false positives (#FP), and false negatives (#FN).

Dataset	Attack Scenario	#Node	#Edge			#Interaction		Detection Results			
			PG	BG	KG	Benign	Malicious	#TP	#TN	#FP	#FN
TRACE Dataset	Extension Backdoor	996	1,297	995	2,292	263	732	729	260	3	3
	Firefox Backdoor	252	263	251	514	243	8	7	231	12	1
	Pine Backdoor	67,352	67,396	67,351	134,747	10	67,341	67,338	10	0	3
	Phishing Executable	23	23	22	45	6	16	13	5	1	3
Simulated Dataset	Configuration Leakage	27	99	26	125	22	4	4	22	0	0
	Content Destruction	84	358	83	441	77	6	6	76	1	0
	Cheating Student	31	51	30	81	23	7	6	23	0	1
	Illegal Storage	270	1,284	269	1,553	259	10	10	255	4	0
	Passwd Gzip Scp	25	75	24	99	15	9	8	15	0	1
	Passwd Reuse	11	14	10	24	7	3	3	6	1	0

For each dataset, we randomly select 80%, 10%, and 10% of system entity interactions to constitute the training, validation, and testing sets. To avoid data snooping and biased parameter risks [68], we split interactions disjointly into these three sets and tune SHADEWATCHER's hyper-parameters based solely on the validation set. With our full knowledge of attack workflow, we manually label the ground truth of interactions through their relations to attacks. Detailed descriptions of attack scenarios are summarized in Appendix E.

TRACE Dataset. Our first dataset is a public APT attack dataset collected by the TRACE team in the DARPA TC program [69]. This dataset was generated during the third red-team vs. blue-team adversarial engagement in April 2018. The engagement simulates an enterprise environment for two weeks in an isolated network with multiple security-critical services such as an SSH server, email server, and SMB server. The red team carries out five APT campaigns to exfiltrate proprietary information. Four of them are visible in system audit records [17], including Extension Backdoor, Firefox Backdoor, Pine Backdoor, and Phishing Executable. To mix benign and malicious audit records, the red team also performs extensive common routines in parallel to attacks, such as web browsing, reading emails, and administrative tasks.

Simulated Dataset. For our second dataset, we simulate six cyber-attacks from previous work, including Configuration Leakage [31], Content Destruction [36], Cheating Student [70], Illegal Storage [36], Passwd Gzip Scp [6], and Passwd Reuse [31]. Following public attack documentation, we implement these attacks in a controlled testbed environment to collect the associated audit records. To incorporate the impacts of benign activities, we emulate diverse system behaviors (e.g., software installation using *apt*) in the best efforts during in-progress attacks.

TABLE II: Dataset statistics in provenance graphs.

Dataset	#Node	#Edge	#Behavior	#Interaction
TRACE Dataset	6,109,307	12,661,091	148,335	7,923,332
Simulated Dataset	367,406	3,022,193	5,524	2,857,717

B. Effectiveness

SHADEWATCHER predicts a probability that a system entity would **not** prefer its interactive entity, where the probability exceeding a pre-defined threshold indicates a malicious interaction (i.e., cyber threat). We evaluate SHADEWATCHER's detection effectiveness using precision, recall, F1-score, and accuracy metrics. Specifically, precision measures correctly detected threats against predicted threats; recall measures correctly detected threats against ground-truth threats; and F1-score calculates the harmonic mean of the precision and recall.

Evaluation on Attack Scenarios. We abstract ten cyber-attacks from the TRACE and Simulated datasets as individual behaviors with sets of benign and malicious system entity interactions. Their statistics and detection results are summarized in Table I and Table III. As observed, SHADEWATCHER exhibits satisfactory detection performance on both experimental datasets. Specifically, it only misses 12 of 68,136 malicious interactions. Upon closer investigation, we find that the majority of false-negative interactions come from web server communications. For example, two of three false negatives in Extension Backdoor (i.e., motivating example) are interactions between *gtcache* and {*146.153.68.151:80*, *162.66.239.75:80*}. SHADEWATCHER misclassifies these malicious interactions because *gtcache* represents an executable downloaded by browsers, and it frequently occurs for such executable to connect to a web server in the training set. To gain further insight, we study how SHADEWATCHER provides recommendations for Extension Backdoor in § VIII-F. From Table I, we also observe that SHADEWATCHER generates relatively more false positives for Firefox Backdoor compared with other attack scenarios. This case shows the importance of usage contexts for distinguishing different system entities and predicting their preferences. For the reason of space, we provide an in-depth analysis of Firefox Backdoor in Appendix F. Furthermore, as discussed in § III-B, audit records may not be reliable after an attacker performs privilege escalation. Therefore, we also evaluate the effectiveness of SHADEWATCHER's detection based only on audit data collected before privilege escalation in Appendix G.

TABLE III: Detection results of attack scenarios.

Dataset	Precision	Recall	F1-Score	Accuracy
TRACE Dataset	99.98%	99.99%	0.9998	0.9996
Simulated Dataset	86.05%	94.87%	0.9024	0.9819

Evaluation on Normal Workloads. To evaluate false alarms in normal workloads, we apply SHADEWATCHER to detect cyber threats based on benign system entity interactions in the testing sets. The results are summarized in Table IV. We see

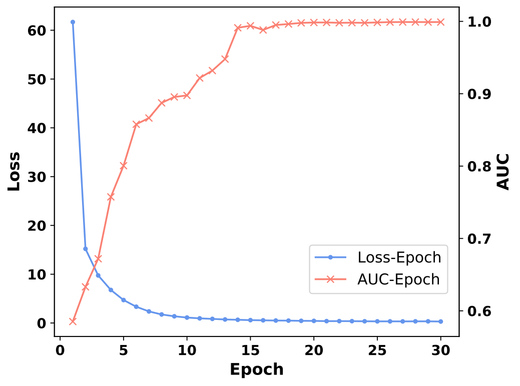


Fig. 5: Training loss and AUC value vs. the number of epochs.

that SHADEWATCHER achieves reasonably low false-positive rates (0.332% and 0.137%) in both datasets. This is because SHADEWATCHER interprets system entity interactions based on their semantics rather than historical frequencies. As such, rare or even unobserved interactions are not necessarily identified as anomalies. For example, although *gtcache* has not been witnessed to interact with */proc/27/stat*, SHADEWATCHER still recommends a low probability (-1.39) for the interaction to be adversarial, because */proc/27/stat* shares similar semantics with other */proc/pid/stat* previously accessed by *gtcache*.

Notice that although SHADEWATCHER achieves reasonable false-positive rates, the total number of false alarms can still be high for manual verification due to the overwhelming volume of audit records. We leave as future work to further assist analysts in attack investigation, e.g., by threat alert triage.

TABLE IV: Detection results of normal workloads.

Dataset	#Interaction (Benign)	Detection Results		
		#TN	#FP	FPR
TRACE Dataset	724,236	721,831	2,405	0.332%
Simulated Dataset	285,732	285,340	392	0.137%

Evaluation on Classification. Since our cyber threat detection is essentially a classification task, we further use Area Under the Curve (AUC) to analyze SHADEWATCHER's capability in distinguishing between benign and malicious system entity interactions. The higher the AUC value is, the better SHADEWATCHER is at interaction classification. More specifically, we study how AUC varies on the testing set while training our recommendation model. Figure 5 shows the training loss and AUC on the TRACE dataset for 30 training epochs. As can be observed, AUC increases to a high value after 14 epochs and remains stable above 0.988, while the training loss drops to a low value after 25 epochs and remains around 0.340. Accordingly, we conclude that SHADEWATCHER is promising at classifying system entity interactions.

