SYSREP: Towards Automatic Attack Investigation via Weight-Aware Reputation Propagation from System Monitoring

Abstract

The need for countering Advanced Persistent Threat (APT) attacks has led to solution that ubiquitously monitor system activities in each host, and perform timely attack investigation over the monitoring data. However, existing solutions face major limitations in automating the process of attack investigation: (1) causality analysis requires security analysts to inspect a large system dependency graph, where nodes represent system entities and edges represent the dependencies among system auditing events; (2) behavior querying requires security analysts to master the query syntax and manually construct behavior queries to search attack patterns.

In this work, we propose a novel approach, SYSREP, which makes the first step towards automatic attack investigation. SYSREP assigns discriminative weights to the edges in a system dependency graph to distinguish *critical edges* that are important to revealing the attack sequence, and propagates reputation values from seed sources to suspicious system entities. The reputation propagation enables SYSREP to automatically determine whether a system entity came from a trusted or a suspicious source, and the edge weights enable SYSREP to automatically reveal critical edges for reconstructing the attack sequence. The evaluations on a wide range of attacks on key system interfaces and real APT attacks demonstrate the effectiveness of SYSREP in reputation propagation and attack sequence reconstruction.

1 Introduction

Advanced Persistent Threat (APT) attacks [20, 52] have plagued many well-protected companies, causing significant financial losses [5–7, 18, 19, 45, 55]. These APT attacks often infiltrate into target systems in multiple stages and span a long duration of time with a low profile, posing challenges for efficient detection and investigation. To counter these advanced attacks, *ubiquitous system monitoring* emerges as an important approach for monitoring system activities and performing attack investigation [24, 25, 29, 31, 32, 34, 35, 38, 39]. System monitoring collects system-level auditing events about system calls, and the collected data further enables security analysts to unfold these attacks to identify root causes and ramifications, which is critical for performing timely system recovery and preventing future compromise.

While the research on audit logging and forensics shows great potential, there has been little work in automating the process of attack investigation. Existing attack investigation solutions based on causality analysis [26, 33–35, 38] and behavior querying [24, 25] require non-trivial efforts of manual inspection, and thus these approaches face the limitations of reaching the goal of automatic attack investigation. Causality analysis assumes causal relationships between system entities (*e.g.,* processes) and system resources (*e.g.,* files and network sockets) involved in one same system call event (*e.g.,* a process reading a file), based on which it organizes these system call events as a dependency graph. By inspecting such a dependency graph, security analysts can trace from the POI (Point-Of-Interest) events to identify the entry points and the ramifications of attacks. The major limitation of causality analysis, however, is the significant manual efforts of inspecting a daunting number of edges (typically, tens of thousands) due to dependency explosion [36, 54, 57]. Behavior querying [24, 25] leverages domain-specific languages (DSLs) to investigate the attack patterns of system call events. The construction of behavior queries, however, requires security analysts to master the syntax of DSLs (typically, by going through a steep learning curve) and construct the queries manually and correctly, which is labor-intensive and error-prone.

Contributions. In this work, we make the first step towards automatic attack investigation via a reputation propagation paradigm built upon system monitoring. We propose a novel approach, SYSREP, which first assigns discriminative weights to edges in a dependency graph to construct a *weighted dependency graph* and then propagates reputation scores in a weight-aware manner from seed sources to system entities in the POI events (referred to as *POI entities*).

The key insight of SYSREP is leveraging discriminative edge weights to differentiate *critical edges* (*e.g.,* downloading malicious scripts) from the non-critical edges (*e.g.,* irrelevant file copies). Critical edges represent the events that are more important w.r.t. POI events for attack sequence construction,

including the events that execute malicious payloads and the events that are required to correlate the malicious payloads with the POI events. These discriminative edge weights enable SYSREP to *automatically reconstruct the attack sequence*. The reputation propagation paradigm enables SYSREP to *automatically determine the suspiciousness/trustworthiness of POI entities* based on whether the system entities originate from suspicious sources (*e.g.,* USB sticks and suspicious IPs) or trusted sources (*e.g.,* verified binaries and trusted websites). The synergy of attack sequence reconstruction and reputation propagation makes SYSREP the first step towards automatic attack investigation.

Challenges. Due to the characteristics of system dependency graph, achieving discriminative weight assignment and reputation propagation for automatic attack investigation faces three major challenges. *Unequal Impacts of Edges*: in a dependency graph produced by causality analysis, *critical edges* that reveal the attack provenance are often buried in a huge number of non-critical edges [34–36, 38]. If the reputation propagation treats all edges equally, then it cannot ensure that the reputation scores propagated through the critical edges will have the more impact on the POI entities than those propagated via non-critical edges. As a result, POI entities that are supposed to be correlated with suspicious sources may be incorrectly correlated with irrelevant system entities as their sources. *Diversified Attack Cases*: critical edges in different attack scenarios often present different properties, and thus relying on a single feature, such as outgoing degree, for computing edge weights is often insufficient to oversee a diversified set of attack cases. *Long Dependency Path*: due to the multi-stage nature of APT attacks [20,52], the dependency path from suspicious sources to POI entities often has many hops. If the reputation propagates in a distributed manner (*i.e.,* a node distributes its reputation along its outgoing edges in certain propagation) [15, 46], the node's reputation will degrade rapidly in the middle of the path, resulting in little impact on the POI entities.

Novel Techniques of SYSREP. To address the aforementioned challenges, SYSREP employs three novel techniques.

Weighted Reputation Propagation: to ensure that reputation propagated through critical edges has more impact on the POI entities than non-critical paths, SYSREP first assigns discriminative weights to edges in the dependency graph to represent the importance of the edges w.r.t. the POI events, and then propagates reputation, proportional to the edge weights, along the paths in the graph.

Discriminative Local Feature Projection: to model the importance of edges w.r.t. the POI event, SYSREP extracts three discriminative features, instead of relying on a single "golden" feature, from a system auditing event, including (i) **relative data size difference** that measures the distance between the size of data processed by the system call and the size of POI entity; (ii) **relative time difference** that measures the distance between the timestamps of the dependency event and the POI event; (iii) **concentration degree** that measures the ratio of the indegree to the outdegree of the involved system entity, which is particularly useful for smoothing out the impacts of system libraries with no incoming edges and long-running processes with many outgoing edges (Section 4.3).

To compute a weight score, instead of adopting a standard classification-based approach (*i.e.,* training a classifier using the three features and outputting a score as the weight), which has limited generalization capability in our problem context (*e.g.,* due to the very limited training data, highly imbalanced classes, etc.), SYSREP leverages ideas from dimensionality reduction and discriminant analysis. Rather than computing a projection vector globally, which may result in serious bias due to the large number of edges, SYSREP employs a novel *discriminative local feature projection* mechanism based on an extended version of Linear Discriminant Analysis (LDA) [42]. For each node, the mechanism computes a discriminative projection vector locally for all its incoming edges and computes a weight for each incoming edge by projecting its three features. This mechanism helps avoid the undesired impacts of unlinked edges to the node, while ensuring that the weights of critical edges are maximally separated from the weights of non-critical edges (Section 4.4);

Reputation Inheritance: to avoid the fast degradation of reputation during propagation, SYSREP employs an *inheritance fashion* opposed to the distribution fashion: when a node propagates its reputation to a receiving node, it considers only the impact it may have, measured by the edge weight, on the receiving node, without distributing its reputation equally among all downstream nodes. This scheme preserves source reputations on a group of highly-related nodes (reflected by weights) even when propagating along long paths (Section 4.5).

Evaluation. We implemented and deployed SYSREP in a server to collect real system monitoring data, and evaluated SYSREP in both benign and malicious scenarios. We performed 8 tasks that inject benign and malicious payloads through key system interfaces (*e.g.,* web downloads and shell executions) and 5 real APT attacks in the deployed environment, and applied SYSREP to perform automatic attack investigation. During our evaluation, the server continues to resume its routine tasks to emulate the real-world deployment where irrelevant system activities and attack activities co-exist. In total, we collected $\sim$2 billion auditing events for the attacks. For evaluation purpose, we set the range of the reputation score to be $[0.0, 1.0]$, with 0.0 for suspicious sources and 1.0 for trusted sources.

Reputation Propagation. To evaluate the effectiveness of SYSREP in reputation propagation, we compare the reputation scores of POI entities against the expected values (0.0 for malicious scenarios and 1.0 for benign scenarios). Our results show that SYSREP is able to accurately propagate the reputation scores from trusted and suspicious sources to the POI entities (averagely 0.99 for benign scenarios and averagely 0.03 in malicious scenarios).

To demonstrate the effectiveness of SYSREP's discrimina-

tive local feature projection, we compare SYSREP with three other weight computation approaches: (1) LP-FIXED, which leverages a fixed set of parameters for projection; (2) LP-GLOBAL, which groups all the edges into two clusters and derives a projection to separate the edges in different clusters; (3) LP-GLOBAL+, which is the same as LP-GLOBAL but removes outlier edges. The results show that SYSREP on average improves LP-FIXED, LP-GLOBAL, and LP-GLOBAL+ by 64.60%, 79.22%, 70.36%, respectively, in benign scenarios and malicious scenarios.

