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

Anonymous Author(s)

ABSTRACT

The need for countering Advanced Persistent Threat (APT) attacks has led to solutions that ubiquitously monitor system activities in each host, and perform timely attack investigation over the monitoring data. However, existing solutions require non-trivial efforts of manual inspection, which are labor-intensive and lack of automation. To bridge the gap, we propose SysRep, a novel approach that facilitates automatic attack investigation via weight-aware reputation propagation from system monitoring. Given a POI to be investigated, SysRep (1) applies causality analysis to construct a system dependency graph for the POI and identify a set of entry nodes, (2) adopts novel methods to compute discriminative weights for edges in the dependency graph, so that critical edges that are important to revealing the attack sequence are easily revealed from non-critical edges, and (3) adopts a novel weight-aware reputation propagation scheme to propagate the reputation from entry nodes (nodes without incoming edges) to the POI along the weighted edges. The POI reputation can be used to identify the root cause nodes among other entry nodes that result in the POI, and determine the trustworthiness/suspiciousness of the POI. The discriminative edge weights can be used to reveal critical edges and reconstruct the attack sequence, which details how the POI was created. Synergistically, SysRep significantly reduces the efforts of manual inspection and facilitates automatic investigation. The evaluation on a wide range of attacks demonstrates the practical efficacy of SysRep.

1 INTRODUCTION

Advanced Persistent Threat (APT) attacks [24, 57] have plagued many well-protected companies, causing significant losses [4, 5, 7–9, 22, 60]. 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 [28, 29, 33, 35, 36, 38, 39, 42, 43]. System monitoring collects system-level auditing events about system calls, and the collected data further enables the security analyst to unfold these attacks to identify the root causes and the ramifications of attacks, which is critical for performing timely system recovery and preventing future compromise.

While the research on attack investigation based on audit logs shows great potential, there has been little work in automating such process. Existing solutions based on causality analysis [30, 37–39, 42] and behavior querying [28, 29] require non-trivial efforts of manual inspection, which are labor-intensive and lack of automation. Causality analysis approaches assume causal relationships between system entities (*e.g.,* files, processes, and network connections) that are involved in the same system call event (*e.g.,* a process reading a file), and based on such assumption, these approaches organize the system call events in a system dependency graph, with nodes being system entities and edges being events. By inspecting such a dependency graph, the security analyst can trace from POI (point-of-interest) entities to identify the root causes and reconstruct the attack sequence. The major limitation of causality analysis, however, is the significant amount of manual efforts in inspecting a daunting number of edges (typically, tens of thousands) due to the dependency explosion problem [40, 59, 62]. Behavior querying approaches [28, 29] leverage domain-specific languages (DSLs) to investigate the attack patterns of system call events. The construction of behavior queries, however, requires that the security analyst masters the syntax of DSLs (typically by going through a steep learning curve) to construct behavior queries and manually inspects the potentially large query results.

Key Insight: Proactively Revealing Critical Edges: In this work, we aim to bridge the gap between the pressing need for automating the attack investigation process and the lack of practical solutions. We have two *key observations*: (1) on a large dependency graph constructed from POI, a small number of *critical edges* that represent the attack sequence (*e.g.,* events that create and execute malicious payloads) are typically buried in many non-critical edges (*e.g.,* events that perform irrelevant system activities); (2) compared to non-critical edges, critical edges typically present a different set of properties (*e.g.,* amount of data flow) that are more correlated with the POI. Unlike existing approaches [30, 37–39, 42] that *reactively* leverage such observations by requiring intensive manual inspection of the large dependency graph to reveal critical edges, we seek to *proactively leverage such observations* by designing an approach to *automate the process of revealing critical edges and associating critical edges with POI entities for POI diagnosis*, which are the two key steps for attack investigation.

Contributions: We propose SysRep, a novel approach that facilitates automatic attack investigation via weight-aware reputation propagation from system monitoring. Given a POI to be investigated, SysRep first applies causality analysis to construct a dependency graph for the POI and identify a set of entry nodes (*i.e.,* nodes on the dependency graph without incoming edges). Next, to automatically reveal critical edges from non-critical edges, SysRep employs novel methods to compute discriminative weights for edges, so that critical edges and non-critical edges can be easily distinguished using their weight scores. The higher weights the more impact on the POI. Finally, to automatically associate entry nodes with the POI through critical edges, SysRep employs a novel weight-aware reputation propagation scheme that propagates reputation from entry nodes (the reputation is assigned by the security analysts) along the weighted edges to infer POI reputation, which characterizes the impact of the entry nodes to the POI. Specifically,

Anonymous Submission to ACM CCS 2019, Due 15 May 2019, London, TBD
2019. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

for entry nodes that represent files, processes, or IPs, the security analyst assigns the reputation based on the domain knowledge about the entities (as these entry nodes have no incoming edges for receiving reputation): high reputation values for trusted sources (*e.g.,* verified binaries), low reputation values for untrusted sources (*e.g.,* unknown IPs and vulnerable binaries), and neutral reputation values for neutral sources like system libraries since they may be used by both benign and malicious applications. Note that this process can be largely automated by compiling a list of verified binaries and libraries or leverage existing reputation systems [21, 53].

The POI reputation can be used to (1) identify the root cause nodes among other entry nodes that result in the POI by comparing the POI reputation with the entry node reputations, and (2) determine the trustworthiness/suspiciousness of the POI based on whether the root cause nodes are trusted sources or untrusted sources. The node reputation and edge weights can be used seamlessly to reconstruct the attack sequence, which details how the POI was created. Synergistically, SysRep significantly reduces the efforts of manual inspection and facilitates automatic attack investigation.

Challenges: The two key steps to enable SysRep are (1) computing effective edge weights to reveal critical edges, and (2) performing effective weight-aware reputation propagation to infer POI reputation. However, there are two major challenges:

(1) Large Number of Non-Critical Edges and Diversified Attack Scenarios: Due to the large number of non-critical edges, revealing critical edges essentially is the problem of "finding a needle in a haystack". Furthermore, critical edges in different attack scenarios typically present different properties, and thus relying on a single feature (*e.g.,* node degree) to compute edge weights is often insufficient to oversee a diversified set of attack scenarios. As such, how to extract good features that capture the fundamental differences between critical edges and non-critical edges and how to combine these features into a discriminative weight become a key challenge.

(2) Long Dependency Paths: Due to the multi-stage nature of APT attacks [24, 57], the dependency paths from entry nodes to the POI often have many hops. If a node propagates its reputation in a distribution fashion (*i.e.,* distributes its reputation along outgoing edges in portions) [17, 49], the reputation will degrade rapidly in the middle of the path, failing to characterize the impact of entry nodes on POI. As such, how to design a good weight-aware reputation propagation scheme that prevents the reputation degradation on long paths becomes another key challenge.