C. Comparison Analysis

To answer how our proposed first-order and high-order modeling facilitates cyber threat detection, we have developed several baseline approaches in place of SHADEWATCHER's components to conduct an ablation study on the TRACE dataset. In particular, we compare TransR with different embedding algorithms, namely one-hot encoding [71], TransE [52] and TransH [53]. We also compare how SHADEWATCHER performs with and without high-order information from GNN.

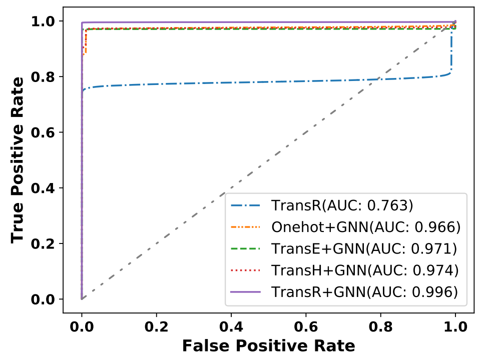


Fig. 6: ROC curves for ablation study.

Figure 6 plots the receiver operating characteristics (ROC) curves of threat detection results, demonstrating that SHADEWATCHER (TransR+GNN) outperforms all other approaches. This is expected as one-hot encoding completely ignores the contextual information of system entities. In addition, both TransE and TransH assume system entity embeddings to be the same under different relations contexts. However, entities typically have multiple aspects, and different relations should focus on different aspects. Following this principle, TransR learns a separate representation for every relation on which an entity is conditioned, so it does not conflate different aspects of the same entity. To understand the internals of TransR, we visualize TransR's embedding spaces in Appendix H. While TransR outperforms both TransE and TransH in our case, it tends to incur a higher runtime overhead for projecting system entities from an entity space to relation spaces. Fortunately, as the projection (i.e., matrix multiplication) can be largely parallelized by modern GPUs, it is no longer expensive to compute. Our experiment shows that TransR is only 1.4X/1.1X slower than TransE/TransH.

Another interesting observation is that without high-order information from GNN, SHADEWATCHER consistently performs the worst across all possible thresholds. However, by enabling GNN, SHADEWATCHER achieves high AUC values ranging from 0.966 to 0.996. One possible reason is that high-order connectivity in a KG is critical for interpreting system entity relationships, hence illustrating the significance of high-order information in audit records for cyber threat analysis.

SHADEWATCHER vs. State-of-the-Art. Elsewhere in the literature, the TRACE dataset has also been used to evaluate intrusion detection systems (i.e., Poirot [19] and Morse [17]), but SHADEWATCHER is fundamentally distinct from them in terms of detection goals and techniques. Especially, SHADEWATCHER performs anomaly-based, not specification-based, detection, requiring no *a priori* knowledge of attacks. However, it is worth mentioning that SHADEWATCHER achieves comparable detection effectiveness to specification-based detectors by successfully detecting all four APT attacks on the TRACE dataset. We further discuss cyber threats that different detection systems might miss. False negatives produced by specification-based detection are primarily attributed to zero-day attacks. For example, Poirot inevitably misses APT attacks not yet described in existing CTI reports. On the other hand, SHADEWATCHER experiences false negatives if a threat actor intentionally steers malicious activities towards the learned recommendation model to evade detection. We provide details of evasion attacks on SHADEWATCHER in § IX.

Unicorn [21] is another anomaly detector that identifies cyber threats without domain expertise. Unfortunately, the coarse granularity of its detection signals prevents responding to threats timely. To illustrate, suppose Unicorn configures the size of graph sketches as 2,000 in [21]. An analyst then has to review 2,000 audit records to investigate a single attack indicator. On the contrary, SHADEWATCHER explores the potential of performing finer-grained detection on system entity interactions, which directly signify adversarial intentions.

D. Model Adaptability

By adapting to analyst feedback on false alarms, SHADEWATCHER supports dynamic updates to its recommendation model. We evaluate the ability of SHADEWATCHER's adaption to reduce false positives on normal workloads in the Simulated dataset. For the case of model adaption, we assume that an analyst continuously reports false-positive interactions to update SHADEWATCHER. To compare false-positive rates (FPR) with and without model adaption, we evaluate SHADEWATCHER on three different training sets: (A) the first 80% interactions; (B) the first 80% interactions with false positives in the subsequent 10% interactions; (C) the first 90% interactions. For a fair comparison, we consistently extract the last 10% interactions as the testing set. The sequence of interactions is determined by their occurrence timestamps in audit records.

By training on the first 80% interactions to predict cyber threats in the subsequent 10% interactions, SHADEWATCHER reports 263 false positives and 285,508 true negatives. As such, compared to (A), both (B) and (C) can be viewed as integrating additional manual feedback on 263 false alarms. The only difference is that (C) includes extra 285,508 true-negative interactions. Observe that in Table V, SHADEWATCHER reduces 118 false positives by incorporating analyst feedback. Increasing the training data from 80% to 90% can further remove 37 false alarms. This makes sense as our model adaption only takes false-positive interactions as additional supervision, and thus it inevitably loses the semantics of true-negative interactions that may also be beneficial for interpreting unknown interactions.

TABLE V: False positive reduction with model adaption.

Training	Feedback	#FP	#TN	FPR
80%	0	392	285,340	0.137%
80%	263	274	285,458	0.096%
90%	263	237	285,495	0.083%

E. Efficiency

We measure the runtime overhead of SHADEWATCHER at different phases, including data processing, training, and testing phases. All experiments are performed five times on the TRACE dataset, and we report the mean results in Table VI.

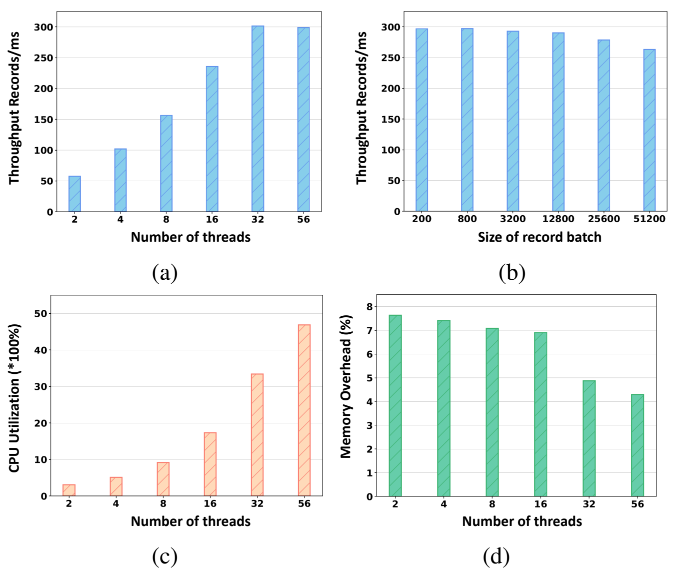


Fig. 7: System performance for provenance graph construction: (a) (c) (d) show the throughput, CPU utilization and memory overhead under different numbers of threads. (b) shows the throughput under 56 threads with different sizes of audit record batches. Note that our experimental machine supports at most 56 threads running in parallel.

Data processing. Data processing aims to parse audit records into a knowledge graph (KG). The overhead mainly comes from provenance graph (PG) construction, noise reduction, and interaction extraction. In total, it takes 100.37 minutes to process the TRACE dataset with 635GB worth of audit data.