We also compare SYSREP with the edge priority computation used in the state-of-the-art causality analysis work PrioTracker [38]. For fair comparison, we adapt their edge priority computation algorithm to assign weights to edges and apply the same propagation algorithm to propagate reputation scores. The results show that SYSREP achieves 57.22% improvement over PrioTracker in benign scenarios and average 87.22% improvement over PrioTracker in malicious scenarios.

Attack Sequence Reconstruction. To evaluate the effectiveness of SYSREP in attack sequence reconstruction, we choose a range of threshold values to prune edges whose weights are below the threshold, and inspect the remaining edges to measure the possible loss of critical edges. The results show that for all attacks, the reputation scores of critical edges (mostly > 0.9) are well separated from the non-critical edges (mostly < 0.1), demonstrating the effectiveness of our discriminative weights. Additionally, setting threshold values within $[0.1, 0.25]$ can prune more than 90% of the irrelevant edges while preserving the critical edges.

2 Background and Motivating Example

2.1 System Monitoring

System monitoring collects auditing events about system calls that are crucial in security analysis, describing the interactions among system entities. As shown in previous studies [24–26, 29, 31, 32, 34, 35, 38, 39], on mainstream operating systems (Windows, Linux, and Mac OS), system entities in most cases are files, processes, and network connections, and the collected system calls are mapped to three major types of system events: (i) file access, (ii) processes creation and destruction, and (iii) network access. As such, we consider *system entities* as *files*, *processes*, and *network connections*. We consider a *system event* as the interaction between two system entities represented as $\langle$*subject, operation, object*$\rangle$. Subjects are processes originating from software applications (*e.g.,* Chrome), and objects can be files, processes, and network connections. We categorize system events into three types according to the types of their object entities, namely *file events*, *process events*, and *network events*.

Both entities and events have critical security-related attributes (Tables 1 and 2). The attributes of entities include the properties to support various security analyses (*e.g.,* file name, process name, and IP address), and the unique identifiers to distinguish entities (*e.g.,* file path, process name and PID, IP and port). The attributes of events include event origins (*e.g.,* start time/end time), operations (*e.g.,* file read/write), and other security-related properties (*e.g.,* failure code).

Table 1: Representative attributes of system entities

Entity	Attributes	Shape in Graph
File	Name, Path	Ellipse
Process	PID, Name, User, Cmd	Square
Network Connection	IP, Port, Protocol	Parallelogram

Table 2: Representative attributes of system events.

Operation	Read/Write, Execute, Start/End
Time	Start Time/End Time, Duration
Misc.	Subject ID, Object ID, Data Amount, Failure Code

2.2 Causality Analysis

Causality analysis [26, 33–35, 38] analyzes the auditing events to infer the dependencies among system entities and present the dependencies as a directed graph. In the dependency graph $G(E,V)$, a node $v \in V$ represents a process, a file, or a network network. An edge $e(u,v) \in E$ indicates a system auditing event involving two entities u and v (*e.g.,* process creation, file read or write, and network access), and its direction (from the source node u to the sink node v) indicates the data flow. Each edge is associated with a time window, $tw(e)$. We use $ts(e)$ and $te(e)$ to represent the starting time and the ending time of e. Formally, in the dependency graph, for two event edges $e_1(u_1, v_1)$ and $e_2(u_2, v_2)$, there exists causal dependency between e_1 and e_2 if $v_1 = u_2$ and $ts(e_1) < te(e_2)$.

Causality analysis enables two important security applications: (i) *backward causality analysis* that helps identify the entry points of attacks, and (ii) *forward causality analysis* that helps investigate the ramifications of attacks. Given a POI event $e_s(u,v)$, a backward causality analysis traces back from the source node u to find all events that have causal dependencies on u, and a forward causality analysis traces forward from the sink node v to find all events on which v has causal dependencies.

2.3 Motivating Example

Figure 1 shows an example dependency graph of a data leakage attack: the attacker exploited the shellshock bug to execute `curl` in the target system and downloaded a backdoor program `bdoor`. The attacker then executed the `bdoor` program to open a backdoor on port 44444. Through the backdoor, the attacker collected sensitive data using `tar`, `bzip2`, and `gpg`, and uploaded the collected data to a remote host. The POI entity is a network connection to the suspicious remote host. The complete graph contains 5,817 nodes and 215,380 edges, which is impractical for a human analyst to manually inspect.

SYSREP enables automatic attack investigation by assigning weights to edges and propagating reputation scores from seed sources to POI entities. In this example, we set the reputation score for the suspicious network source (*i.e.,* the top

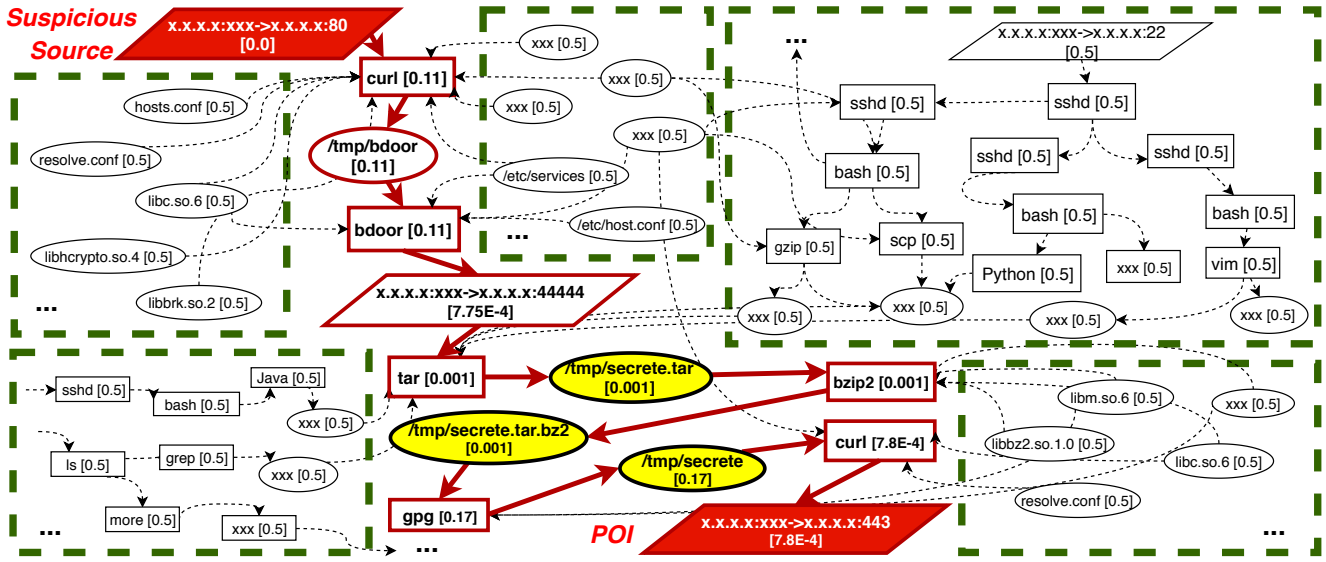

Figure 1: Partial dependency graph of a motivating data leakage attack case. Rectangles denote processes, ovals denote files, and parallelograms denote network sockets. Yellow ovals are the files leaked in this attack. Nodes in the attack path have red frames while normal activities and system libraries are in dashed green rectangles. Critical edges are represented with solid red arrows. Non-critical edges are represented with dashed arrows. The complete graph contains 5,817 nodes and 215,380 edges. Note that the reputation of the suspicious source on the top left propagates mainly through the critical edges.

left parallelogram) to 0.0 and the reputation scores for system libraries to 0.5.

Challenges. From Figure 1, we can see that the critical edges (*i.e.,* the red bold edges) representing the attack sequence are buried in many irrelevant system activities, such as remote login via ssh and normal program execution like Java and Python. Thus, SysRep needs to assign high weights to the critical edges to distinguish them from edges representing irrelevant system activities, and propagates reputation scores based on the weights to ensure that the critical edges have the most impact on the POI entity. Furthermore, the dependency path from the suspicious source to the POI entity has more than 10 hops, where each node in the middle of the path also receives reputation from other sources. As such, SysRep needs to make sure that the reputation propagation does not suffer from fast degradation in a long path with many irrelevant nodes that can potentially distort the final reputation score.

Techniques of SysRep. SysRep first computes the weights using three novel discriminative features (Section 4.3). In this example, compared with other (*i.e.,* non-critical) edges, the critical edges have more similar data amount as the data amount transmitted in the POI event, the start timestamps of these edges are closer to the POI event, and the incoming and outgoing edges are relatively small for the nodes involved in the critical edges. Thus, by combining these features into the edge weights (Section 4.4), SysRep maximizes the differences between critical edges and other edges, and reveals the critical edges. Based on the weights, SysRep adopts an inheritance fashion that considers the impact of the edges to propagate the reputation scores from the seed sources in the graph (Section 4.5). From Figure 1, we observe that the reputation scores of nodes on the critical edges are close to 0, as compared to the irrelevant system libraries with reputation scores close to 0.5. By connecting the nodes with low reputation scores and the critical edges, we are able to reconstruct the attack sequence. We also observe that the POI entity receives a very low reputation score that is almost 0, which indicates that it comes from a suspicious source as expected.