Novel Techniques of SysRep: To address the aforementioned challenges, SysRep employs two novel techniques:

(1) Discriminative Feature Projection: To compute edge weights, instead of relying on a single "golden" feature, SysRep extracts three discriminative features from every event edge (*i.e.,* relative data amount difference, relative time difference, and concentration degree) that capture the properties of critical edges from three different aspects (*i.e.,* data flow, time, and structure) (Section 4.3).

To compute a single discriminative weight score from the three features, SysRep clusters the incoming edges of each node into two groups, *i.e.,* one group for critical edges and other group for non-critical edges based on our insight. Then, SysRep employs a novel *discriminative feature projection* scheme based on Linear Discriminant Analysis (LDA) [46] to compute an optimal projection vector, so that the projected values of critical edges and the projected values of non-critical edges are maximally separated. The normalized projected values will then serve as as edge weights (Section 4.4).

(2) Inheritance-Based Reputation Propagation: To prevent the rapid degradation of reputation on long dependency paths, SysRep propagates reputation in an *inheritance fashion* (Section 4.5): the reputation of a source node gets inherited to all its sink nodes rectified by the edge weights. Such propagation scheme ensures that the reputation from root cause nodes through the critical edges can be preserved even when propagated along long paths.

Evaluation: We implemented and deployed SysRep (~8000 lines of code) 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 attacks in the deployed environment, and applied SysRep to perform 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 ~2 billion system auditing events.

For evaluation purposes, we set the range of edge weight to be $(0.0, 1.0]$, and the range of the reputation score to be $[0.0, 1.0]$, with 0.0 for untrusted sources, 1.0 for trusted sources, and 0.5 for neutral sources like system libraries. Our evaluation results show that SysRep is effective in revealing critical edges (average weight 0.89) from non-critical edges (average weight 0.06) for all cases. To evaluate the effectiveness of reputation propagation, we measure the reputation scores of POI, which are expected to be close to 1.0 for benign scenarios and 0.0 for malicious scenarios. The results show that SysRep accurately propagates the reputation scores from trusted and untrusted sources to the POI entities (averagely 0.99 for benign scenarios and averagely 0.03 in malicious scenarios).

We also perturb the reputation scores for entry nodes using a uniform random sampling from $[0.2, 0.8]$ for system libraries, $[0.0, 0.2]$ for untrusted sources, and $[0.8, 1.0]$ for trusted sources to evaluate the resilience of SysRep against attacks the on reputation assignment. The results show that the edge weights of SysRep prevent the perturbed reputation of system libraries from adversely impacting the POI, and the POI reputation is always strongly correlated to the reputation of root cause nodes.

Our performance evaluations show that SysRep can finish the analysis for a real attack case within two minutes, where the log parsing (accounting for 75% of the execution time) can be easily optimized by adopting cache or databases [29].

2 BACKGROUND AND MOTIVATION

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 [28–30, 33, 35, 36, 38, 39, 42, 43], 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

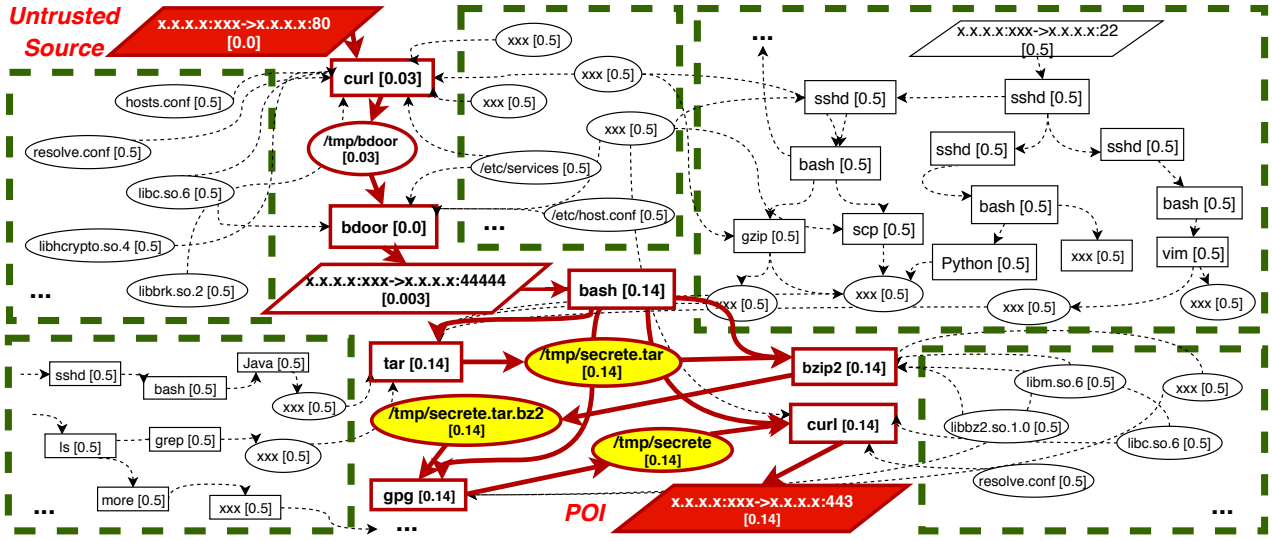


Figure 1: Partial dependency graph of a motivating data leakage attack case (rectangles for processes, ovals for files, parallelograms for network connections, yellow ovals for files leaked in this attack). The complete dependency graph constructed from the POI (red parallelogram at the bottom) via backward causality analysis contains 106 nodes and 237 edges. The untrusted network connection (red parallelogram at the top left) is the root cause node of the POI. Critical edges that represent the attack sequence are represented by solid red arrows, and nodes on the attack sequence have red frames. Non-critical edges are represented by dashed arrows, and non-critical nodes such as normal activities and system libraries are organized in dashed green rectangles. As we can see, attack investigation is a process of finding a needle in a haystack. SysRep automates this process by computing discriminative weights to reveal critical edges and propagating reputation from entry nodes to infer the POI reputation, which can be used to identify the root cause nodes, determine the POI trustworthiness/suspiciousness, and reconstruct the attack sequence, synergistically.

system entities as *files*, *processes*, and *network connections*. We consider a *system event* as the interaction between two system entities represented as ⟨*subject, operation, object*⟩. 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). Representative attributes of entities include file name, process executable name, IP, and port. Representative attributes of events include event origins (*e.g.,* start time/end time) and operations (*e.g.,* file read/write).

2.2 Causality Analysis

Causality analysis [30, 37–39, 42] analyzes the auditing events to infer their dependencies 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 connection. 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 direction of data flow. Each edge is associated with a time window, $tw(e)$. We use $ts(e)$ and $te(e)$ to represent the start time and the end time of e. Formally, in the dependency graph, for two events $e_1(u_1, v_1)$ and $e_2(u_2, v_2)$, there exists causal dependency between e_1 and e_2 if $v_1 = u_2$ and $ts(e_1) < te(e_2)$.