To improve system efficiency, we leverage multiple threads to construct a PG in parallel. Compared with the single-thread (naïve) prototype, the multi-thread design accomplishes an 8X speedup. That is, it saves over five hours on just data processing. To further explore its scalability, we measure the throughput, CPU utilization, and memory overhead with different configuration values of thread numbers and record batch sizes. Figure 7a shows that SHADEWATCHER processes up to 300 audit records per millisecond with 56 threads. Especially if an end host generates 400k records per day [14], we estimate that SHADEWATCHER is capable of scaling the PG construction to upwards of 65,000 hosts. In general, SHADEWATCHER's throughput increases along with the thread number. However, it will reach an upper bound due to the limitation of disk I/O performance. Observe that in Figure 7b, it is flexible in choosing sizes of audit record batches while not heavily affecting system efficiency. Figure 7c and Figure 7d further illustrate that CPU utilization increases along with increasing thread number, while memory overhead the opposite. This is within our expectation as most memory overhead comes from record batches stored in the record queue, and more threads can consume batches faster, leading to fewer batches maintained in the memory.

Training. We evaluate the training overhead by measuring how long SHADEWATCHER takes to obtain a recommendation model that yields the best accuracy on the validation set. Because most training computations (i.e., matrix operations) can be accelerated by GPU parallel computation, we apply a high-performance GPU to train our model. On average, it takes 2,103 seconds and 1,106 seconds per epoch to train the TransR and GNN modules, respectively. The fully-trained model is then saved to disk so that SHADEWATCHER does not need to be retrained from scratch with new incoming training samples. Note that we train the model on a single GPU. The training time can be further reduced by parallelizing SHADEWATCHER on multiple GPUs [72].

Testing. The testing phase refers to the duration from inputting system entity interactions to predicting cyber threats via the recommendation model from the training phase. On average, it takes 8.16 seconds to predict 792,333 interactions (68,097 malicious and 724,236 benign interactions). So far, we have conducted all experiments on GPUs, but GPUs may not be available in most real-life threat detection scenarios. Therefore, we further evaluate SHADEWATCHER's efficiency on CPUs by performing detection on the same server with GPU disabled. Although the testing time increases to 220 seconds, we believe the efficiency is still promising as the scale of the dataset is comparable to two months of user daily data.

TABLE VI: Runtime overhead on the TRACE dataset.

Phase	Component	Mean	Std Dev
Data Processing	PG Construction (Naïve)	5.97 Hours	N/A
	PG Construction (Parallel)	40.47 Min.	0.21
	Noise Reduction	55.77 Min.	0.09
	Interaction Extraction	4.13 Min.	0.01
Training	System Entity Embedding	12.27 Hours	N/A
	Graph Neural Network	6.45 Hours	
Testing	Cyber Threat Detection	8.16 Sec.	0.93

F. Case Study

We demonstrate how SHADEWATCHER uses recommendations to guide cyber threat analysis by two APT attacks: Extension Backdoor and Firefox Backdoor. Extension Backdoor is the attack from which the motivating example is derived. We leave the description of Firefox Backdoor in Appendix F.

Extension Backdoor generates a total of 995 system entity interactions, 732 of which are manually labeled as threats. We would like to point out that the interaction between *gtcache* and */etc/passwd* is treated as a benign activity. This is because */etc/passwd* serves as public information on a system, which is regularly read by a variety of daily programs, such as *bash*, *ssh*, and even *cat*. Accordingly, access to */etc/passwd* locally does not necessarily bring risks.

As shown in Figure 8, SHADEWATCHER predicts the probabilities of being adversarial for individual system entity interactions. The solid (green) edges denote ground-truth benign interactions, while the dashed (red) edges represent malicious interactions. We notice that of 263 benign interactions, most (230) come from process status retrievals (i.e., interactions with */proc/pid/stat*), while of 732 malicious interactions, most (721) come from port scans for internal reconnaissance (i.e., interactions with *128.55.12.73*). Experimental results show that SHADEWATCHER accurately identifies all benign activities except three false positives (e.g., interactions with */proc/24(29,1896)/stat*) and warns of all malicious activities except three false negatives (i.e., interactions with */tmp/ztmp*, *146.153.68.151:80*, and *162.66.239.75:80*). More importantly, benign and malicious interactions are well-separated with considerable margins. Case in point, all benign interactions are predicted with low probabilities (below -1.0) of being cyber threats, while the malicious ones are given high probabilities (beyond 0.4). Similar phenomena can also be found in other attack scenarios. Therefore, we hypothesize that SHADEWATCHER is not sensitive to thresholds. This also explains the nearly perfect AUC in § VIII-C.

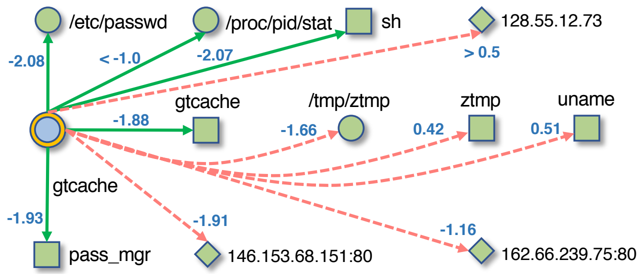


Fig. 8: Recommendations on the Extension Backdoor.

Although it is reasonable to access */etc/passwd* locally, we claim that */etc/passwd* is not supposed to be sent out to public networks since it includes sensitive user information. Any interactions between */etc/passwd* and public networks should be recommended as potential data exfiltration and reported to analysts for inspection. To validate this claim, we conduct an additional experiment to predict the probability of the interactions between */etc/passwd* and network sockets to be adversarial. As expected, both the interactions between */etc/passwd* and $\{$*146.153.68.151:80*, *162.66.239.75:80*$\}$ are given high probabilities (0.68 and 1.71), which are well beyond our threshold (-0.5) indicating attack activities.

IX. DISCUSSION & LIMITATION

Benign Dataset. SHADEWATCHER, like most anomaly detectors [21], [26], [73]–[75], requires attack-free data to profile benign behaviors. However, it does not guarantee that real-world audit records are perfectly clean. Accordingly, it is necessary to explore SHADEWATCHER's robustness against potential data contamination. To this end, we provide empirical results to demonstrate what happens if we purposefully poison our training set by treating malicious system entity interactions for detection in the TRACE dataset as benign training data. Results show that false negatives increase from 10 to 18 compared with training on benign audit data. As most malicious interactions (68,079) are still detected, we hypothesize that SHADEWATCHER is generally robust to data contaminations.

Evasion Attack. Evading SHADEWATCHER requires non-trivial efforts even for an attacker who is aware of its detection logic. Specifically, SHADEWATCHER differentiates between benign and malicious system entity interactions by modeling not only first-order (causal) but also high-order (semantic) connectivities. Therefore, directly incorporating noisy interactions irrelevant to attacks does not serve the purpose of evasion. Instead, the attacker must carefully match the semantics of malicious interactions to those of benign interactions. One possible strategy is to manipulate the structures of our knowledge graph (KG) to perform adversarial attacks on GNNs. While such attacks have shown potential on network graphs (e.g., citation network) [76], [77], we argue that they are less effective on the KG as system auditing provides a significantly more constrained setting for KG perturbations [22]. Especially, attackers cannot add an edge between two files or arbitrarily delete system entity interactions (e.g., */etc/passwd* interacting with *146.153.68.151:80* to exfiltrate data) without affecting attacks. Interesting future work would be designing specialized adversarial attacks for system auditing analysis.