3 Overview and Threat Model

Figure 2 illustrates the architecture of SysRep. SysRep consists of three phases: (1) dependency graph generation, (2) graph preprocessing & discriminative weight computation, and (3) attack investigation. In Phase I, SysRep leverages mature system auditing tools (*e.g.,* auditd [49], ETW [41], DTrace [16], and Sysdig [53]) to collect system-level audit logs about system calls. Given a POI event, SysRep parses the collected logs and performs causality analysis [34, 35] to generate the dependency graph for the event (Section 4.1). In Phase II, SysRep preprocesses the graph by merging the same type of edges between two nodes that occur within a time window threshold to reduce the graph size and splitting the nodes to remove parallel edges. This process transforms the large dependency graph into a simple directed graph (Section 4.2), which is easier for weight computation and reputation propagation. To produce a weighted dependency graph such that critical edges are easily distinguishable from non-critical edges by their weights, for each edge, SysRep extracts three novel features that model the importance of the event edge w.r.t. the POI event from three dimensions: time

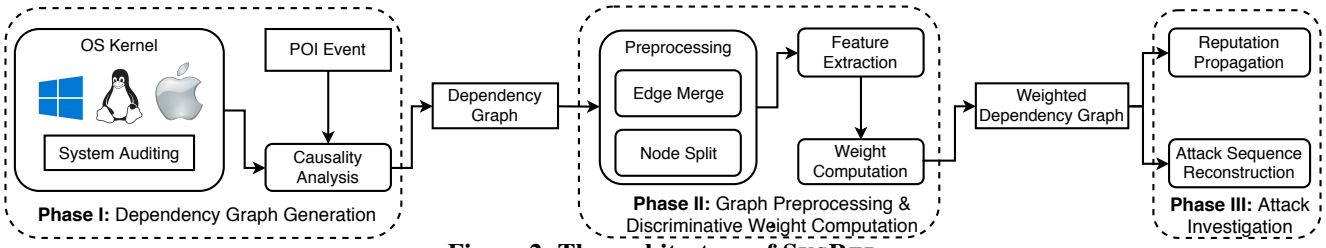

Figure 2: The architecture of SysRep

Table 3: System calls processed by SysRep

Event Category	Relevant System Call
Process/File	read, write, readv, writev
Process/Process	execve, fork, clone
Process/Network	read, write, sendto, recvfrom, recvmsg

dimension, data amount dimension, and structure dimension (Section 4.3). SysRep then employs a novel *local feature projection mechanism* to project the three features to a combined, discriminative weight (Section 4.4). In Phase III, SysRep enables automatic attack investigation by (1) propagating reputation from seeds sources on the weighted dependency graph to automatically determine the suspiciousness/trustworthiness of POI entities (Section 4.5) and (2) providing a suggested range of threshold values based on the discriminative edge weights to automatically reconstruct attack sequence by distinguishing critical edges from other edges.

Threat Model. Our threat model follows the threat model of previous work on system monitoring [13, 24, 25, 29, 34, 35, 37, 38]. We assume that the system monitoring data collected from kernel space [16, 41, 49, 53] is not tampered, and that the kernel is trusted. Any kernel-level attack that deliberately compromises security auditing systems is beyond the scope of this work.

4 The Design of SysRep

In this section, we present the three phases and the components of SysRep in detail.

4.1 Phase I: Dependency Graph Generation

System Auditing. SysRep leverages system auditing tools [16, 41, 49, 53] to collect information about system calls from the kernel, and then parses the collected events to build a global dependency graph. SysRep focuses on three types of system events: (i) file access, (ii) processes creation and destruction, and (iii) network access. Table 3 shows the representative system calls (in Linux) processed by SysRep. Particularly, for a process entity, we use the process name and PID as its unique identifier. For a file entity, we use the absolute path as its unique identifier. For a network connection entity, as processes usually communicate with some servers using different network connections but with the same IPs and ports, treating these connections differently greatly increases the amount of data we trace and such granularity is not required in most of the cases [24, 25, 38]. Thus, we use 5-tuple ($\langle srcip, srcport, dstip, dstport, protocol \rangle$) as a network connection's unique identifier. Failing to distinguish different entities causes problems in relating events to entities and tracking the dependencies of events. For each system call, SysRep extracts the security-related attributes of entities (Table 1) and events (Table 2). SysRep filters out failed system calls, which could cause false dependencies among events.

Causality Analysis. Given a POI event e_s, SysRep applies causality analysis (Section 2.2) to produce a dependency graph G_d, as shown in Algorithm 1[1]. Algorithm 1 adds e_s to a queue (Line 2), and repeats the process of finding eligible incoming edges of the edges (*i.e.,* incoming edges of the source nodes of edges) in the queue (Lines 3-9) until the queue is empty. The output is a dependency graph that only contains relevant system entities and events w.r.t. the given POI event.

Algorithm 1: Backward Causality Analysis

```
   Input: POI Event: e_s, Global Dependency Graph: G
   Output: Dependency Graph for the POI Event: G_d
1  G_d ← new Graph()
2  Queue.add(e_s)
3  while Queue is not Empty do
4      u ← Queue.pop().source
5      set ← G.incomingEdgeOf(u)
6      for e ∈ set do
7          if ts(e) < te(e_s) then
8              G_d.add(e)
9              Queue.add(e)
10         end
11     end
12 end
```

4.2 Phase II: Graph Preprocessing

SysRep performs graph preprocessing to transform the dependency graph from a directed multigraph to a simple directed graph with no parallel edges.

Edge Merge. The dependency graph produced by causality analysis often has many edges between the same pair of nodes [57]. The reason for generating these excessive edges is that the OS typically finishes a read/write task (*e.g.,* file read/write) by distributing the data to multiple system calls and each system call processes only a portion of the data. Inspired by the recent work that proposes Causality Preserved

[1] Forward causality analysis can be implemented in a similar way.

Reduction (CPR) [57] for dependency graph reduction, SysRep merges the edges between two nodes. As shown in the study [57], CPR does not work well for processes that have many interleaved read and write system calls, which introduces excessive causality. As such, SysRep adopts a more aggressive approach: for edges between two nodes that represent the same system call, SysRep will merge them into one edge if the time differences of these edges are smaller than a given time threshold. Empirically, we set the time threshold as 10 seconds, which is large enough for most processes to finish file transfers and network communications in modern computers. Since such merge is performed after the dependency graph generation, all the dependencies are still preserved but only the time windows of certain edges are merged.

Node Split. After the edge merge, the dependency graph may still have parallel edges (*i.e.,* edges indicating read or write from different system calls). For example, a process may receive data from a network socket via both `read` and `recvfrom`. These parallel edges create complications for weight computation: for a node, some of its incoming edges' time windows may violate the causal dependencies on the outgoing edges.

To address the problem, SysRep first enumerates all the pairs of nodes that have parallel edges. For a pair of nodes (u,v) that have parallel edges from u to v, SysRep splits u into multiple copies and assigns each copy to one parallel edge, so that each copy of u only has one outgoing edge to v. The copies of u also inherit all u's incoming edges that have causal dependencies on the u's outgoing edges. The output is a simple directed dependency graph without parallel edges.

4.3 Phase II: Feature Extraction

For each edge, SysRep extracts three novel discriminative features that model its importance w.r.t. the POI event.

- **Relative Data Amount Difference:** For an edge $e(u,v)$, we measure the distance between the size of data processed by the system call and the size of POI entity. The intuition is that the smaller the distance is, the more important this edge is to the data flow in the POI event.

  $$f_{D(e)} = 1/(\mid s_e - s_{e_s} \mid + \alpha), \tag{1}$$

  Equation (1) gives the *relative data amount difference feature*, where s_e and s_{e_s} represent the data amount associated with the event edge e and the POI event e_s. Note that we use a small positive number α to handle the special case when e is the POI event. Thus, the POI event will have the highest data amount difference feature value $f_{D(e_s)} = 1/\alpha$. Empirically, we set $\alpha = 1e{-}4$.
- **Relative Time difference:** For an edge $e(u,v)$, we measure the distance between its end time $te(e)$ and the end time of the POI event $te(e_s)$. The intuition is that the event that occurred closer to the POI event is more temporally related to the POI event.

  $$f_{T(e)} = \ln(1 + 1/\mid t_e - t_{e_s} \mid), \tag{2}$$

  Equation (2) gives the *relative time difference* feature, where $t_{(e)}$ and t_{e_s} represent timestamp values (we use the event end time) of the event edge e and the POI event e_s. To handle the special case when e is the POI event (*i.e.,* $\mid t_e - t_{e_s} \mid = 0$), we use one tenth of the minimal time unit (nanosecond) in the audit logging framework (*i.e.,* $1e{-}10$) to compute its feature: $f_{T(e_s)} = ln(1 + 1e10)$. This ensures that the POI event has the highest feature value.
- **Concentration Degree:** One important category of non-critical edges that often appear in a causal graph are events that access system libraries [54, 57]. These edges are often associated with considerable data amount and occur at various timestamps, and hence using only $f_{T(e)}$ and $f_{D(e)}$ is less effective in distinguishing critical edges from them. To address this challenge, we observe that most system library nodes are source nodes in the corresponding edges and do not have any incoming edges. Another category of non-critical edges are events that involve long-running processes as source nodes, which often have few incoming edges but many outgoing edges. Based on this observation, we define the *concentration degree* for the edge $e(u,v)$ as:

  $$f_{C(e)} = InDegree(u)/OutDegree(u), \tag{3}$$

  where $InDegree(u)$ and $OutDegree(u)$ represent the in-degree and out-degree of the source node u in $e(u,v)$. For seed nodes, since they are very important in initiating the reputation propagation, we set their concentration degree to be 1. This feature helps smooth out the impacts of system libraries with no incoming edges and long-running processes with many outgoing edges.