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

Causality analysis enables two important security applications: (i) *backward causality analysis* that identifies the entry points of attacks, and (ii) *forward causality analysis* that investigates 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 vulnerability 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. Given a POI entity which is the network connection to a suspicious remote host (*i.e.,* the red parallelogram at the bottom),

the dependency graph (Figure 1) constructed from the POI contains 106 nodes and 237 edges. The root cause node of the POI is an untrusted network connection represented by a red parallelogram at the top left corner. The critical edges that represent the attack sequence are represented by red arrows, and the involved nodes are in red frames. The goal of attack investigation is to inspect the dependency graph to identify the root cause node of the POI, reveal the attack edges, and reconstruct the attack sequence.

Challenges: As observed in Figure 1, attack investigation is a process of *finding a needle in a haystack*: a limited number of critical edges (*i.e.,* 12) are buried in many non-critical edges (*i.e.,* 225; *e.g.,* normal system library file accesses), and the root cause node (*i.e.,* 1) is buried in many other entry nodes (*i.e.,* 80; *e.g.,* ssh logging in). Existing approaches [30, 37–39, 42] require intensive efforts of manually inspecting these edges and nodes for revealing critical edges, identifying root cause nodes, and reconstructing the attack sequence. As such, how to automate this process for effectively finding a needle in a haystack becomes a key challenge.

Using SysRep for Automatic Attack Investigation: SysRep automates the attack investigation process by (1) computing discriminative weights for edges to reveal critical edges from non-critical edges, and (2) propagating reputation from entry nodes along the weighted edges to infer the POI reputation.

(1) Revealing Critical Edges: In Figure 1, the average weight of critical edges is 0.77, much higher than the average weight of non-critical edges (*i.e.,* 0.07). As such, critical edges can be easily revealed from non-critical edges via setting a threshold (Section 4.5).

(2) Identifying Root Cause Nodes & Determining POI Trustworthiness/Suspiciousness: For demonstration purposes, we set the reputation of the untrusted network connection source to 0.0 and the reputation to the other entry nodes such as neutral system libraries to 0.5. Note that the security analyst has the flexibility to adopt other reputation assignment schemes for entry nodes based on the domain knowledge or external reputation sources (Section 4.5). The POI reputation after propagation is 0.14, much closer to the reputation of the untrusted network source rather than the system libraries. As such, we conclude that the untrusted network source is the root cause node that has a much higher impact on the POI. Furthermore, since both the POI reputation and the reputation of its root cause node are low, we conclude that the POI is also untrusted.

(3) Reconstructing Attack Sequence: With effective edge weights, critical edges can be easily revealed from non-critical edges. Furthermore, with node reputation, critical nodes (*i.e.,* nodes on critical edges) can also be clearly revealed from non-critical nodes. In Figure 1, the average reputation of critical nodes is 0.09, much closer to the reputation of the root cause node (0.0) than the average reputation of non-critical nodes (0.45). By setting proper thresholds (Section 4.5), root cause node, critical nodes, critical edges, and POI entity can be connected together to reconstruct the attack sequence (*i.e.,* the dependency path in red shown in Figure 1), which details how the POI was created from the root cause node.

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 [52], ETW [45], DTrace [18], and Sysdig [58]) to collect system-level audit logs about system calls. Given a POI event, SysRep parses the collected logs and performs causality analysis [38, 39] 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 compute discriminative weights for edges so that critical edges can be easily revealed from non-critical edges, for each edge, SysRep extracts three novel features that model the impact of the event edge on the POI event from three aspects: time aspect, data amount aspect, and structure aspect (Section 4.3). SysRep then employs a novel *discriminative feature projection scheme* to combine the three features into a single discriminative weight (Section 4.4). In Phase III, SysRep propagates reputation from entry nodes to the POI along the weighted edges to infer the POI reputation. The POI reputation will be used to identify the root cause nodes and determine the POI trustworthiness/suspiciousness. Furthermore, SysRep provides a suggested range of threshold values based on the edge weights, which will be used to reconstruct the attack sequence.

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

Threat Model: Our threat model follows the threat model of previous work on system monitoring [15, 28, 29, 33, 38, 39, 41, 42]. We assume that the system monitoring data collected from kernel space [18, 45, 52, 58] 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 DESIGN OF SYSREP

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

4.1 Phase I: Dependency Graph Generation

System Auditing: SysRep leverages system auditing tools [18, 45, 52, 58] 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) process 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 [28, 29, 42]. Thus, we use 5-tuple ($\langle$*srcip, srcport, dstip, dstport, protocol*$\rangle$) as a network connection's unique identifier. Failing to

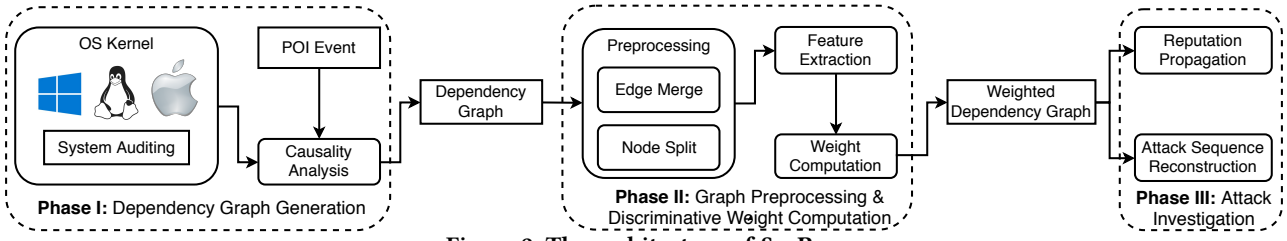


Figure 2: The architecture of SysRep

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, SysRep applies causality analysis (Section 2.2) to produce a dependency graph G_d for the POI. Causality analysis adds the POI to a queue, 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 until the queue is empty. The output is a dependency graph that only contains relevant system entities and events with respect to the POI.

4.2 Phase II: Graph Preprocessing

SysRep preprocesses the dependency graph and transforms it to a simple directed graph without parallel edges.

Edge Merge: The dependency graph produced by causality analysis often has many edges between the same pair of nodes [62]. 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 proposed Causality Preserved Reduction (CPR) [62] for dependency graph reduction, SysRep merges the edges between two nodes. As shown in the study [62], 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 node pairs that have parallel edges. For a node pair (u, v) with parallel edges from u to v, SysRep splits u into multiple copies and assigns each copy to one parallel edge, so that each copy only has one outgoing edge to v. The copies of u also inherit all u's incoming edges that have causal dependencies on 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 features that model the impact of the edge on the POI event from three different aspects.