False Alarm. A well-known limitation of anomaly-based detection is generating high false-positive rates (FPRs). SHADEWATCHER inherits this by detecting *anomalous* system entity interactions as cyber threats. However, as we demonstrate in Table IV, SHADEWATCHER achieves acceptable FPRs (below 0.5%) considering that it produces much finer-grained detection signals than existing anomaly detectors. Although, at first glance, Unicorn [21] generates a lower FPR on the DARPA TC dataset, it analyzes cyber threats at the granularity of behaviors that are very large and difficult to interpret in practice. Counterintuitively, the high FPR is also a major concern for specification-based detectors [78], although such detection, in theory, would raise fewer false positives by rigid attack matching. For example, RapSheet [14] reports a 2.2% FPR for identifying APT kill chains in provenance graphs.

One approach to mitigating false positives is adopting threat alarm triage techniques [79], the goal of which is to rank true alarms higher than false alarms before manual investigation. For example, we can integrate SHADEWATCHER with NoDoze [11] to assign a threat score for individual malicious system entity interactions so that analysts can focus on the most anomalous ones to accelerate incident response.

X. RELATED WORK

System Auditing. System auditing has recently attracted increasing attention due to its deep visibility into advanced cyber-attacks [80]–[82]. Extensive literature exists on collecting and storing whole-system audit data effectively and reliably [7], [8], [44], [83]–[85]. Although auditing is widely supported in modern SIEM [3]–[5], audit record analysis is still limited to two fundamental challenges: 1) as audit records capture low-level system calls, the volume of audit data is typically too overwhelming for analysts to investigate. Recent work aims to decrease the overall number of audit records by data reduction [6], [46] and graph compression [47], [86], [87]; 2) as each audit record is conservatively considered dependent on all the preceding records, system causality analysis may suffer from the dependency explosion problem (especially in the case of analyzing long-running processes). To address dependency explosion, researchers have explored different techniques to provide precise system provenance, which include execution unit partition [45], [67], [88], [89], dynamic tainting [90], [91], modeling-based inference [92], [93], record-and-replay systems [9], [94], and application log fusing [39], [95], [96]. Although the scope of SHADEWATCHER is different from these solutions, both noisy record reduction and accurate causality tracking help improve the quality of system entity interactions for cyber threat analysis.

Forensic investigation and intrusion detection are the ultimate objectives of system auditing. Recent studies have attempted to facilitate attack forensics by prioritizing causality tracking [10], [24], improving threat queries [97]–[99], and abstracting high-level behaviors [31]. Moreover, current attack detection approaches can be roughly categorized into three directions: statistical analysis [11], [21], specification matching [16], [17], [19] and learning-driven prediction [22], [73], which we elaborate on in § II-B.

Knowledge Graph-based Recommendation. Recommendations play an important role in a wide range of user-oriented applications to capture user preference on items. As a prevalent solution, collaborative filtering extracts behavioral patterns of users from historical user-item interactions to infer user preference [28], [100]. Going beyond the interaction data, researchers introduce a knowledge graph (KG), which offers rich content relatedness among items and supplements behavioral similarity among users to enhance the recommendation performance. Current work on KG-based recommendation falls into three research lines: 1) embedding-based methods [55], [101], which focus on first-order connectivity in a KG and derive knowledge-aware embeddings [33], [52] as the contents of items to enrich item representations; 2) path-based methods, which consider higher-order connectivity in the KG and extract multi-hop paths connecting the target user and items by brute-force searching [102], meta-path matching [103], or reinforcement learning [104]; 3) GNN-based methods [29], [30], which employ graph neural network (GNN)'s information propagation and aggregation mechanisms to encode higher-order connectivity into the representations of users and items. By identifying different-order connectivities in audit records, SHADEWATCHER provides the first recommendation system that detects cyber threats via predicting the preferences of system entities on interactions.

XI. CONCLUSION

In this paper, we present a recommendation-guided cyber threat detection system, SHADEWATCHER. It takes the first step to bring the benefits of recommendations to audit record analysis. By exploiting first-order and high-order information in audit records, SHADEWATCHER models preferences of a system entity on its interactive entities and recommends adversarial interactions. We evaluate SHADEWATCHER against two datasets of real-life APT campaigns and simulated cyber-attacks. Results show that SHADEWATCHER detects threats with high accuracy and 0.332% and 0.137% false alarm rates.

ACKNOWLEDGEMENT

We thank the anonymous reviewers for their insightful comments. We also thank Kaihang Ji, Zijun Yin, Zhaoying Li, and Sihao Diao for feedback on early drafts of this paper. Some of the experiments were conducted using the infrastructure of National Cybersecurity R&D Laboratory, Singapore. This research is supported by the National Research Foundation, Singapore under its Industry Alignment Fund – Pre-positioning (IAF-PP) Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of National Research Foundation, Singapore.