Local Feature Normalization. Before using the three features to compute edge weights, it is often a good practice to scale them in the same range [11, 22]. Global feature scaling using standard methods such as range normalization and standardization does not make much sense in our context, since a node is only affected by its parents but not by its children or siblings. Recognizing this, SysRep adopts a *local feature scaling* scheme: for an edge $e(u,v)$, SysRep *locally normalizes* its each feature by the sum of corresponding features of all incoming edges of the sink node v. This scheme enables the three features of all v's incoming edges to be compared on the same scale.

4.4 Phase II: Weight Computation

For each edge, SysRep computes a discriminative weight using its three normalized features. The weight models the aggregated importance of the event edge w.r.t. the POI event in the attack sequence reconstruction, so that critical edges can be easily distinguished from non-critical edges.

One approach to compute a weight score is to adopt a classification-based approach to train a binary classifier using the three features and output a probability score. Though this approach has demonstrated its applicability for several domains [23, 56], it faces significant limitations in our problem

Algorithm 2: Weight Computation

Input: Dependency Graph for the POI event: G_d
Output: Weighted Dependency Graph, G_{wd}

```
for v ∈ G_d do
    Set ← G_d.incomingEdgeOf(v)
    group_1, group_2 ← Multi-KMeans++(Set)
    (ω*_D, ω*_T, ω*_C) ← extendedLDA(group_1, group_2)
    for e ∈ Set do
        W_e = ω*_D f_D(e) + ω*_T f_T(e) + ω*_C f_C(e)
    end
    W' ← Σ_{e∈Set} W_e
    for e ∈ Set do
        W_e ← W_e / W'
    end
end
```

context. To achieve high classification accuracy, supervised learning-based approaches often require large amount of training data and the training data and the test data come from the same distribution [11, 22]. However, in our context, features are extracted w.r.t. the specific POI event, and thus the model learned for one type of attack with a specific POI can hardly generalize to other types of attacks with different POIs. The highly imbalanced classes (*i.e.,* the very few number of critical edges as compared to non-critical edges) further impedes the model generalization.

Recognizing such limitations, SYSREP leverages ideas from dimensionality reduction and discriminant analysis [22]. Specifically, SYSREP employs a novel *discriminative local feature projection mechanism* based on an extended version of Linear Discriminant Analysis (LDA) [42], and computes a projection vector to project the three-dimensional feature vector to a one-dimensional weight, while ensuring that the projected weights of critical edges and non-critical edges are maximally separated. We next present the weight computation mechanism (as shown in Algorithm 2) in detail.

Step 1: Local Edge Clustering. Due to the localized nature of our problem context and features (Section 4.3), rather than computing a projection vector globally for all edges, SYSREP computes a projection vector *locally* for the incoming edges of each node. This helps avoid the bias introduced by the large number of irrelevant edges.

In Step 1, for each node, SYSREP locally clusters all its incoming edges in two groups, which is a prerequisite for discriminant analysis. Specifically, SYSREP adopts Multi-KMeans++ clustering algorithm [12]. Based on KMeans, KMeans++ uses a different method for choosing the initial seeds to avoid poor clustering. Multi-KMeans++ is a meta algorithm that performs n runs of KMeans++ and then chooses the best clustering that has the lowest distance variance over all clusters. Empirically, we set $k = 2$ (since we want two groups) and $n = 20$ (to ensure the best clustering is chosen).

Step 2: Local Feature Projection via Extended LDA. For each node, after locally clustering its incoming edges in two groups, SYSREP employs an extended version of Linear Discriminant Analysis (LDA) [42] to compute a projection vector, and applies the projection vector to the feature vector of each incoming edge to compute an edge weight.

Formally, for a node v, we denote the feature vectors of its N incoming edges as $x^1, x^2, \ldots, x^N$, which are clustered in two groups: group g_1 contains N_1 edges, and group g_2 contains N_2 edges (*i.e.,* $N_1 + N_2 = N$). The group mean vectors are $\mu_1 = \frac{1}{N_1}\sum_{x\in g_1} x$ and $\mu_2 = \frac{1}{N_2}\sum_{x\in g_2} x$. The between-group scatter matrix is defined as $S_b = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T$. The within-group scatter matrix is defined as $S_w = \sum_{x\in g_i}(x - \mu_i)(x - \mu_i)^T$. LDA is a technique that seeks to reduce dimensionality while preserving as much of the group discriminatory information as possible. Specifically, LDA finds a projection vector ω that maximizes the following Fisher criterion:

$$J(\omega) = \frac{\omega^T S_b \omega}{\omega^T S_w \omega} \tag{4}$$

In order words, LDA looks for the best projection direction such that the projected samples from the same group are close to each other (as enforced by the denominator $\omega^T S_w \omega$), and the projected samples from different groups are far away from each other as possible (as enforced by the numerator $\omega^T S_b \omega$). Solving the optimization problem yields:

$$\omega^* = \arg\max J(\omega) = S_w^{-1}(\mu_1 - \mu_2) \tag{5}$$

Denote the projection vector as $\omega^* = (\omega_D^*, \omega_T^*, \omega_C^*)$, for an incoming edge $e(u, v)$ of node v, its (unnormalized) weight W_e is computed as:

$$W_e = \omega_D^* f_{D(e)} + \omega_T^* f_{T(e)} + \omega_C^* f_{C(e)} \tag{6}$$

The applicability of Equation (5) requires that S_w is nonsingular (*i.e.,* S_w^{-1} exists). However, this criterion may be violated quite often in our problem context, due to the large imbalance between the number of critical edges and the number of non-critical edges. Furthermore, standard LDA only ensures that the projected values of different groups are largely separated, rather than guaranteeing which group has higher projected values, while our goal is to make critical edges have higher weights than non-critical edges. Another limitation of standard LDA is that sometimes the projected values are negative, while we require the edge weights to be a non-negative number to model the importance w.r.t. the POI event.

Recognizing such limitations, we *extend* the standard LDA in three aspects.

(a) Handling singular S_w: When S_w is singular, we select the projection vector (we normalize it first) from the following two candidates that results in a larger Fisher criterion numerator (*i.e.,* $\omega^T S_b \omega$):

- $S_w^+(\mu_1 - \mu_2)$. S_w^+ is the Moore-Penrose [10] inverse of S_w.
- $\mu_1 - \mu_2$ (*i.e.,* the direction of group mean difference)

(b) Negating the projection vector by condition: To make critical edges have higher projected values, we negate the projection vector by condition. Note that this problem is fundamentally challenging, since we do not have labels for critical

edges and thus we do not know which group contains critical edges. We approach to this problem using a set of heuristics:

- If all three dimensions of the projection vector are non-positive, negate.
- If all three dimensions of the projection vector are non-negative, do not negate.
- Else, negate by condition:
  - If only one group has seed edges and this group has a lower projected mean, negate. This is based on the insight that seed edges should have high weights.
  - Else, if the group with a smaller size has a lower projected mean, negate. This is based on the insight that the number of critical edges is smaller than the number of non-critical edges in most cases.

(3) Scaling the projected weights: We scale the projected weights to the range $[0,1]$. We further add a small offset (we use 1/100 of the difference between the smallest value and the second smallest value) to each scaled weight, so that all scaled weights are positive.

Step 3: Local Edge Weight Normalization. Same as local feature normalization (Section 4.1), for an edge $e(u,v)$, we *locally normalize* its scaled weight by the sum of corresponding weights of all incoming edges of the sink node v:

$$W_e = W_e / \sum_{e' \in IncomingEdge(v)} W_{e'}, \tag{7}$$

After normalization, the weights of all edges are in the range $(0,1]$, and the sum of weights of all incoming edges of any node (except for nodes that do not have incoming edges) is equal to 1. Note that for nodes that only have one incoming edge, we skip the local clustering and projection process and directly set its final weight to 1. This completes the Phase II by producing a weighted dependency graph with discriminative weights to distinguish critical edges from non-critical edges.

4.5 Phase III: Attack Investigation

Reputation Propagation. To perform reputation propagation, SYSREP assigns initial reputations to seed nodes and propagates the reputations using an inheritance fashion.

Reputation Scheme. The design goal of the reputation scheme is to capture the fact that if a file (or program) is downloaded or installed from a reputable source, it should be more reliable than the file (program) downloaded from an suspicious website or from an suspicious equipment, and the reputation of files (programs) from the reliable sources should be higher than the reputation of those from suspicious sources. As such, we define the reputation of a system entity to be $[0.0, 1.0]$, where a file (program) from trusted sources should have a reputation closer to 1.0, and a file (program) from suspicious sources should have a reputation closer to 0.0.