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

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

  Equation (1) gives the *relative data amount difference* feature, where s_e and s_{e_s} represent the data amount associated with the 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 correlated 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 the timestamp values (we use the event end time) of the 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 [59, 62]. These edges are often associated with considerable data amount and occur at various timestamps, and hence using only $f_{D(e)}$ and $f_{T(e)}$ is less effective in revealing 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:

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

  Here, $InDegree(u)$ and $OutDegree(u)$ represent the in-degree and out-degree of the source node u. For entry nodes, since they

Algorithm 1: 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.incomingEdge(v)
    group_1, group_2 ← Multi-KMeans++(Set)
    (ω*_D, ω*_T, ω*_C) ← extendedLDA(group_1, group_2)
    for e ∈ Set do
        W_eUN = ω*_D f_D(e) + ω*_T f_T(e) + ω*_C f_C(e)
    W' ← Σ_{e∈Set} W_eUN
    for e ∈ Set do
        W_e ← W_eUN / W'
```

are very important in initiating the reputation propagation, we set their concentration degree to be 1.0. This feature helps smooth out the impacts of system libraries with no incoming edges and long-running processes with many outgoing edges.

4.4 Phase II: Weight Computation

One key step of SysRep is to compute discriminative weights for edges to reveal critical edges from non-critical edges. In our design, the edge weight is a real number in $(0, 1]$ that models the aggregated impact of the edge on the POI event. To compute effective edge weights, our key insight is to *emulate the actions of a human analyst when revealing critical edges*: the human analyst traces back from the POI; for each node, the human analyst inspects all its incoming edges and reveals critical edges; the human analyst then switches to the source nodes of the identified critical edges and iterates the last step. As such, for each node, (1) SysRep employs Multi-KMeans++ clustering algorithm [14] to cluster its incoming edges into two groups based on the three features in Section 4.3, where one group consists of critical edges and the other group consists of non-critical edges. dThen, (2) SysRep employs a novel discriminative feature project scheme based on Linear Discriminant Analysis (LDA) [46] to compute an optimal projection vector, so that the projected values of critical edges and the projected values of non-critical edges are maximally separated. The normalized projected values will then serve as edge weights.

Algorithm 1 gives the complete algorithm for computing edge weights. Next, we describe each step in detail.

Step 1: Feature Normalization: Before using the three features, SysRep first normalizes them in the same range [13, 26]. Global normalization for all edges on the graph does not make sense in our context, as (1) a node is only affected by its parents but not by its children or siblings, and (2) according to our previous insight, we seek to cluster all incoming edges locally of each node. As such, 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. In this way, the features of all v's incoming edges are normalized to the same scale (*i.e.,* $[0, 1]$) and are amenable to the clustering in the next step.

Step 2: Edge Clustering: According to our previous insight, for each node, SysRep clusters its incoming edges locally into two groups based on the scaled three features, where one group consists of critical edges and the other group consists of non-critical edges. In the current design, SysRep employs the Multi-KMeans++ clustering algorithm [14]. KMeans clustering aims to partition the observations (points) into k clusters such that each observation belongs to the cluster with the nearest center. 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. We chose $k = 2$ for clustering into two groups. We experimented different values for n and chose $n = 20$ based on our empirical analysis.

Step 3: Discriminative Feature Projection: For each node, SysRep leverages the clustering results in Step 2 and employs a novel discriminative feature projection scheme based on an extended version of Linear Discriminant Analysis (LDA) [46] to compute an optimal projection vector, so that the projected values of two groups are maximally separated.

Formally speaking, for each node v, the feature vectors $\{x\}$ of its N incoming edges are clustered into two groups, g_1 (containing N_1 edges) and g_2 (containing N_2 edges), with group mean vectors: $\mu_1 = \frac{1}{N_1}\sum_{x\in g_1} x$, $\mu_2 = \frac{1}{N_2}\sum_{x\in g_2} x$, $N_1 + N_2 = N$. 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 in discriminant analysis [26] that seeks to reduce the dimensionality of features 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 seeks 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 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 solution to Equation (5) as $\omega^* = (\omega_D^*, \omega_T^*, \omega_C^*)$, for an incoming edge $e(u, v)$ of node v, its (unnormalized) weight $W_{e_{UN}}$ is computed as:

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

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 maximally 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.

Recognizing such limitations, we *extend the standard LDA* from the following two aspects.

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

- $S_w^+(\mu_1 - \mu_2)$, where S_w^+ is the Moore-Penrose [12] inverse of S_w.
- $\mu_1 - \mu_2$ (*i.e.,* the direction of group mean difference)

(b) Correcting the projection vector direction: We correct the direction of the projection vector (*i.e.,* negate), if critical edges have lower projected values than non-critical edges. 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 this problem using a set of heuristics:

I If all three dimensions of the projection vector are non-positive, negate.

II Else if all three dimensions of the projection vector are non-negative, do not negate.

III Else, if the group with a smaller size has a smaller 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.

After correcting the direction of the projection vector, we apply it to the feature vectors to compute the projected weights. Note that sometimes, the projected weights may contain non-positive values. To be amenable to the next step, in such condition, we shift all the projected weights to make them positive.

Step 4: Edge Weight Normalization: As a final step, same as how we normalize the features, for an edge $e(u, v)$, we *locally normalize* its projected weight by the sum of corresponding weights of all incoming edges of the sink node v:

$$W_e = W_{e_{UN}} / \sum_{e' \in incomingEdge(v)} W_{e'_{UN}} \tag{7}$$

This step ensures that for any node except entry nodes (*i.e.,* no incoming edges), the weights of all its incoming edges are in the range $(0, 1]$ and the sum of weights is equal to 1. Note that for nodes that only have one incoming edge, we skip the Steps 1-4 and directly set their edge weights to 1. Integrated in the reputation propagation scheme in Section 4.5, these normalized edge weights ensure that the reputation of a sink node never exceeds the maximum reputation of all its source nodes. This completes the Phase II by producing a weighted dependency graph with discriminative edge weights for revealing critical edges from non-critical edges.

4.5 Phase III: Attack Investigation

Reputation Propagation: To perform reputation propagation, SysRep assigns initial reputations to entry nodes and propagates the reputation in 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 a suspicious equipment, and the reputation of files (programs) from the reliable sources should be higher than the reputation of those from untrusted sources. As such, we define the reputation of a system entity to be $[0.0, 1.0]$, where a file (program) originated from trusted sources should have a reputation closer to 1.0, a file (program) originated from untrusted sources should have a reputation closer to 0.0. In the design of SysRep, the initial reputation of entry nodes is assigned by the security analyst based on the domain knowledge about the entities (as these nodes have no incoming edges for receiving reputation): high reputation for trusted sources, low reputation for untrusted sources, and neutral reputation (*i.e.,* 0.5) for neutral sources like system libraries. Note that this process can be largely automated by compiling a list of verified libraries and binaries or leveraging existing reputation systems [21, 53].

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 2: 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 entry node then
5              continue
6          Set ← G.incomingEdge(v)
7          for ∀e ∈ Set do
8              s ← e.source
9              res += s.reputation * e.weight
10         rep ← res
11         diff += |v.reputation − rep|
12         v.reputation ← rep
13         δ ← diff
```

Algorithm 2 gives the details of reputation propagation. A node v in the dependency graph receives its reputation by inheriting the weighted reputation from all of its parent nodes (Lines 7-12). 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 rapidly in a few hops, which does not work well for dependency graphs that often have many paths with many hops. The process of inheritance-based reputation propagation is an iterative process. For each iteration, the algorithm computes the sum of reputation differences for all the nodes (Line 11, 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 reputation of all nodes becomes stable. Empirically, we set $\delta = 1e - 13$. Note that the reputation of entry nodes remains unchanged (Lines 4-5).