REFERENCES

[1] "Equifax Information Leakage," https://en.wikipedia.org/wiki/Equifax.
[2] "SolarWinds: How Russian spies hacked the Justice, State, Treasury, Energy and Commerce Departments," https://www.cbsnews.com/news/solarwinds-hack-russia-cyberattack-60-minutes-2021-02-14/, 2021, online; Accessed 17 August 2021.
[3] "Datadog: Modern monitoring & security," https://www.datadoghq.com/, 2021, online; Accessed 5 May 2021.
[4] "Logrhythm," https://logrhythm.com, [n.d.], online; Accessed 9 March 2021.
[5] "Splunk," https://docs.splunk.com/Documentation/CIM/4.19.0/User/SplunkAuditLogs, 2020, online; Accessed 5 May 2021.
[6] Z. Xu, Z. Wu, Z. Li, K. Jee, J. Rhee, X. Xiao, F. Xu, H. Wang, and G. Jiang, "High fidelity data reduction for big data security dependency analyses," in *ACM CCS*, 2016.
[7] A. Gehani and D. Tariq, "Spade: support for provenance auditing in distributed environments," in *Proceedings of the 13th International Middleware Conference*, 2012.
[8] A. Bates, D. J. Tian, K. R. Butler, and T. Moyer, "Trustworthy whole-system provenance for the linux kernel," in *USENIX Security*, 2015.
[9] Y. Ji, S. Lee, E. Downing, W. Wang, M. Fazzini, T. Kim, A. Orso, and W. Lee, "Rain: Refinable attack investigation with on-demand inter-process information flow tracking," in *ACM CCS*, 2017.
[10] Y. Liu, M. Zhang, D. Li, K. Jee, Z. Li, Z. Wu, J. Rhee, and P. Mittal, "Towards a timely causality analysis for enterprise security." in *NDSS*, 2018.
[11] 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 *NDSS*, 2019.
[12] W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, D. Wang, Z. Chen, Z. Li, J. Rhee, J. Gui *et al.*, "This is why we can't cache nice things: Lightning-fast threat hunting using suspicion-based hierarchical storage," in *ACSAC*, 2020.
[13] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "Holmes: real-time apt detection through correlation of suspicious information flows," in *IEEE Security and Privacy*, 2019.
[14] W. U. Hassan, A. Bates, and D. Marino, "Tactical provenance analysis for endpoint detection and response systems," in *IEEE Security and Privacy*, 2020.
[15] X. Jiang, A. Walters, D. Xu, E. H. Spafford, F. Buchholz, and Y.-M. Wang, "Provenance-aware tracing ofworm break-in and contaminations: A process coloring approach," in *IEEE ICDCS*, 2006.
[16] M. N. Hossain, S. M. Milajerdi, J. Wang, B. Eshete, R. Gjomemo, R. Sekar, S. Stoller, and V. Venkatakrishnan, "Sleuth: Real-time attack scenario reconstruction from cots audit data," in *USENIX Security*, 2017.
[17] M. N. Hossain, S. Sheikhi, and R. Sekar, "Combating dependence explosion in forensic analysis using alternative tag propagation semantics," in *IEEE Security and Privacy*, 2020.
[18] B. Zong, X. Xiao, Z. Li, Z. Wu, Z. Qian, X. Yan, A. K. Singh, and G. Jiang, "Behavior query discovery in system-generated temporal graphs," in *VLDB*, 2015.
[19] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, "POIROT: Aligning attack behavior with kernel audit records for cyber threat hunting," in *ACM CCS*, 2019.
[20] S. Wang, Z. Chen, X. Yu, D. Li, J. Ni, L.-A. Tang, J. Gui, Z. Li, H. Chen, and S. Y. Philip, "Heterogeneous graph matching networks for unknown malware detection." in *IJCAI*, 2019.
[21] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "Unicorn: Runtime provenance-based detector for advanced persistent threats," in *NDSS*, 2020.
[22] 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 *USENIX Security*, 2021.
[23] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Chen, W. Cheng, C. Gunter *et al.*, "You are what you do: Hunting stealthy malware via data provenance analysis," in *NDSS*, 2020.
[24] 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 *USENIX Security*, 2021.
[25] S. Rule, "SIEM rules ignore bulk of MITRE ATT&CK framework," https://www.scmagazine.com/news/network-security/siem-rules-ignore-bulk-of-mitre-attck-framework-placing-risk-burden-on-users, 2021, online; Accessed 10 March 2021.
[26] 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 *IEEE Security and Privacy*, 2022.
[27] X. He and T.-S. Chua, "Neural factorization machines for sparse predictive analytics," in *ACM SIGIR*, 2017.
[28] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua, "Neural graph collaborative filtering," in *ACM SIGIR*, 2019.
[29] X. Wang, X. He, Y. Cao, M. Liu, and T.-S. Chua, "Kgat: Knowledge graph attention network for recommendation," in *ACM KDD*, 2019.
[30] X. Wang, T. Huang, D. Wang, Y. Yuan, Z. Liu, X. He, and T.-S. Chua, "Learning intents behind interactions with knowledge graph for recommendation," in *ACM WWW*, 2021.
[31] J. Zeng, Z. L. Chua, Y. Chen, K. Ji, Z. Liang, and J. Mao, "Watson: Abstracting behaviors from audit logs via aggregation of contextual semantics," in *NDSS*, 2021.
[32] "MITRE TA0010: Exfiltration," https://attack.mitre.org/tactics/TA0010, 2020, online; Accessed 15 March 2020.
[33] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity and relation embeddings for knowledge graph completion," in *AAAI*, 2015.
[34] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.
[35] "Transparent Computing Engagement 3 Data Release," https://github.com/darpa-i2o/Transparent-Computing/blob/master/README-E3.md, [n.d.], online; Accessed 10 March 2020.
[36] A. Goel, K. Po, K. Farhadi, Z. Li, and E. De Lara, "The taser intrusion recovery system," in *SOSP*, 2005.
[37] "MITRE T1204: User Execution," https://attack.mitre.org/techniques/T1204, 2020, online; Accessed 15 March 2020.
[38] J. Wu, X. Wang, F. Feng, X. He, L. Chen, J. Lian, and X. Xie, "Self-supervised graph learning for recommendation," in *ACM SIGIR*, 2021.
[39] W. U. Hassan, M. A. Noureddine, P. Datta, and A. Bates, "Omega-log: High-fidelity attack investigation via transparent multi-layer log analysis," in *NDSS*, 2020.
[40] H. Ragib, R. Sion, and M. Winslett, "The case of the fake picasso: Preventing history forgery with secure provenance," in *FAST*, no. 2009.
[41] R. Paccagnella, K. Liao, D. Tian, and A. Bates, "Logging to the danger zone: Race condition attacks and defenses on system audit frameworks," in *ACM CCS*, 2020.
[42] R. Paccagnella, P. Datta, W. U. Hassan, A. Bates, C. Fletcher, A. Miller, and D. Tian, "Custos: Practical tamper-evident auditing of operating systems using trusted execution," in *NDSS*, 2020.
[43] "Linux Kernel Audit Subsystem," https://github.com/linux-audit/audit-kernel, [n.d.], online; Accessed 10 March 2021.
[44] D. J. Pohly, S. McLaughlin, P. McDaniel, and K. Butler, "Hi-fi: collecting high-fidelity whole-system provenance," in *ACSAC*, 2012.
[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 *USENIX Security*, 2017.
[46] K. H. Lee, X. Zhang, and D. Xu, "Loggc: garbage collecting audit log," in *ACM CCS*, 2013.
[47] Y. Tang, D. Li, Z. Li, M. Zhang, K. Jee, X. Xiao, Z. Wu, J. Rhee, F. Xu, and Q. Li, "Nodemerge: template based efficient data reduction for big-data causality analysis," in *ACM CCS*, 2018.