Seed Sources and System Libraries. Seed nodes are the nodes without incoming edges in the dependency graph. They are usually trusted sources such as official updates like Microsoft updater or Chrome updater (assigned high reputations), system libraries like libc (assigned neutral reputations), or suspicious sources like USB sticks or malicious websites (assigned with low reputations). For each trusted seed, its initial reputation is set to 1.0; for each suspicious seed, its initial reputation is set to 0.0. Additionally, since the system libraries will be used by both legitimate users and attackers, we cannot easily infer a node's nature from its correlations with the system libraries. Thus, the initial reputations of seed nodes representing system packages are set to 0.5.

Reputation Inheritance. Based on the edge weights and the chosen seeds, we iteratively update other nodes' reputation values. To prevent fast degradation of reputation values, SYSREP updates a node's reputation using an inheritance fashion:

$$R_v = \sum_{e \in IncomingEdge(v)} R_{sourceNode(e)} * W_e, \tag{8}$$

where $IncomingEdge(v)$ represents the set of incoming edges of v, $sourceNode(e)$ represents the source node of e, $R_{sourceNode(e)}$ represents the reputation of $sourceNode(e)$, and W_e represents the weight of e.

Algorithm 3: Reputation Propagation

```
   Input: Weighted Dependency Graph, G_wd
   Output: Weighted Dependency Graph, G_wd
1  while δ > thresold do
2      diff ← 0.0
3      for ∀v ∈ G do
4          if v is seed then
5              continue
6          end
7          Set ← G.incomingEdgeOf(v)
8          for ∀e ∈ Set do
9              s ← e.source
10             res += s.reputation * e.weight
11         end
12         rep ← res
13         diff += |v.reputation − rep|
14         v.reputation ← rep
15         δ ← diff
16     end
17 end
```

Algorithm 3 gives the details of reputation propagation. A node v in the dependency graph receives its reputation by inheriting the weighted reputations from all of its parent nodes (Lines 7-11). Note that if we adopt a distribution fashion that ensures the sum of reputations for v's children nodes to be equal to v's reputation, then the reputation will degrade quickly in a few hops, which does not work well for dependency graphs that often have many paths with many hops. The process of reputation propagation is an iterative process. For each iteration, the algorithm computes the sum of reputation differences for all the nodes (Line 13). The propagation terminates when the aggregate difference between the current

iteration and the previous iteration is smaller than a threshold δ (Line 1), indicating that the reputations for all nodes become stable. Empirically, we set $\delta = 1e-13$. Note that the reputations of the seed nodes remain unchanged (Lines 4-6).

Attack Sequence Reconstruction. SYSREP leverages the weights in the dependency graph for reconstructing the attack sequence, which consists of critical edges and their nodes. Since the weights computed by SYSREP aim to maximize the difference between critical and non-critical edges, we can specify a minimum weight as a threshold and hide the edges with weights below that threshold. For example, the weights of edges from the irrelevant system activities (*e.g.,* daily tasks or file indexing) usually have lower weights than those representing the attack provenance. By carefully choosing this threshold, SYSREP can filter out these irrelevant system activities while retaining the attack provenance (Section 5.2.2).

In practice, security analysts can first hide the non-critical edges using a higher threshold to focus on the investigation of the attack, and gradually show part of the non-critical edges by lowering the threshold to get more context of the attack-related activities. For example, certain system libraries may reveal the functionality that the malicious payloads possess.

5 Evaluation

We build SYSREP upon Sysdig [53], and deployed our tool in a server to collect system auditing events and perform attack investigation. The server is used by other users for performing daily tasks, so that enough noise of irrelevant system activities can be collected. We performed a series of attacks based on known exploits in the deployed environment, and applied SYSREP to perform attack investigation on the attacks, demonstrating the effectiveness of SYSREP. In total, our evaluations use real system monitoring data that consists of *2 billion* events. Each attack is done with the time gap being at least 1 hour.

Specifically, we conduct three sets of evaluations. First, to evaluate the effectiveness of SYSREP in propagating reputations, we compare the reputation scores of the POI entities against the expected reputation scores in benign scenarios (POI entities coming from trusted sources) and attack scenarios (POI entities coming from suspicious sources). We also compare the results with three other weight computation approaches. Second, we compare the weight computation in SYSREP with the edge prioritization of the state-of-the-art causality analysis, PrioTracker [38], which prioritizes edges during dependency search based on the fanout of nodes. Finally, we evaluate the effectiveness of SYSREP in revealing critical edges for attack sequence reconstruction.

5.1 Overall Evaluation Setup

The evaluations are conducted on a server with an Intel(R) Xeon(R) CPU E5-2637 v4 (3.50GHz), 256GB RAM running 64bit Ubuntu 18.04.1. We performed 8 tasks to inject benign and malicious payloads into the system through key system interfaces that are vulnerable for attacks. We also performed 5 real APT attacks in the deployed environment. We then collected the system auditing events and applied SYSREP to analyze the events.

Table 4: POI reputations of benign payloads through key system interfaces (expected 1.0**)**

Attack/Task	LP-FIXED	PrioTracker	LP-GLOBAL	LP-GLOBAL+	SYSREP
3File	0.85	0.50	0.50	0.98	1.00
2File	0.80	0.50	0.50	0.50	1.00
curl	0.70	0.60	0.51	0.51	0.99
shell_script	0.62	0.50	0.50	0.57	0.96
python_wget	0.71	0.65	0.61	0.63	0.99
scp	0.74	1.00	0.93	0.92	1.00
shell_wget	0.80	0.75	0.82	0.89	0.98
wget	0.71	0.54	0.50	0.50	1.00
avg	0.74	0.63	0.61	0.69	0.99

Table 5: POI reputations of malicious payloads through key system interfaces (expected: 0.0**)**

Attack/Task	LP-FIXED	PrioTracker	LP-GLOBAL	LP-GLOBAL+	SYSREP
3File	0.15	0.50	0.49	0.02	∼0.00
2File	0.20	0.50	0.50	0.50	∼0.00
curl	0.30	0.40	0.49	0.49	0.01
shell_script	0.38	0.50	0.50	0.50	0.04
python_wget	0.29	0.35	0.39	0.37	0.01
scp	0.26	∼0.00	0.07	0.08	∼0.00
shell_wget	0.20	0.25	0.18	0.11	0.02
wget	0.21	0.46	0.50	0.50	∼0.00
avg	0.25	0.37	0.39	0.32	0.01

5.1.1 Benign and Malicious Payloads Through Key System Interfaces

We performed 8 tasks that employ the common system interfaces to inject benign or malicious payloads. These representative system interfaces are commonly exploited in attacks [14].

- File merge: *2File*, *3File*
- Shell execution: *shell-script* (list all files in the Home folder and write the results to a file)
- File download: *curl*, *wget*, *shell-wget* (wget called by a shell script), *python-wget* (wget called by a Python script)
- File transfer: *scp*

5.1.2 APT Attacks

We performed five APT attacks that capture the important traits of APT attacks depicted from the Cyber Kill Chain framework [4]. Note that an APT attack consists of a series of steps, and some steps may not be captured by system monitoring (*e.g.,* user inputs and inter-process communications). Such limitations can be addressed by employing more powerful auditing tools, but it is out of the scope of this paper. Thus, we identified 10 key steps that are related to POI entities for our evaluations in the five APT attacks.

APT Attack 1: Zero-Day Penetration to Target Host. The scenario emulates the attacker's behavior who penetrates the victim's host leveraging previously unknown Zero-day attack. Zero-day vulnerabilities are attack vectors that previously unknown to the community, therefore allow the attacker to put their first step into their targets. In our case, we assume that the `bash` binary in victim's host is outdated and vulnerable to shellshock [1]. The victim computer hosts web service

that has CGI written as BASH script. The attacker can run an arbitrary command when she passes the specially crafted attack string as one of environment variable. Leveraging the vulnerability, the attacker runs a series of remote commands to plant and run initial attack by: (1) transferring the payload (*penetration-c1*), (2) changing its permission, and (3) running the payload to bootstrap its campaign (*penetration-c2*).

APT Attack 2: Password Cracking After Shellshock Penetration. After initial shellshock penetration, the attacker first connects to Cloud services (*e.g.,* Dropbox, Twitter) and downloads an image where C2 (Command and Control) host's IP address is encoded in EXIF metadata (*password-crack-c1*). The behavior is a common practice shared by APT attacks [9, 21] to evade the network-based detection system based on DNS blacklisting.

Using the IP, The malware connects to C2 host. C2 host directs the malware to take some lateral movements, including a series of stealthy reconnaissance maneuvers. In this stage, the attacker generally takes a number of actions. Among those, we emulate the password cracking attack. The attacker downloads password cracker payload (*password-crack-c2*) and runs it against password shadow files (*password-crack-c3*).

APT Attack 3: Data Leakage After Shellshock Penetration. After lateral movement stage, the attacker attempts to steal all the valuable assets from the host. This stage mainly involves the behaviors of local and remote file system scanning activity, copying and compressing of important files, and transferring to its C2 host. The attacker scans the file system, scrap files into a single compress file and transfer it back to C2 host (*data-leakage*).