Attack Sequence Reconstruction: SysRep leverages the edge 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. By carefully choosing this threshold, SysRep can filter out these irrelevant system activities while retaining the attack

provenance. In practice, the security analyst can first hide the non-critical edges using a higher threshold to focus on the investigation of the attack, and gradually shows 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 functionalities that the malicious payloads possess.

5 EVALUATION

We built SysRep (~8000 lines of code) upon Sysdig [58], 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 investigate these attacks, demonstrating the practical efficacy of SysRep. In total, our evaluations use real system monitoring data that consists of *2 billion* events.

We conduct four sets of evaluations. First, we evaluate the weight computation of SysRep to see whether it can effectively reveal critical edges from non-critical edges. Second, to evaluate the effectiveness of SysRep in propagating reputation, we compare the reputation scores of the POI against the expected reputation scores in both benign and attack scenarios. Third, we evaluate the impact of different reputation assignment schemes of SysRep. Finally, we show the performance statistics of different components of SysRep.

5.1 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.

5.1.1 Benign and Malicious Payloads Through Key System Interfaces. We performed 8 cases that employ the common system interfaces to inject benign and malicious payloads. These representative system interfaces are commonly exploited in attacks [16].

- 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 Real Attacks. We performed five real attacks that capture the important traits of APT attacks depicted from the Cyber Kill Chain framework [3]. 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, which is out of the scope of this paper. In total, we identified 10 key steps that are related to the POI for our evaluations in the five real attacks.

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 [6]. 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*).

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 [11, 25] 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*).

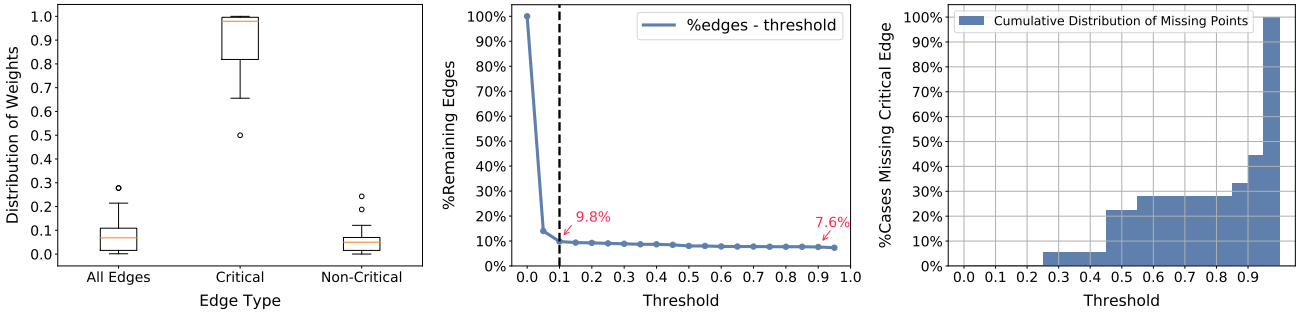
Attack 3: Data Leakage After Shellshock Penetration: After lateral movement stage, the attacker attempts to steal all the valuable assets from the host. This stage mainly involves the behaviors of local and remote file system scanning activity, copying and compressing of important files, and transferring 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*).

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 [48] which implements a remote access using the conventional web service API that internally uses DBUS service to run the designated commands. Due to the developers' 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*).

Attack 5: VPNFilter: We prototyped a famous IoT attack campaign; VPNFilter malware [10], which infected millions of dozens of different IoT devices exploiting a number of known or zero-day vulnerabilities [1, 2]. 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

Table 4: SysRep statistics of 18 cases

Case	Causality Analysis		Entry Nodes		#Critical Edges	Node Reputation		Edge Weight	
	#Nodes	#Edges	#Libs	#Non-Libs		Critical	Non-Critical	Critical	Non-Critical
curl	212	15142	177	1	2	0.00	0.46	0.99	0.02
mal_script	229	21124	184	1	3	0.02	0.50	0.97	0.01
python_wget	162	10597	139	2	2	0.01	0.49	0.99	0.06
scp	60	7912	52	1	3	0.00	0.43	1.00	0.07
shell_wget	115	4998	103	2	3	0.01	0.48	0.66	0.06
3File	225	25211	181	3	4	0.00	0.26	0.50	0.01
2File	221	21100	180	2	3	0.00	0.41	0.67	0.02
wget	217	19078	179	1	2	0.00	0.42	1.00	0.02
command-injection-c1	51	65	48	1	2	0.00	0.47	1.00	0.02
command-injection-c2	1438	9713	1430	1	2	0.02	0.50	0.96	0.00
data-leakage	4252	151200	4246	1	5	0.08	0.50	1.00	0.00
password-crack-c1	35	731	31	2	2	0.00	0.44	0.79	0.07
password-crack-c2	61	2477	54	1	4	0.01	0.44	1.00	0.04
password-crack-c3	25	58426	14	4	3	0.00	0.28	0.66	0.24
penetration-c1	60	314	54	1	2	0.02	0.47	0.97	0.07
penetration-c2	30	139	21	1	1	0.03	0.45	0.90	0.12
vpnfilter-c1	14	274	11	1	2	0.00	0.39	0.99	0.08
vpnfilter-c2	16	598	11	1	2	0.00	0.34	1.00	0.19
avg	**412.39**	**19394.39**	**395.28**	**1.50**	**2.61**	**0.01**	**0.43**	**0.89**	**0.06**



(a) Edge weights computed by SysRep **(b) Effectiveness of edge filtering** **(c) Critical edge loss from filtering**

Figure 3: Effectiveness of SysRep in attack sequence reconstruction

on-demand, at run-time. We prototyped the malware referring to one of its sample for x86 architecture [54].

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*).

5.2 Evaluation Results

5.2.1 Revealing Critical Edges Based on Edge Weights. Table 4 shows the detailed summary of statistics of using SysRep to investigate all the 18 attack cases. As we can see, the number of nodes and the number of edges in the dependency graph are 412 and 19,394 on average, and can be as large as 4,252 and 151,200, while the number of critical edges are generally quite small (<= 5), which is consistent with previous studies [36, 38, 39, 44]. Thus, it is non-trivial to reveal these edges from the massive non-critical edges.