[48] H. Irshad, G. Ciocarlie, A. Gehani, V. Yegneswaran, K. H. Lee, J. Patel, S. Jha, Y. Kwon, D. Xu, and X. Zhang, "Trace: Enterprise-wide provenance tracking for real-time apt detection," *IEEE Transactions on Information Forensics and Security*, 2021.
[49] "Postgresql Relational Database," https://www.postgresql.org, 2021, online; Accessed 25 March 2020.
[50] "Elasticsearch NoSQL Database," https://www.elastic.co/elasticsearch, 2021, online; Accessed 18 November 2021.
[51] "Your window into the Elastic Stack," https://www.elastic.co/kibana, 2021, online; Accessed 18 November 2021.
[52] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *NeurIPS*, 2013.
[53] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes," in *AAAI*, 2014.
[54] H. Wang, F. Zhang, X. Xie, and M. Guo, "Dkn: Deep knowledge-aware network for news recommendation," in *ACM WWW*, 2018.
[55] Y. Cao, X. Wang, X. He, Z. Hu, and T.-S. Chua, "Unifying knowledge graph learning and recommendation: Towards a better understanding of user preferences," in *ACM WWW*, 2019.
[56] J. Qiu, J. Tang, H. Ma, Y. Dong, K. Wang, and J. Tang, "Deepinf: Social influence prediction with deep learning," in *ACM KDD*, 2018.
[57] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017.
[58] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *ICML*, 2018.
[59] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," *UAI*, 2009.
[60] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King, "Recommender systems with social regularization," in *ACM WSDM*, 2011.
[61] "Apache Kafka," https://kafka.apache.org/, 2019, online; Accessed 21 July 2021.
[62] J. Khoury, T. Upthegrove, A. Caro, B. Benyo, and D. Kong, "An event-based data model for granular information flow tracking," in *USENIX TaPP*, 2020.
[63] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *OSDI*, 2016.
[64] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.
[65] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," in *The journal of machine learning research*, 2014.
[66] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010.
[67] K. H. Lee, X. Zhang, and D. Xu, "High accuracy attack provenance via binary-based execution partition." in *NDSS*, 2013.
[68] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, "Dos and don'ts of machine learning in computer security," in *USENIX Security*, 2022.
[69] "DARPA Transparent Computing," https://www.darpa.mil/program/transparent-computing, 2014, online; Accessed 12 March 2020.
[70] "CVE-2020-15778," https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-15778, 2020, online; Accessed 9 March 2020.
[71] P. Rodríguez, M. A. Bautista, J. Gonzalez, and S. Escalera, "Beyond one-hot encoding: Lower dimensional target embedding," in *Image and Vision Computing*, 2018.
[72] X. Yi, Z. Luo, C. Meng, M. Wang, G. Long, C. Wu, J. Yang, and W. Lin, "Fast training of deep learning models over multiple gpus," in *Proceedings of the 21st International Middleware Conference*, 2020.
[73] T. Chen, L.-A. Tang, Y. Sun, Z. Chen, and K. Zhang, "Entity embedding-based anomaly detection for heterogeneous categorical events," in *IJCAI*, 2016.
[74] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *ACM CCS*, 2017.
[75] M. Du, Z. Chen, C. Liu, R. Oak, and D. Song, "Lifelong anomaly detection through unlearning," in *ACM CCS*, 2019.
[76] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *KDD*, 2018.
[77] B. Wang and N. Z. Gong, "Attacking graph-based classification via manipulating the graph structure," in *CCS*, 2019.
[78] ""How Many Alerts is Too Many to Handle?" https://www.fireeye.com/offers/rpt-idc-the-numbers-game.html, 2014, online; Accessed 4 April 2021.
[79] C. Zhong, J. Yen, P. Liu, and R. F. Erbacher, "Automate cybersecurity data triage by leveraging human analysts' cognitive process," in *BigDataSecurity*. IEEE, 2016.
[80] S. T. King and P. M. Chen, "Backtracking intrusions," in *SOSP*, 2003.
[81] S. T. King, Z. M. Mao, D. G. Lucchetti, and P. M. Chen, "Enriching intrusion alerts through multi-host causality." in *NDSS*, 2005.
[82] F. Wang, Y. Kwon, S. Ma, X. Zhang, and D. Xu, "Lprov: Practical library-aware provenance tracing," in *ACSAC*, 2018.
[83] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. I. Seltzer, "Provenance-aware storage systems." in *USENIX ATC*, 2006.
[84] T. F.-M. Pasquier, J. Singh, D. Eyers, and J. Bacon, "Camflow: Managed data-sharing for cloud services," in *IEEE Transactions on Cloud Computing*, 2015.
[85] T. Pasquier, X. Han, T. Moyer, A. Bates, O. Hermant, D. Eyers, J. Bacon, and M. Seltzer, "Runtime analysis of whole-system provenance," in *ACM CCS*, 2018.
[86] W. U. Hassan, L. Aguse, N. Aguse, A. Bates, and T. Moyer, "Towards scalable cluster auditing through grammatical inference over provenance graphs," in *NDSS*, 2018.
[87] M. N. Hossain, J. Wang, O. Weisse, R. Sekar, D. Genkin, B. He, S. D. Stoller, G. Fang, F. Piessens, E. Downing *et al.*, "Dependence-preserving data compaction for scalable forensic analysis," in *USENIX Security*, 2018.
[88] S. Ma, K. H. Lee, C. H. Kim, J. Rhee, X. Zhang, and D. Xu, "Accurate, low cost and instrumentation-free security audit logging for windows," in *ACSAC*, 2015.
[89] S. Ma, X. Zhang, and D. Xu, "Protracer: Towards practical provenance tracing by alternating between logging and tainting." in *NDSS*, 2016.
[90] J. Newsome and D. X. Song, "Dynamic taint analysis for automatic detection, analysis, and signaturegeneration of exploits on commodity software." in *NDSS*, 2005.
[91] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda, "Panorama: capturing system-wide information flow for malware detection and analysis," in *ACM CCS*, 2007.
[92] Y. Kwon, D. Kim, W. N. Sumner, K. Kim, B. Saltaformaggio, X. Zhang, and D. Xu, "Ldx: Causality inference by lightweight dual execution," in *ASPLOS*, 2016.
[93] Y. Kwon, F. Wang, W. Wang, K. H. Lee, W.-C. Lee, S. Ma, X. Zhang, D. Xu, S. Jha, G. F. Ciocarlie *et al.*, "MCI: Modeling-based causality inference in audit logging for attack investigation." in *NDSS*, 2018.
[94] Y. Ji, S. Lee, M. Fazzini, J. Allen, E. Downing, T. Kim, A. Orso, and W. Lee, "Enabling refinable cross-host attack investigation with efficient data flow tagging and tracking," in *USENIX Security*, 2018.
[95] R. Yang, S. Ma, H. Xu, X. Zhang, and Y. Chen, "Uiscope: Accurate, instrumentation-free, and visible attack investigation for gui applications," in *NDSS*, 2020.
[96] L. Yu, S. Ma, Z. Zhang, G. Tao, X. Zhang, D. Xu, V. E. Urias, H. W. Lin, G. Ciocarlie, V. Yegneswaran *et al.*, "Alchemist: Fusing application and audit logs for precise attack provenance without instrumentation," in *NDSS*, 2021.
[97] P. Gao, X. Xiao, Z. Li, F. Xu, S. R. Kulkarni, and P. Mittal, "AIQL: Enabling efficient attack investigation from system monitoring data," in *USENIX ATC*, 2018.
[98] P. Gao, X. Xiao, D. Li, Z. Li, K. Jee, Z. Wu, C. H. Kim, S. R. Kulkarni, and P. Mittal, "SAQL: A stream-based query system for real-time abnormal system behavior detection," in *USENIX Security*, 2018.
[99] P. Fei, Z. Li, Z. Wang, X. Yu, D. Li, and K. Jee, "Seal: Storage-efficient causality analysis on enterprise logs with query-friendly compression," in *USENIX Security*, 2021.
[100] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *ACM WWW*, 2017.
[101] F. Zhang, N. J. Yuan, D. Lian, X. Xie, and W. Ma, "Collaborative knowledge base embedding for recommender systems," in *ACM KDD*, 2016.
[102] X. Wang, D. Wang, C. Xu, X. He, Y. Cao, and T. Chua, "Explainable reasoning over knowledge graphs for recommendation," in *AAAI*, 2019.
[103] B. Hu, C. Shi, W. X. Zhao, and P. S. Yu, "Leveraging meta-path based context for top- N recommendation with A neural co-attention model," in *ACM KDD*, 2018.

[104] Y. Xian, Z. Fu, S. Muthukrishnan, G. de Melo, and Y. Zhang, "Reinforcement knowledge graph reasoning for explainable recommendation," in *ACM SIGIR*, 2019.
[105] A. Tsymbal, "The problem of concept drift: definitions and related work," in *Computer Science Department, Trinity College Dublin*, 2004.
[106] A. Roy and S. Pan, "Incorporating extra knowledge to enhance word embedding." in *IJCAI*, 2020.
[107] N. Michael, J. Mink, J. Liu, S. Gaur, W. U. Hassan, and A. Bates, "On the forensic validity of approximated audit logs," in *ACSAC*, 2020.

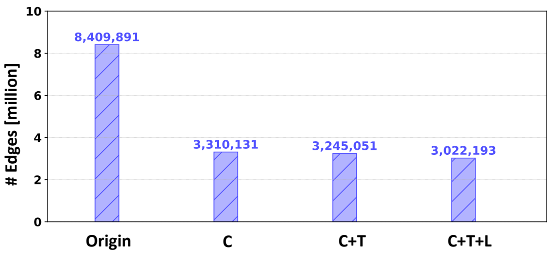
Appendix A
Discussion of Online Detection.