APT Attack 4: Command-line Injection with Input Sanitization Failures. Different from the previous shellshock case, a program may contain vulnerabilities introduced by developer errors and this can also be a initial attack vector that invites the attacker into their target systems. To represent such cases, we wrote an web application prototype that fails to sanitize inputs for a certain web request, hence allows Command line Injection attack. Our prototype service mimics the Jeep-Cherokee attack case [44] which implements a remote access using the conventional web service API that internally uses DBUS service to run the designated commands. Due to the developer mistake, the web service fails to sanitize the remote inputs, the attacker can append arbitrary commands followed by semi-colon(`;`). Leveraging this vulnerability, we can download backdoor program (*commend-injection-c1*) and collect sensitive data (*command-injection-c2*).

APT Attack 5: VPNFilter. We prototyped a famous IoT attack campaign; VPNFilter malware [8], which infected millions of dozens of different IoT devices exploiting a number of known or zero-day vulnerabilities [2, 3]. The attack's significance lies in how the malware operates during its lateral movement stage following its initial penetration. The campaign employs up-to-date hacker practices to bypass conventional security solutions based on static blacklisting approaches and has an architecture to download plug-in payload on-demand, at run-time. We prototyped the malware referring to one of its sample for x86 architecture [50].

The VPNFilter stage 1 malware accesses a public image repository to get an image. In the EXIF metadata of the image, it contains the IP address for the stage 2 host (*vpnfilter-c1*). It downloads the VPNFilter stage2 from the stage2 server, and runs it (*vpnfilter-c2*).

Table 6: POI reputations of APT attacks (expected: 0.0)

Attack	LP-FIXED	PrioTracker	LP-GLOBAL	LP-GLOBAL+	SYSREP
data-leakage	0.16	∼0.00	0.16	0.15	∼0.00
password-crack-c1	0.04	0.25	0.03	∼0.00	∼0.00
password-crack-c2	0.40	∼0.00	0.50	∼0.00	0.01
password-crack-c3	0.04	∼0.00	∼0.00	0.02	∼0.00
penetration-c1	0.13	0.30	0.43	0.03	0.03
penetration-c2	0.13	0.43	0.13	0.13	0.04
command-injection-c1	0.27	∼0.00	0.49	∼0.00	∼0.00
command-injection-c2	0.30	∼0.00	0.24	0.17	0.04
vpnfilter-c1	0.05	∼0.00	0.02	0.01	0.01
vpnfilter-c2	0.04	∼0.00	∼0.00	∼0.00	∼0.00
avg	0.15	0.10	0.20	0.05	0.01

5.2 Evaluation Results

5.2.1 Reputation Propagation

We evaluate the effectiveness of SYSREP in identifying whether POI entities come from trusted sources or suspicious sources via reputation propagation in both normal and malicious scenarios, respectively.

Evaluation Setup. All the nodes that have no incoming edges are considered as seed nodes. We set the reputation of seed nodes representing trusted sources to 1.0, and seed nodes representing system libraries to 0.5. In malicious scenarios and APT attacks, we set the reputation of seed nodes representing suspicious sources to 0.0. We propagate the reputation from seed nodes according to Algorithm 3, and record the reputation scores of the POI entities (referred to as *POI reputation*). An effective approach will lead to a higher POI reputation in the benign scenarios and a lower POI reputation in the malicious scenarios.

To demonstrate the effectiveness of SYSREP's weight computation, we compare the POI reputations computed by the following four weight computation approaches:

- LP-FIXED: We select a fixed parameter vector $(0.1, 0.5, 0.4)$ empirically and normalize it to be the projection vector.
- LP-GLOBAL: We globally cluster all edges in the graph using Multi-KMeans++ and compute the projection vector using extended LDA.
- LP-GLOBAL+: Same as previous one, but for nodes that have only one incoming edge (*i.e.,* outlier edges), we do not consider these edges in the global clustering and global projection vector computation, and directly assign their final weights to 1.
- SYSREP: This is the one described in Section 4.4. We locally cluster the incoming edges of every node using Multi-KMeans++ and locally compute the projection vector using extended LDA.

Furthermore, we compare our weight computation approach with the event priority computation used in the state-

of-the-art causality analysis approach, PrioTracker [38]. PrioTracker mainly uses the fanout of nodes to prioritize the dependencies in the causality analysis for a given POI event. While they also use a rareness score based on the reference models built upon normal activities in their proprietary environment, their models are not publicly available and difficult to generalize from their organizations to our deployed environment. Given that the rareness score accounts for only a small portion of their priority (*i.e.,* 27%), we use only the fanout of nodes to compute edge priories.[2] We then adapt the computed priories as the edge weights, and apply our algorithm for reputation propagation. In this way, we can do a fair comparison between SysRep and PrioTracker in reputation propagation.

Tables 4 and 5 show the results for benign and malicious payloads through key system interfaces. Table 6 shows the results for the APT attacks.

POI Reputations of SysRep. The results show that SysRep effectively propagate the reputation scores from trusted and suspicious sources to the POI entities (averagely 0.99 for benign scenarios and averagely 0.03 in malicious scenarios). This indicates the effectiveness of our weight computation (Section 4.4) and reputation inheritance (Section 4.5).

Comparisons of Four Weight Computation Approaches. We have the following observations: (1) The performance of LP-Global+ significantly improves over LP-Global. This shows the effectiveness and necessity of treating outlier edges differently when doing weight computation. (2) LP-Fixed performs better than LP-Global and LP-Global+ in system tasks through key system interfaces in both benign and malicious scenarios (Table 4 and Table 5). This shows that the dependency graph is quite diverse and it is difficult to separate all edges into two discriminative groups. However, treating the outlier edges differently in LP-Global+ improves over LP-Fixed for APT attacks. (3) SysRep achieves the best performance in most of the cases. Specifically, SysRep achieves 34.67%, 62.82%, 43.76% improvement over LP-Fixed, LP-Global, and LP-Global+ in benign scenarios (Table 4) and average 94.52%, 95.61%, 86.21% improvement in malicious scenarios (Tables 5 and 6). The results clearly show the necessity and superiority of clustering and projecting edges locally for each sink node. Note that this approach also treats outliers locally by directly setting their weights to 1, and thus SysRep embraces the merits of LP-Global+ and achieves the best performance.

Comparison with PrioTracker. The results show that SysRep achieves 57.22% improvement over PrioTracker in benign scenarios(Table 4) and average 87.22% improvement over PrioTracker in malicious scenarios (Tables 5 and 6). As we can see from Table 6, while the average POI reputation achieved by PrioTracker is 0.10, it achieves bad performance for *penetration-c1* (0.30), *penetration-c2* (0.43), and *password-crack-c1* (0.25), which will incorrectly label the malicious payloads as neutral, while SysRep correctly assigns low POI reputations (≤ 0.04) for these attack steps. These results demonstrate the superiority of SysRep's discriminative weight computation over PrioTracker's priority computation.

[2]Note that the reference models are complementary to SysRep, and SysRep can easily integrate the rareness score with our weights to compute to final edge weights.

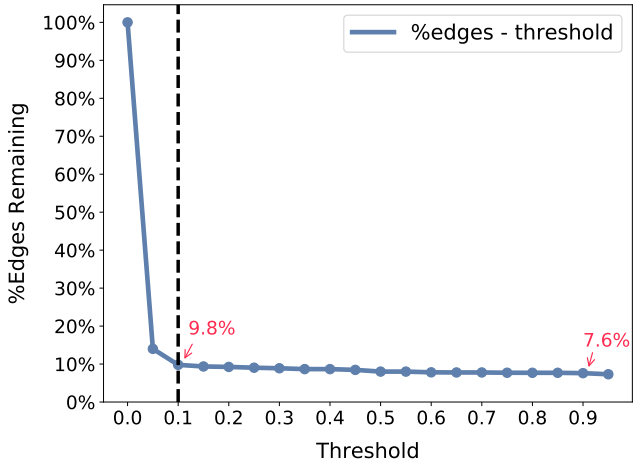

Figure 3: Effectiveness of filtering. The percentage of edges remaining after filtering drops significantly at $T = 0.10$ and remains stable below 10%.

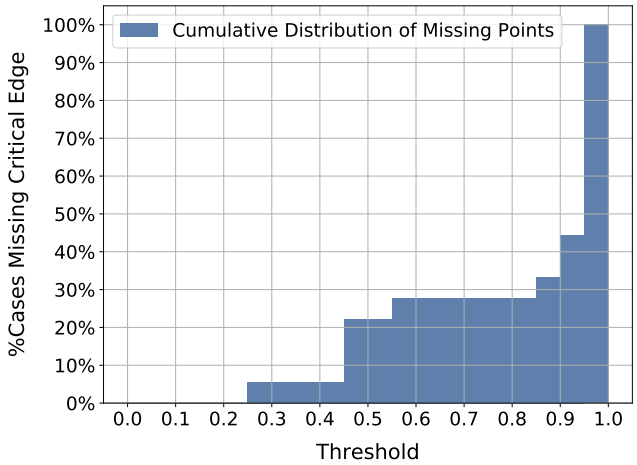

Figure 4: Critical edge loss from filtering. All missing points are distributed between $T = 0.25$ and $T = 1.0$.