Edge Weight Distribution: Figure 3a shows the distribution of average weight of critical edges and non-critical edges in each case using a box plot. We can see that the average weights of critical edges are significantly higher than that of non-critical edges. Note that the average weight of all edges are close to the average weight of non-critical edges because most of the edges are non-critical.

Filtering Non-Critical Edges: 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 attack cases in Section 5.1 by selecting a range of values from 0.05 to 0.95 with a pace of 0.05 as thresholds. Figure 3b 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 attack cases, as shown in Figure 3c. 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 3b and Figure 3c, we can conclude that *the weight scores of almost all the critical edges are above 0.90, while the weight scores of most of the non-critical edges are below 0.10.* Such results clearly demonstrate the effectiveness of SysRep in

Table 5: POI reputation of benign payloads through key system interfaces (expected 1.0)

Case	LP-Fixed	Fanout	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

leveraging the discriminative weights (Section 4.4) to reveal critical edges from non-critical edges. Furthermore, based on Figure 3c, we can suggest an optimal range of the threshold: $[0.1, 0.25]$.

5.2.2 Reputation Propagation. We evaluate the effectiveness of SysRep in identifying whether POI entities come from trusted sources or untrusted sources via reputation propagation in both normal and malicious scenarios, respectively.

Reputation Assignment: For evaluation purposes, we set the reputation of entry nodes representing trusted sources to 1.0, and entry nodes representing system libraries to 0.5. In malicious scenarios and real attacks, we set the reputation of entry nodes representing untrusted sources to 0.0. We propagate the reputation from entry nodes, and record the reputation scores of the POI (referred to as *POI reputation*). An effective approach will lead to a POI reputation that is close to 1.0 in the benign scenarios, instead of closing to 0.5 (similar to neutral libraries) or 0.0 (incorrectly associating the POI to untrusted sources). Similarly, an effective approach should lead to a POI reputation close to 0.0 in the malicious scenarios.

Comparison Approaches: To demonstrate the effectiveness of SysRep's weight computation, we compare SysRep with the following three weight computation approaches:

- LP-Fixed: We select a fixed parameter vector $(0.1, 0.5, 0.4)$ by manually tuning the parameters and then 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 our extended version of 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 1.0.

Furthermore, we compare our weight computation approach with the fanout approach used in the state-of-the-art causality analysis approach, PrioTracker [42]. PrioTracker mainly uses the fanout of nodes to prioritize the dependencies in the causality analysis for a given POI event. 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 fanout approach in reputation propagation. However, note that PrioTracker does not enable reputation propagation as SysRep does.

POI Reputations of SysRep: Tables 5 and 6 show the results for benign and malicious payloads through key system interfaces. Table 7 shows the results for the real attacks. The results show that SysRep effectively propagates the reputation scores from entry nodes 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 and reputation propagation.

Table 6: POI reputation of malicious payloads through key system interfaces (expected: 0.0)

Case	LP-Fixed	Fanout	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

Table 7: POI reputation of real attacks (expected: 0.0)

Case	LP-Fixed	Fanout	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

Entry Nodes: The Column "Entry Nodes" in Table 4 shows the number of entry nodes. As we can see, most of the entry nodes in dependency graph are libraries nodes, whose reputation can be assigned automatically using a pre-compiled list of verified libraries (for high reputation) and vulnerable libraries (for low reputation). The remaining non-lib nodes are likely to be root causes nodes, and their initial reputation could be set by security analysts based on their domain knowledge. The results indicate that SysRep converts the labor-intensive graph inspection task to the reputation assignment, significantly reducing manual efforts in attack investigation.

Impact of Non-Critical Edges with High Weights: Although most of the non-critical edges in a dependency graph have low weights as shown in Figure 3a, it's still possible for some non-critical edges to have high weights (*e.g.,* edges corresponding to data exchange of similar amount as the POI in irrelevant activities). However, we observe that the path from these activities to the POI are often cut off by edges with very low weights later, and hence these activities will have limited influence on the reputation of the POI node. To further confirm our observation, we use DBSCAN [23] (epsilon=0.01, minimal samples=5) to cluster the nodes in the real attack cases based on their final reputations. Results show that in all cases the nodes form two clusters and the nodes in the attack path are put in one cluster, which contains only nodes that are relevant to the attack activities. This indicates that non-critical edges with high weights did not adversely impact the POI reputation.

Comparisons with Other Weight Computation Approaches: From Tables 5 to 7, 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 5 and Table 6). This shows that the dependency graph is quite

Table 8: POI reputation of different reputation schemes

Case	Uniform Lib	Uniform All	
	POI	POI	Avg. Non-Lib
curl	0.99	0.91	0.91
mal_script	0.95	0.83	0.87
python_wget	0.99	0.87	0.88
scp	1.00	0.92	0.92
shell_wget	0.98	0.88	0.89
3File	1.00	0.97	0.97
2File	1.00	0.88	0.88
wget	1.00	0.99	0.99
command-injection-c1*	1.00	0.82	0.82
command-injection-c2*	0.97	0.90	0.93
data-leakage*	1.00	0.83	0.83
password-crack-c1*	1.00	0.85	0.85
password-crack-c2*	0.99	0.92	0.93
password-crack-c3*	0.99	0.89	1.00
penetration-c1*	0.97	0.85	0.87
penetration-c2*	0.95	0.79	0.83
vpnfilter-c1*	0.99	0.80	0.80
vpnfilter-c2*	1.00	0.86	0.86
avg	**0.99**	**0.88**	**0.89**

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 real attacks; (3) SysRep achieves the best performance in most of the cases. Specifically, SysRep achieves an average of 34.67%, 62.82%, and 43.76% improvement over LP-Fixed, LP-Global, and LP-Global+ in benign scenarios (Table 5), and an average of 94.52%, 95.61%, 86.21% improvement in malicious scenarios (Tables 6 and 7). 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 Fanout: The results show that SysRep achieves an average of 57.22% improvement over fanout in benign scenarios (Table 5) and an average of 87.22% improvement in malicious scenarios (Tables 6 and 7). As we can see from Table 7, while the average POI reputation achieved by fanout 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 (closing to 0.5), while SysRep correctly assigns low POI reputations (≤ 0.04) for these attack steps. These results demonstrate the superiority of SysRep over fanout.

5.2.3 Impacts of Reputation Assignment Schemes. In previous evaluations, we have demonstrated the effectiveness of SysRep when the reputations of entry nodes are well separated. Next, we would like to see how resilient SysRep is when the reputation of entry nodes is mutated intentionally by attackers or unintentionally by mistake. Thus, we run the experiments on all the cases under two new reputation schemes: (1) **Uniform Library**: assigning the reputation of a library node using a uniform random value in $[0.2, 0.8]$, *i.e.,* making some of them malicious and some of them benign, and keeping other nodes unchanged; (2) **Uniform All**: besides shuffling the reputation uniformly for libraries, assigning the reputation of a trusted source using a uniform random value in $[0.8, 1.0]$, and using $[0.0, 0.2]$ for a untrusted source, respectively.