We currently design ShadeWatcher to perform offline cyber threat analysis similar to existing intrusion detection systems (e.g., [11], [14], [21]). As our offline implementation can process large-scale system entity interactions within seconds, it is already possible for ShadeWatcher to perform real-time analysis. However, adapting ShadeWatcher to an online approach brings new challenges. For example, similar to most learning approaches using embedding techniques, ShadeWatcher must retrain its embedding space over time for new system entities. This phenomenon is well known as *concept drift* [105]. One potential solution is to take morphological knowledge [106] of system entities (e.g., file path, owner, and privileges) into consideration to generalize and capture the semantics of previously unobserved entities. We leave this optimization as future work.

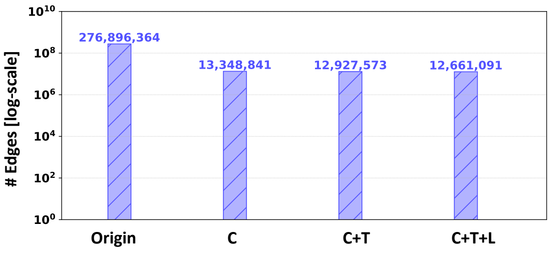
Appendix B
Noise Reduction

To cut down irrelevant or redundant audit records in the casual analysis of cyber threats, we adopt several noise reduction techniques from previous work while constructing a provenance graph. Specifically, Causality Preserved Reduction [6] (CPR) aims to aggregate audit records with identical provenance impact scope. For example, in Figure 1a, *gtcache* receives network packages from *146.153.68.151:80* 220 times for a single data transfer operation. As the absence of redundant *receive* activities does not alter any data provenance, CPR merges them and only keeps the first occurrence. Additionally, recent work [46] discovers that system activities on specific files are not helpful for provenance analysis. For example, applications regularly create temporary files to store intermediate results and delete them after termination. As these files exclusively interact with a single process during their lifecycle, they are of no interest for causal analysis and can be safely eliminated. Another type of noisy record to be removed comes from interactions with read-only libraries during the process initialization [47]. Towards this end, we define a whitelist of trusted libraries and remove them if not affecting the correctness of the causal analysis.

Figure 9 shows the number of edges in provenance graphs before and after using different noise reduction techniques. We observe that ShadeWatcher cuts off around 95% and 65% edges on the TRACE and Simulated datasets compared with the original provenance graphs. Since ShadeWatcher removes a large volume of noisy audit records, the quality of system entity interactions is significantly improved for cyber threat detection. Note that our reduction statistics is not the same as the recent study [107] due to different system entity granularity levels as discussed in § VIII-A.



(a) TRACE dataset



(b) Simulated dataset

Fig. 9: Effectiveness of different reduction techniques, where C, T, and L represent causality preserved reduction, temporary file reduction, and system library reduction, respectively.

Appendix C
Positive Interaction Sampling

For each system entity interaction observed in normal audit records, we treat it as a negative (benign) instance. Then, we randomly sample unobserved interactions as positive (potentially malicious) instances. Notice that this sampling strategy does not guarantee to generate malicious instances.

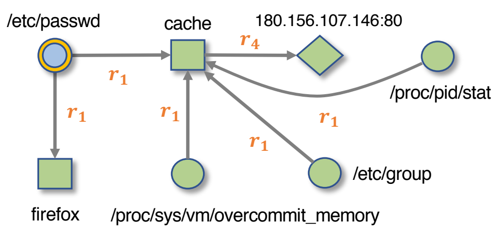
A similar phenomenon can be found in the recommendation domain. Typically, users are reluctant to express their preferences on items, and recommendation systems have to infer them from implicit user behaviors, such as clicks and purchases. Nevertheless, the items that a user dislikes still cannot be determined from user-item interactions. To address this challenge, recommendation systems [29], [100], [102] assume items with which a user never interacts to be disliked. Although unobserved interactions do not yet indicate preferences[4], it is an acceptable recommendation objective to rank observed interactions higher than unobserved ones.

Since ShadeWatcher performs anomaly detection, we follow a similar assumption to recommend a higher probability of being adversarial for unobserved system entity interactions. To provide a deep understanding, we compare ShadeWatcher's detection effectiveness with and without sampled positive interactions. The results are summarized in Table VIII. We find that our sampling strategy brings significant detection improvements. This is reasonable because a recommendation model cannot differentiate between benign and malicious interactions by training only on negative instances.

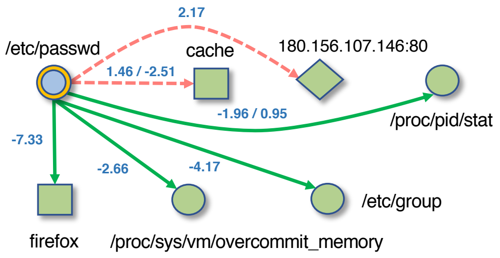
[4]The unobserved user-item interactions are a mixture of actual dislikes and missing values (i.e., users may interact with an item in the future).

TABLE VII: Overview of attack scenarios in the TRACE and Simulated datasets.

Dataset	Attack Scenario	Description of Scenario
TRACE Dataset [48]	Extension Backdoor	An attacker exploits a target host via a vulnerable browser plugin pass-mgr. The compromised plug-in downloads and executes a malicious program, which scans ports for internal recon and exfiltrates sensitive information.
	Firefox Backdoor	A malicious ad server exploits Firefox to execute an in-memory payload. This provides a remote console to exfiltrate sensitive information. A cache process is also exploited and displays similar behaviors as the compromised Firefox.
	Pine Backdoor	Pine is compromised by a malicious executable to scan ports for internal recon and establish a connection to the attacker's machine.
	Phishing Executable	An attacker sends a malicious executable as an e-mail attachment to exploit a vulnerability in Pine. However, the attack fails even though a user manually downloads and runs the executable.
Simulated Dataset	Configuration Leakage [31]	A downloaded txt file leverages the code executable vulnerability in vim to collect machine configuration for future system compromise preparation.
	Content Destruction [36]	An insider tampers with classified programs and documents.
	Cheating Student [70]	A student compromises OpenSSL service in a teaching assistant's server to steal exam answers.
	Illegal Storage [36]	An attacker creates a directory in another user's home directory and uses it to store illegal files.
	Passwd Gzip Scp [6]	An attacker steals user account information from /etc/passwd file, compresses it using gzip, and transfers the data to a remote machine using ssh service.
	Passwd Reuse [31]	An administrator reads encrypted user password from /etc/shadow file, decodes it with John, and uses the plaintext to log in on other applications.



(a) Provenance Graph



(b) Bipartite Graph

Fig. 10: Recommendations on the Firefox Backdoor.

TABLE VIII: Impact of positive interaction sampling on detection. *Positive Rate* (*PR*) shows the ratio of sampled positive instances to negative interactions. We use *PR=2* to report experimental results in § VIII due to the best accuracy.

Positive Rate	TRACE Dataset			
	Precision	Recall	F1 Score	Accuracy
0	0%	0%	0%	0.008
1	100%	99.90%	0.9995	0.9990
2	**99.98%**	**99.99%**	**0.9998**	**0.9996**
3	99.63%	99.97%	0.9980	0.9959

Appendix D
Audit Data Collection

To collect whole-system audit data, we use the Linux Audit [43] with the following system calls: *read*, *write*, *open*, *close*, *mq_open*, *openat*, *unlink*, *link*, *linkat*, *unlinkat*, *rmdir*, *mkdir*, *rename*, *pipe*, *pipe2*, *dup*, *dup2*, *fcntl*, *clone*, *fork*, *vfork*, *execve*, *kill*, *sendto*, *recvfrom*, *sendmsg*, *sendmmsg*, *recvmsg*, *recvmmsg*, *connect*, *socket*, and *getpeername*.