5.2.2 Attack Sequence Reconstruction

The important goal of attack sequence reconstruction is to filter out as many irrelevant edges as possible while preserving critical edges in the dependency graph. Based on the edge weights (Section 4.4), SysRep hides non-critical edges whose weights are below a threshold. To provide a guidance on choosing this threshold, we test the filtering performance on all the attacks studied in Section 5.2.1 by using a range of values from 0.05 to 0.95 with a pace of 0.05 as the thresholds.

Figure 3 shows the average percentage of remaining edges after edge filtering. We observe that when the threshold reaches 0.10, the average percentage of remaining edges is

9.8%, and further increasing the threshold from 0.10 to 0.90 only results in a slightly increased amount of pruned edges (2.2%). Such results indicate that most of the edges having reputation scores below 0.10 or above 0.90.

While a higher threshold can hide more irrelevant edges, it may cause the loss of critical edges as well. Thus, we define the *missing point* as the greatest threshold that preserves all the critical edges for a dependency graph, and measure the missing points for all the attacks. Figure 4 shows the cumulative distribution of the missing points for all the attacks. We observe that: (1) all missing points are greater than 0.25, and (2) about 80% of the missing points are greater than 0.90.

Combining the results in Figure 3 and Figure 4, we can conclude that the reputation scores of almost all the critical edges are above 0.90, while the reputation scores of most of the non-critical edges are below 0.10. Such results clearly demonstrate the effectiveness of SYSREP in leveraging the discriminative weights (Section 4.4) to distinguish critical edges from non-critical edges. Furthermore, based on Figure 4, we can suggest an optimal range of the threshold: $[0.1, 0.25]$.

6 Discussion

6.1 Alternative Approaches for Weight Computation

As shown in Section 5.2, the SYSREP achieves the best performance in all the compared weight computation approaches, especially much better than empirically setting a fixed projection vector. As mentioned in Section 4.4, classification-based approaches have very limited generalization capability due to the lack of enough training samples and the highly specialized context introduced by the POI event. Among unsupervised learning approaches, approaches based on anomaly detection [17] might be a substitution for KMeans. In order to compute the best projection vector, we extend the standard LDA from several aspects including handling the singular within-group scatter matrix S_w, negating the projection vector by condition to ensure critical edges have higher projected weights than non-critical edges, and scaling the projected weights to a positive range. We acknowledge that there might be other ways to extend the standard LDA and other methods to achieve discriminative dimensionality reduction [42, 51], which we leave for future exploration.

6.2 Parallelization

Part of the construction of weighted dependency graph can be potentially parallelized with distributed computing. The dependency graph produced by traditional causality analysis [34, 35] can be parallelized by searching the dependency separately. The feature extraction for each edge is independent and can be parallelized. In the scenarios where multiple hosts are involved, dependencies on each host can be precomputed in parallel and thus cross-host causality analysis becomes the concatenation of multiple generated dependency graphs. The weight computation can be parallelized by processing the set of incoming edges of each node separately. However, the reputation propagation cannot be easily separated into independent components for parallel processing (though, we can convert Equation (8) into a matrix-vector product form to save CPU cycles), due to the dependencies on the computation results, and the massive dependencies among system events also pose significant challenges for parallelization. Weighted causality analysis with parallelization is an interesting research direction that requires non-trivial efforts.

6.3 Attacks Against the Reputation System

In practice, the attacker, with some knowledge about the proposed system, may optimize its attack strategy to stay under cover. For instance, (i) to have a lower time weight, the attacker may inject its malicious files earlier but start its attack later, or (ii) to have a lower data weight, the attacker may perform the attack using multiple processes and each process is only associated with small data amount (*i.e.,* the attacker distributes the malicious behavior to multiple processes). An attacker may also choose to gain reputation first before attacking the system or stay under cover and attack probabilistically. In this paper, we do not consider the potential attacks against the reputation system, which we leave for future work.

6.4 Industrial View

Because APT attacks consist of many small steps over a long period of time, even though security experts can obtain the system-wide log, it is time-consuming to manually inspect the daunting number of edges in the system dependency graph, and thus it is hard to discover the complete attack steps and reconstruct the attack sequence. Moreover, depending on the individual's capability, the quality of the analysis may vary a lot. By enabling automatic investigation, SYSREP not only reduces the time consumption of the analysis but also mitigates the dependency on the capability of the security analysts. This makes SYSREP highly applicable to small-scale businesses that are not affordable to hire a large team of security analysts to conduct labor-intensive investigation.

7 Related Work

In this section, we survey three categories of related work.

Weight Computation. Several components of SYSREP are built up on a set of existing techniques. Our local edge clustering step is based on Multi-KMeans++ [12], which optimizes the seed initialization and leads to better clustering quality, compared with the standard KMeans. Our local feature projection step is constructed based on Linear Discriminant Analysis (LDA) [42], which finds a linear combination of features that characterizes or separates multiple classes of objects. Yet, to apply it in our setting, we extend the standard LDA to handle

the challenges including lack of a prior class information, matrix singularity, and limited number of labeled instances.

Reputation Propagation. Our reputation propagation model is inspired by the TrustRank algorithm [28], which is originally designed to separate spam and reputable web pages: it first selects a small set of reputable seed pages, then propagates the trust scores following the link structures using the PageRank algorithm [46], and identifies spam pages as those with low scores. Similar ideas have been applied in security and privacy application scenarios including Sybil detection [15, 23, 27], fake review detection [48], and attribute inference attacks [30, 56]. The model implemented in this work differs in that it propagates the reputation score in an inheritance fashion rather than a distribution fashion, which avoids the serious degradation of reputation scores after propagating on long dependency paths.

Forensic Analysis Using System Audit Logs. Significant progress has been made to leverage system-level audit logs for forensic analysis. Causality analysis based on system monitoring data plays a critical role. King et al. [34] proposed a backward causality analysis technique to perform intrusion analysis by automatically reconstructing a series of events that are dependent on a user-specified event. King et al. further improved the approach to track dependencies cross different hosts [35]. Goel et al. [26] proposed a technique that recovers from an intrusion based on forensic analysis. Further efforts have been made to mitigate the dependency explosion problem by performing fine-grained causality analysis. BEEP [36] identified the event handling loops in programs and enabled selective logging for unit boundaries and unit dependencies through instrumentation. Ma et al. [40] alternated between logging and taint tracking to derive more accurate dependencies. Kwon et al. [32] further leveraged context-sensitive causal models to improve the analysis. Liu et al. [38] proposed to learn a reference model of normal activities to prioritize abnormal activities. While these existing works focus on computing the dependencies for a POI event, they face major limitations in automating the attack investigation since they require non-trivial efforts from the security analysts to inspect the output dependency graphs, which are often quite large. SYSREP makes the first step to address this problem through automatic reputation propagation and attack sequence reconstruction. Nonetheless, SYSREP can be integrated with these works by using their dependency graphs to capture more types of attacks.

Behavior querying leverages domain-specific languages (DSLs) to search the attack patterns of system call events. Gao et al. [25] proposed a domain-specific language that enables efficient attack investigation by querying the historical system audit logs stored in databases. Gao et al. [24] further proposed a language for the streaming settings to query the real-time anomaly patterns. A major limitation of these DSLs is that they require the security analysts to manually construct the behavior queries, which is labor-intensive and error-prone.

To mitigate the burden of storing large amount of system monitoring data, Lee et al. [37] proposed to leverage garbage collection to remove temporary files, Xu et al. [57] proposed to merge dependencies while still preserving high-fidelity causal dependencies, and Tang et al. [54] used templates to summarize the set of system libraries loaded at the process initiation period. SYSREP can be integrated with these works to achieve better scalability for handling cases that span a long period of time (*e.g.,* months).

To reduce the false alarms of threat detection, Hassan et al. [29] proposed to rely on the contextual and historical information of generated threat alert to combat threat alert fatigue. Milajerdi et al. [43] proposed to rely on the correlation of suspicious information flows to detect ongoing attack campaigns. Pasquier et al. [47] provides runtime analysis of provenance by combining runtime kernel-layer reference monitor with a query module mechanism. SYSREP can be interoperated with these runtime systems to achieve a better defense.

8 Conclusion

We propose a novel approach, SYSREP, which assigns discriminative weights to edges in a dependency graph built from system auditing events and propagates reputation scores in the weighted graph from seed sources to POI events. The discriminative edge weights enable SYSREP to reveal critical edges for automatically reconstructing the attack sequence. The reputation propagation paradigm enables SYSREP to automatically determine the suspiciousness/trustworthiness of POI entities based on whether the system entities originate from suspicious sources or trusted sources. The synergy of attack sequence reconstruction and reputation propagation makes SYSREP the first step towards automatic attack investigation. The evaluations on system tasks that inject benign and malicious payloads through key system interfaces and real APT attacks demonstrate that (1) SYSREP effectively propagates reputation since POI entities receive expected reputation scores in both benign and malicious scenarios; (2) SYSREP effectively reveals critical edges since the computed weights for the critical edges are well separated from the computed weights for the non-critical edges. These results make SYSREP the first step towards automatic attack investigation.