Table 9: Performance statistics of SysRep

Case	Causality	Edge Merge	Node Split	Weight	Rep. Propagation
penetration-c1	0.088	0.001	0.002	0.013	0.069
penetration-c2	0.086	0.000	0.000	0.027	0.001
password-crack-c1	0.186	0.001	0.000	0.009	0.000
password-crack-c2	0.183	0.007	0.001	0.015	0.001
password-crack-c3	0.256	0.152	0.001	0.022	0.000
data-leakage	0.568	0.450	0.019	1.697	0.031
command-injection-c1	0.246	0.001	0.001	0.020	0.000
command-injection-c2	0.215	0.025	0.006	0.668	0.008
vpnfilter-c1	0.179	0.002	0.000	0.006	0.000
vpnfilter-c2	0.162	0.007	0.000	0.053	0.000
avg	**0.21**	**0.06**	**0.003**	**0.25**	**0.01**

Table 8 shows the results of the two schemes. Note that to easily compute the average, for the real attack cases (marked with $*$), we convert the POI reputation p to $1 - p$, so it will be shown as close to 1 when it is actually close to 0. As we can see, in the Uniform Library scheme, for all cases, the POI reputation is close to 1.0, indicating that even with the noise introduced in the libraries, the edge weights of SysRep prevents the adverse impact on the POI reputation. In the Uniform All scheme, the POI reputations of some cases have obvious changes (*e.g.,* mal_script, python_wget, shell_wget *etc.*). Column *Avg. Non-Lib* shows the average reputation of non-lib entry nodes representing trusted or suspicious sources. The Pearson correlation between the reputations of the POI and the non-lib entry nodes is 0.885, indicating a strong correlation.

These results indicate that SysRep is resilient to mutations on the library reputation and the POI reputation is strongly correlated with non-lib entry nodes. In real world scenarios, it will be relatively easy for the attacker to perturb the reputation for libraries given the massive number of libraries to keep track of. Also, since there are a few non-lib entry nodes in each attack and security analysts will be asked to inspect them, it will be more difficult for the attacker to mutate the reputation without being noticed. If the attacker succeeds in tricking the security analysts in trusting a suspicious source, the security analyst will misclassify the attack since in the end the security analyst will determine whether it forms an attack based on the source, no matter SysRep is used or not.

5.2.4 Performance of SysRep. To understand the performance of SysRep in investigating real attacks, we measure the execution time of SysRep on the attack cases. SysRep starts the computation by parsing a log (92.252s averagely) and building a global graph representation (3.277s averagely). Table 9 shows the execution time for the remaining components of SysRep. Besides the steps shown in the preprocessing step, *Causality Analysis*, *Edge Merge*, and *Node split* require 0.21s, 0.06s, and 0.003 on average. Note that *Weight Computation* and *Reputation Propagation* only requires 0.25s and 0.01s on average. In summary, the total time for running an analysis is about 2 minutes, but the major cost (*i.e.,* log parsing) can be improved by adopting caching or database indexing [29].

5.2.5 Summary. The evaluation results show that SysRep is effective in revealing critical edges from non-critical edges in both benign and attack scenarios, and is effective in propagating reputation from entry nodes to the POI. The results further show that the edge weights of SysRep can prevent perturbed reputation from system libraries and make sure the reputation of POI is strongly correlated to root cause nodes. Finally, SysRep can finish the analysis for a case within 2 minutes, and has the potential to be improved by adopting caching or database indexing.

6 DISCUSSION

6.1 Weight Computation

As shown in Section 5.2.2, SysRep achieves the best performance in all the compared weight computation approaches. One alternative for weight computation is to train a binary classifier using the three features and output a probability score as the edge weight. However, these classification-based approaches have very limited generalization capability in our problem context as our features are computed with respect to the specific POI, and thus the model learned for one type of attack can hardly generalize to other types of attacks with different POIs. Furthermore, these approaches require large amount of training data [13, 26], while in our problem context critical edges (*i.e.,* positive examples) are limited.

Among unsupervised learning approaches, approaches based on anomaly detection [19] 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 [46, 56], which we leave for future exploration.

6.2 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, (1) to have a lower time weight, the attacker may inject its malicious files earlier but start its attack later, or (2) 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. As SysRep's edge weights considers three features instead of one, both of these situations can be mitigated unless the attacker has the full knowledge of all the other edges and make all the features similar to them. An attacker may also choose to gain reputation first before attacking the system or stay under cover and attack probabilistically. To mitigate such situation, when assigning reputation, advanced analysis techniques [20, 55] on binaries and libraries can be applied to detect missing vulnerabilities.

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++ [14], which optimizes the seed initialization for better clustering quality, compared with the standard KMeans. Our local feature projection step is based on Linear Discriminant Analysis (LDA) [46], 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 [32], 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 [49], and identifies spam pages as those with low scores. Similar ideas have been applied in security and privacy application scenarios including Sybil detection [17, 27, 31], fake review detection [51], and attribute inference attacks [34, 61]. 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: Causality analysis based on system monitoring data plays a critical role. King et al. [38, 39] 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. Goel et al. [30] 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 [36, 40, 44], prioritizing dependencies [42], customized kernel [15], and optimizing storage [41, 59, 62]. However, these works still require non-trivial efforts from the security analysts to inspect the output dependency graphs, while 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. [28, 29] proposed a domain-specific languages that enables efficient attack investigation by querying the historical and real-time stream of system audit logs. 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 reduce the false threat alarms Hassan et al. [33] proposed to rely on the contextual and historical information of generated threat alert to combat threat alert fatigue. Milajerdi et al. [47] proposed to rely on the correlation of suspicious information flows to detect ongoing attack campaigns. Pasquier et al. [50] proposed a runtime analysis of provenance by combining runtime kernel-layer reference monitor with a query module mechanism. SysRep can be interoperated with these systems to achieve a better defense.

8 CONCLUSION

We propose a novel approach, SysRep, which computes discriminative weights for edges in a dependency graph built from system auditing events, and propagates reputation scores in the weighted graph to the POI. The discriminative edge weights enable SysRep to reveal critical edges for reconstructing the attack sequence. The POI reputation enables SysRep to identify the root causes of POI and determine the trustworthiness/suspiciousness of POI. Synergistically, SysRep significantly reduces the efforts of manual inspection and facilitates automatic attack investigation.

REFERENCES

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

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

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

[4] [n. d.]. The Equifax Data Breach. https://www.ftc.gov/equifax-data-breach.

[5] [n. d.]. The Marriott Data Breach. https://www.consumer.ftc.gov/blog/2018/12/marriott-data-breach.

[6] 2014. 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.