Although we implement ShadeWatcher to take inputs from Linux Audit, it can also be extended to support other audit sources: CamFlow [85] for Linux, ETW for Windows, and Dtrace for FreeBSD.

Appendix E
Attack scenario description

Table VII presents the overview of attack scenarios in the TRACE and Simulated datasets. Note that the original TRACE dataset consists of five APT attacks. However, because Phishing Email is invisible in audit records [17], we do not include it in the evaluation. More specifically, once a user visits a phishing website and enters credentials, subsequent system activities would be identical to normal workloads. Therefore, system auditing by nature cannot detect Phishing Email.

Appendix F
Case Study on Firefox Backdoor

In this incident, *Firefox* is compromised by a malicious ad server to steal sensitive user information. Then, multiple *cache* processes are launched and display similar behaviors. As the provenance of *cache* is missing in the original TRACE dataset [17], we only capture the malicious behavior rooted at */etc/passwd* for the second part of the attack in Figure 10a.

Firefox Backdoor generates a total of 251 system entity interactions, eight of which are manually labeled as cyber threats. Figure 10b illustrates ShadeWatcher's recommendation results. The only false negative comes from the interaction between */etc/password* and one of the seven *cache* processes. We also see that the interaction between */etc/passwd* and *180.156.107.146:80* is given a very high probability (2.17) to be adversarial, which matches our intuition that sensitive files normally do not interact with public networks. Another interesting observation is that ShadeWatcher generates 12 false-positive interactions with */proc/pid/stat*. By looking into these false alarms, we discover that they are all associated

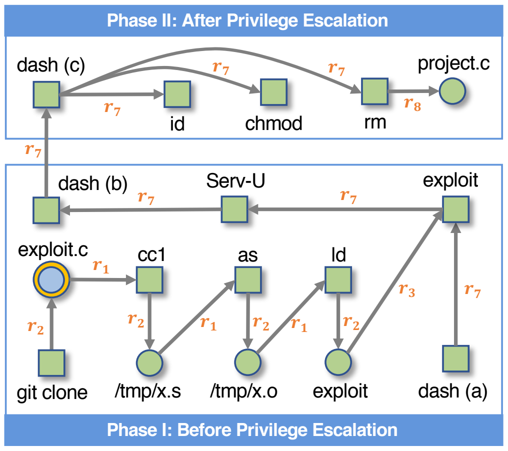


Fig. 11: A simplified provenance graph of Content Destruction, where r_8 denotes *delete*. Dash (a), dash (b), and dash (c) are */bin/dash* with different command-line arguments: "-sh", "sh -c chmod u+s /home/test/CVE; id; echo 'opening root shell'; /bin/sh;", and "/bin/sh".

with files that are rarely accessed in a running system (e.g., */proc/2457/stat*). This reveals a limitation of our first-order modeling, which differentiates system entities via the nuance of contextual information. In particular, SHADEWATCHER cannot uncover side information of entities without rich contexts. One possible solution to improve SHADEWATCHER's effectiveness is incorporating morphological knowledge (e.g., file path) to help capture entity semantics. Nevertheless, obtaining high-quality representations of system entities is a separate research topic that requires non-trivial efforts. For example, it is challenging to determine useful features from numerous entity attributes and effectively integrate them into the final representations. While morphological information is not considered in this study, our proposed knowledge graph provides a unified representation that can conveniently combine different sources of knowledge from system auditing.

APPENDIX G
ATTACK WITH PRIVILEGE ESCALATION

Recall that after an attacker escalates privilege on a host, he/she can tamper with audit data at will to cover attack footprints. However, as mentioned in § III-B, the attacker has no way of manipulating previous audit records that have tracked the evidence of privilege escalation via a variety of secure auditing solutions (e.g., centralized auditing servers). Therefore, it is necessary to explore the effectiveness of a detection system in a scenario where the only audit records available are the ones from before privilege escalation.

Towards this end, we evaluate how SHADEWATCHER performs on audit data before the attacker escalates privilege in Content Destruction. In this scenario, an attacker exploits a local privilege escalation vulnerability of SolarWinds Serv-U[5] to get a root shell and delete classified files. The attack is launched in two phases. In the first phase, the attacker downloads (*git*) the source code of an exploit and compiles (*gcc*) it to an executable (*exploit*). Then, the attacker runs the executable (*./exploit*) to open a root shell (*dash*). In the second phase, the attacker adopts the privileged shell to discover (*ls*) and delete (*rm*) sensitive data (*project.c*) on the victim's machine. A simplified provenance graph built upon Content Destruction is shown in Figure 11, where the attacker's activities before and after privilege escalation are separated into two boxes, namely, Phase I and Phase II.

[5]https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-12181

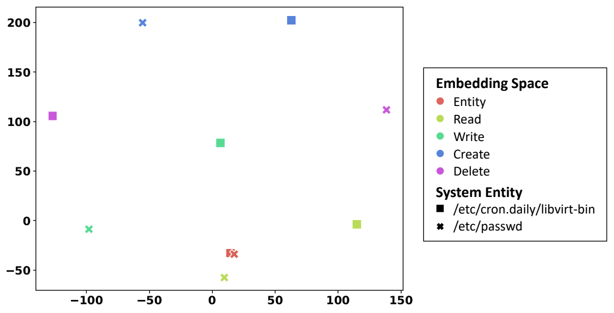


Fig. 12: Visualizing system entity embeddings with t-SNE.

To evaluate SHADEWATCHER on audit data before privilege escalation, we perform threat detection on 46 system entity interactions extracted from Phase I in Figure 11. Two of 46 interactions that exploit the Serv-U vulnerability (i.e., *exploit.c*$\xrightarrow{r_0}$*Serv-U* and *exploit.c*$\xrightarrow{r_0}$*/bin/dash (b)*) are labeled as malicious. Observe that in Table IX, both malicious interactions are recommended with high probabilities (well beyond -0.5) of being adversarial. Additionally, SHADEWATCHER predicts the rest of the interactions in Phase I as benign activities. The experimental results verify our hypothesis in § III-B that even without audit data after privilege escalation, SHADEWATCHER still can detect cyber threats by identifying system entity interactions used for escalating privileges.

TABLE IX: Recommendations on two malicious system entity interactions before privilege escalation.

Initiator	Target	Score
/home/test/CVE/exploit.c	/usr/local/Serv-U/Serv-U	4.65
/home/test/CVE/exploit.c	/bin/dash (b)	4.70

APPENDIX H
VISUALIZATION OF SYSTEM ENTITY EMBEDDING.

We use the t-SNE technique to visualize TransR's embedding spaces on a two-dimensional plane. In particular, Figure 12 shows the embeddings of */etc/passwd* and */etc/cron.daily/libvirt-bin* in entity and relation spaces. The entity space intuitively reflects the embeddings of TransE and TransH. We can observe that 1) */etc/passwd* has various representations when involved in *read*, *write*, *create*, and *delete* relations; and 2) */etc/passwd* and */etc/cron.daily/libvirt-bin* are nearby in the entity space but far away from each other in specific relation spaces. The visualization confirms that TransR captures intrinsic characteristics of system entities in the contexts of different relations, while TransE and TransH cannot compare the difference.