References

[1] CVE-2014-6271: bash: specially-crafted environment variables can be used to inject shell commands. `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271`.

[2] CVE-2017-6334: WEB Netgear NETGEAR DGN2200 dnslookup.cgi Remote Command Injection. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-6334.

[3] CVE-2018-7445: NETBIOS MikroTik RouterOS SMB Buffer Overflow. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-7445.

[4] cyberkillchain. https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html.

[5] The Equifax data breach. https://www.ftc.gov/equifax-data-breach.

[6] The Marriott data breach. https://www.consumer.ftc.gov/blog/2018/12/marriott-data-breach.

[7] Yahoo discloses hack of 1 billion accounts, 2016. https://techcrunch.com/2016/12/14/yahoo-discloses-hack-of-1-billion-accounts/.

[8] Schneier on Security: Router Vulnerability and the VPNFilter Botnet. https://www.schneier.com/blog/archives/2018/06/router_vulnerab.html, 2018.

[9] VPNFilter: New Router Malware with Destructive Capabilities. https://symc.ly/2IPGGVE, 2018.

[10] Arthur Albert. *Regression and the Moore-Penrose pseudoinverse*. Elsevier, 1972.

[11] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2009.

[12] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, 2007.

[13] Adam M. Bates, Dave Tian, Kevin R. B. Butler, and Thomas Moyer. Trustworthy whole-system provenance for the linux kernel. In *USENIX Security*, 2015.

[14] Matt Bishop. *Introduction to Computer Security*. Addison-Wesley Professional, 2004.

[15] Qiang C., Michael S., Xiaowei Y., and Tiago P. Aiding the detection of fake accounts in large scale social online services. USENIX, 2012.

[16] Bryan Cantrill, Adam Leventhal, and Brendan Gregg. DTrace, 2017. http://dtrace.org/.

[17] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009.

[18] cnn. OPM government data breach impacted 21.5 million, 2015. http://www.cnn.com/2015/07/09/politics/office-of-personnel-management-data-breach-20-million.

[19] Ebay. Ebay Inc. to ask Ebay users to change passwords, 2014. http://blog.ebay.com/ebay-inc-ask-ebay-users-change-passwords/.

[20] Fireeye. Anatomy of advanced persistent threats. 2017.

[21] FireEye Inc. HammerToss: Stealthy Tactics Define a Russian Cyber Threat Group. Technical report, FireEye Inc., 2015.

[22] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, NY, USA:, 2001.

[23] Peng Gao, Binghui Wang, Neil Zhenqiang Gong, Sanjeev R. Kulkarni, Kurt Thomas, and Prateek Mittal. Sybilfuse: Combining local attributes with global structure to perform robust sybil detection. In *CNS*, 2018.

[24] Peng Gao, Xusheng Xiao, Ding Li, Zhichun Li, Kangkook Jee, Zhenyu Wu, Chung Hwan Kim, Sanjeev R. Kulkarni, and Prateek Mittal. SAQL: A stream-based query system for real-time abnormal system behavior detection. In *USENIX Security*, 2018.

[25] Peng Gao, Xusheng Xiao, Zhichun Li, Fengyuan Xu, Sanjeev R. Kulkarni, and Prateek Mittal. AIQL: Enabling efficient attack investigation from system monitoring data. In *USENIX ATC*, 2018.

[26] Ashvin Goel, Kenneth Po, Kamran Farhadi, Zheng Li, and Eyal de Lara. The taser intrusion recovery system. In *SOSP*, 2005.

[27] Neil Zhenqiang Gong and Di Wang. On the security of trustee-based social authentications. *Trans. Info. For. Sec.*, 9(8), 2014.

[28] Zoltán Gyöngyi, Hector Garcia-Molina, and Jan Pedersen. Combating Web Spam with Trustrank. In *Proceedings of the International Conference on Very Large Data Bases*, 2004.

[29] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. Nodoze: Combatting threat alert fatigue with automated provenance triage. 2019.

[30] Jinyuan Jia, Binghui Wang, Le Zhang, and Neil Zhenqiang Gong. Attriinfer: Inferring user attributes in online social networks using markov random fields. In *WWW*, 2017.

[31] Xuxian Jiang, AAron Walters, Dongyan Xu, Eugene H. Spafford, Florian Buchholz, and Yi-Min Wang. Provenance-aware tracing of worm break-in and contaminations: A process coloring approach. In *ICDCS*, 2006.

[32] Yonghwi K., Fei W., Weihang W., Kyu H. L., Wen-C. L., Shiqing M., Xiangyu Z., Dongyan X., Somesh J., Gabriela F. C., Ashish G., and Vinod Y. MCI : Modeling-based causality inference in audit logging for attack investigation. In *NDSS*, 2018.

[33] Taesoo Kim, Xi Wang, Nickolai Zeldovich, and M. Frans Kaashoek. Intrusion recovery using selective re-execution. In *OSDI*, 2010.

[34] Samuel T. King and Peter M. Chen. Backtracking intrusions. In *SOSP*, 2003.

[35] Samuel T. King, Zhuoqing Morley Mao, Dominic G. Lucchetti, and Peter M. Chen. Enriching intrusion alerts through multi-host causality. In *NDSS*, 2005.

[36] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. High accuracy attack provenance via binary-based execution partition. In *NDSS*, 2013.

[37] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. Loggc: garbage collecting audit log. In *CCS*, 2013.

[38] Yushan Liu, Mu Zhang, Ding Li, Kangkook Jee, Zhichun Li, Zhenyu Wu, Junghwan Rhee, and Prateek Mittal. Towards a timely causality analysis for enterprise security. In *NDSS*, 2018.

[39] Shiqing Ma, Juan Zhai, Yonghwi Kwon, Kyu Hyung Lee, Xiangyu Zhang, Gabriela F. Ciocarlie, Ashish Gehani, Vinod Yegneswaran, Dongyan Xu, and Somesh Jha. Kernel-supported cost-effective audit logging for causality tracking. In *USENIX ATC*, 2018.

[40] Shiqing Ma, Xiangyu Zhang, and Dongyan Xu. Protracer: towards practical provenance tracing by alternating between logging and tainting. 2016.

[41] Microsoft. ETW events in the common language runtime, 2017. https://msdn.microsoft.com/en-us/library/ff357719(v=vs.110).aspx.

[42] Sebastian Mika, Gunnar Ratsch, Jason Weston, Bernhard Scholkopf, and Klaus-Robert Muller. Fisher discriminant analysis with kernels, 1999.

[43] Sadegh M Milajerdi, Rigel Gjomemo, Birhanu Eshete, R Sekar, and VN Venkatakrishnan. HOLMES: real-time APT detection through correlation of suspicious information flows. 2019.

[44] Charlie Miller and Chris Valasek. Remote Exploitation of an Unaltered Passenger Vehicle. *Black Hat USA*, 2015:91, 2015.

[45] NPR. Home Depot Confirms Data Breach At U.S., Canadian Stores, 2014. http://www.npr.org/2014/09/09/347007380/home-depot-confirms-data-breach-at-u-s-canadian-stores.

[46] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report 1999-66, 1999.

[47] Thomas Pasquier, Xueyuan Han, Thomas Moyer, Adam Bates, Olivier Hermant, David Eyers, Jean Bacon, and Margo Seltzer. Runtime Analysis of Whole-system Provenance. In *ACM CCS*. ACM, 2018.

[48] Shebuti Rayana and Leman Akoglu. Collective opinion spam detection: Bridging review networks and metadata. In *KDD*, 2015.

[49] Redhat. The linux audit framework, 2017. https://github.com/linux-audit/.

[50] Sergei Shevchenko. VPNFilter 'botnet': a SophosLabs analysis. *A SophosLabs technical paper*, 2018.

[51] Masashi Sugiyama. Local fisher discriminant analysis for supervised dimensionality reduction. In *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006.

[52] Symantec. Advanced Persistent Threats: How They Work, 2017. https://www.symantec.com/theme.jsp?themeid=apt-infographic-1.

[53] Sysdig. Sysdig, 2017. http://www.sysdig.org/.

[54] Yutao Tang, Ding Li, Zhichun Li, Mu Zhang, Kangkook Jee, Xusheng Xiao, Zhenyu Wu, Junghwan Rhee, Fengyuan Xu, and Qun Li. Nodemerge: Template based efficient data reduction for big-data causality analysis. In *ACM CCS*, 2018.

[55] New York Times. Target data breach incident, 2014. http://www.nytimes.com/2014/02/27/business/target-reports-on-fourth-quarter-earnings.html?_r=1.

[56] Binghui Wang, Jinyuan Jia, and Neil Zhenqiang Gong. Graph-based Security and Privacy Analytics via Collective Classification with Joint Weight Learning and Propagation. 2019.

[57] Zhang Xu, Zhenyu Wu, Zhichun Li, Kangkook Jee, Junghwan Rhee, Xusheng Xiao, Fengyuan Xu, Haining Wang, and Guofei Jiang. High fidelity data reduction for big data security dependency analyses. In *CCS*, 2016.