[7] 2014. Home Depot Confirms Data Breach At U.S., Canadian Stores. http://www.npr.org/2014/09/09/347007380/home-depot-confirms-data-breach-at-u-s-canadian-stores.

[8] 2015. OPM government data breach impacted 21.5 million. http://www.cnn.com/2015/07/09/politics/office-of-personnel-management-data-breach-20-million.

[9] 2016. Yahoo discloses hack of 1 billion accounts. https://techcrunch.com/2016/12/14/yahoo-discloses-hack-of-1-billion-accounts/.

[10] 2018. Schneier on Security: Router Vulnerability and the VPNFilter Botnet. https://www.schneier.com/blog/archives/2018/06/router_vulnerab.html.

[11] 2 2018. VPNFilter: New Router Malware with Destructive Capabilities. https://symc.ly/2IPGGVE.

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

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

[14] David Arthur and Sergei Vassilvitskii. 2007. K-means++: The Advantages of Careful Seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07)*.

[15] Adam M. Bates, Dave Tian, Kevin R. B. Butler, and Thomas Moyer. 2015. Trustworthy Whole-System Provenance for the Linux Kernel. In *USENIX Security*.

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

[17] Qiang C., Michael S., Xiaowei Y., and Tiago P. 2012. Aiding the Detection of Fake Accounts in Large Scale Social Online Services. USENIX.

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

[19] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly Detection: A Survey. *ACM Comput. Surv.* 41, 3, Article 15 (July 2009), 58 pages. https://doi.org/10.1145/1541880.1541882

[20] Marco Cova, Viktoria Felmetsger, Greg Banks, and Giovanni Vigna. 2006. Static detection of vulnerabilities in x86 executables. In *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*. IEEE, 269–278.

[21] Cyren. 2017. IP Reputation Check. http://www.cyren.com/ip-reputation-check.html.

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

[23] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. [n. d.]. A density-based algorithm for discovering clusters in large spatial databases with noise.

[24] Fireeye. 2017. Anatomy of Advanced Persistent Threats. (2017). https://doi.org/current-threats/anatomy-of-a-cyber-attack.html

[25] FireEye Inc. 2015. *HammerToss: Stealthy Tactics Define a Russian Cyber Threat Group*. Technical Report. FireEye Inc.

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

[27] Peng Gao, Binghui Wang, Neil Zhenqiang Gong, Sanjeev R. Kulkarni, Kurt Thomas, and Prateek Mittal. 2018. SYBILFUSE: Combining Local Attributes with Global Structure to Perform Robust Sybil Detection. In *CNS*.

[28] Peng Gao, Xusheng Xiao, Ding Li, Zhichun Li, Kangkook Jee, Zhenyu Wu, Chung Hwan Kim, Sanjeev R. Kulkarni, and Prateek Mittal. 2018. SAQL: A Stream-based Query System for Real-Time Abnormal System Behavior Detection. In *USENIX Security*.

[29] Peng Gao, Xusheng Xiao, Zhichun Li, Fengyuan Xu, Sanjeev R. Kulkarni, and Prateek Mittal. 2018. AIQL: Enabling Efficient Attack Investigation from System Monitoring Data. In *USENIX ATC*.

[30] Ashvin Goel, Kenneth Po, Kamran Farhadi, Zheng Li, and Eyal de Lara. 2005. The Taser Intrusion Recovery System. In *SOSP*.

[31] Neil Zhenqiang Gong and Di Wang. 2014. On the Security of Trustee-Based Social Authentications. *Trans. Info. For. Sec.* 9, 8 (2014).

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

[33] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. 2019. NODOZE: Combatting Threat Alert Fatigue with Automated Provenance Triage. (2019).

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

[35] Xuxian Jiang, AAron Walters, Dongyan Xu, Eugene H. Spafford, Florian Buchholz, and Yi-Min Wang. 2006. Provenance-Aware Tracing of Worm Break-in and Contaminations: A Process Coloring Approach. In *ICDCS*.

[36] 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. 2018. MCI : Modeling-based Causality Inference in Audit Logging for Attack Investigation. In *NDSS*.

[37] Taesoo Kim, Xi Wang, Nickolai Zeldovich, and M. Frans Kaashoek. 2010. Intrusion Recovery Using Selective Re-execution. In *OSDI*.

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

[39] Samuel T. King, Zhuoqing Morley Mao, Dominic G. Lucchetti, and Peter M. Chen. 2005. Enriching Intrusion Alerts Through Multi-Host Causality. In *NDSS*.

[40] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. 2013. High Accuracy Attack Provenance via Binary-based Execution Partition. In *NDSS*.

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

[42] Yushan Liu, Mu Zhang, Ding Li, Kangkook Jee, Zhichun Li, Zhenyu Wu, Junghwan Rhee, and Prateek Mittal. 2018. Towards a Timely Causality Analysis for Enterprise Security. In *NDSS*.

[43] Shiqing Ma, Juan Zhai, Yonghwi Kwon, Kyu Hyung Lee, Xiangyu Zhang, Gabriela F. Ciocarlie, Ashish Gehani, Vinod Yegneswaran, Dongyan Xu, and Somesh Jha. 2018. Kernel-Supported Cost-Effective Audit Logging for Causality Tracking. In *USENIX ATC*.

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

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

[46] Sebastian Mika, Gunnar Ratsch, Jason Weston, Bernhard Scholkopf, and Klaus-Robert Muller. 1999. Fisher Discriminant Analysis With Kernels.

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

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

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

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

[51] Shebuti Rayana and Leman Akoglu. 2015. Collective Opinion Spam Detection: Bridging Review Networks and Metadata. In *KDD*.

[52] Redhat. 2017. The Linux audit framework. https://github.com/linux-audit/.

[53] Sender Score. 2017. Sender Score. https://www.senderscore.org/.

[54] Sergei Shevchenko. 2018. VPNFilter 'botnet': a SophosLabs analysis. *A SophosLabs technical paper* (2018).

[55] Nick Stephens, John Grosen, Christopher Salls, Andrew Dutcher, Ruoyu Wang, Jacopo Corbetta, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. 2016. Driller: Augmenting Fuzzing Through Selective Symbolic Execution.. In *NDSS*, Vol. 16. 1–16.

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

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

[58] Sysdig. 2017. Sysdig. http://www.sysdig.org/.

[59] Yutao Tang, Ding Li, Zhichun Li, Mu Zhang, Kangkook Jee, Xusheng Xiao, Zhenyu Wu, Junghwan Rhee, Fengyuan Xu, and Qun Li. 2018. NodeMerge: Template Based Efficient Data Reduction For Big-Data Causality Analysis. In *ACM CCS*.

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

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

[62] Zhang Xu, Zhenyu Wu, Zhichun Li, Kangkook Jee, Junghwan Rhee, Xusheng Xiao, Fengyuan Xu, Haining Wang, and Guofei Jiang. 2016. High Fidelity Data Reduction for Big Data Security Dependency Analyses. In *CCS*.