A PT-based approach to construct efficient provenance graph for threat alert investigation

Shiming Men[1*], *Jian* Wang[1], and *Mai* Ye[1]

[1]Nanjing University of Aeronautics and Astronautics, China

Abstract. The Provenance Graph, constructed based on audit logs, encapsulates a wealth of causal relationships. When existing threat detection software issues an Event of Alert (EOA), security analysts conduct threat alert investigations based on the provenance graph. However, a provenance graph comprising tens of thousands of nodes and edges can lead to substantial time overhead. Moreover, coarse-grained audit logs can cause a dependency explosion in the provenance graph, thereby generating false causal relationships and interfering with the accuracy of the results. Hardware-assisted Processor Tracing (PT) can address the problem of false causal relationships caused by dependency explosion, by extracting the execution trace of a process and restoring instruction-level data flow. Leveraging PT features, we propose an efficient scheme, termed as GETSUS, which can reduce the size of the provenance graph while fully preserving the event sequences leading to the EOA. Empirical evidence shows that GETSUS can shrink a large provenance graph (approximately 200,000 edges) to a small one (around 50 edges). Compared to other schemes, the partial provenance graph output by GETSUS can fully retain the semantics related to the EOA, while improving the reduction efficiency by about tenfold.

1 Introduction

In today's world, with the continuous development of information technology, an increasing number of entities, such as corporations and individuals, are becoming targets of cyber-attacks. To counter these attacks, many corporations deploy the Threat Detection Software (TDS). This software continuously monitors activities and generates an Event of Alert (EOA) when suspicious activities occur. Cybersecurity analysts are required to manually investigate these EOAs. However, due to the lack of contextual information surrounding the alerts, it is challenging for analysts to conduct accurate investigations.

Provenance Graphs, based on system logs, have been demonstrated to provide rich contextual information [1, 2]. These graphs record the causal relationships among system entities (processes, files, and network sockets). Upon the generation of an Event of Alert (EOA), cybersecurity analysts can analyse the provenance graph [3-10], identify the attack entry nodes leading to the EOA (backward trace), and comprehend the consequences of the attack (forward trace) [7, 8].

Consequently, an increasing number of security analysts are opting to use provenance-based schemes for threat alert investigations. Through meticulous analysis of previous works [9, 10, 11], we have derived three key insights. First, within large provenance graphs containing an EOA, a small number of suspicious edges representing attack sequences are hidden among a vast number of normal edges. Compared to normal edges, these suspicious edges differ in multiple attributes and are more correlated with the EOA. Second, coarse-grained information based on system calls is insufficient to describe the activities of a process. Third, the suspiciousness level of each event should be adjusted based on the suspiciousness level of adjacent events. Despite the appeal of provenance-based methods and key insights, existing schemes for conducting threat alert investigations based on the provenance graph have several key limitations:

Dependency Explosion Problem [12, 13]. Current provenance tracing techniques only utilize coarse-grained provenance information, assuming that the output of an event is causally dependent on all previous input events [12], ultimately leading to a dependency explosion. This explosion can result in false dependencies during causal analysis, affecting the accuracy of the final results. The DepImpact [14] scheme, due to its exclusive use of coarse-grained information, fails to completely eliminate edges without real dependencies. Furthermore, dependency explosions can lead to the inclusion of more normal edges in the provenance graph, which exacerbates the difficulty of investigation.

Completeness Problem. Missing parts of the event sequence can render the final investigation results unreliable. NoDoze [4] opts to use a single feature to distinguish between suspicious and normal edges which lacks sufficient causal information for differentiation. Furthermore, NoDoze merges fixed-length event sequences to obtain the final partial graph which can

* Corresponding author: mailto:menshiming@nuaa.edu.cn
The author(s) declare(s) that there is no conflict of interest regarding the publication of this paper.

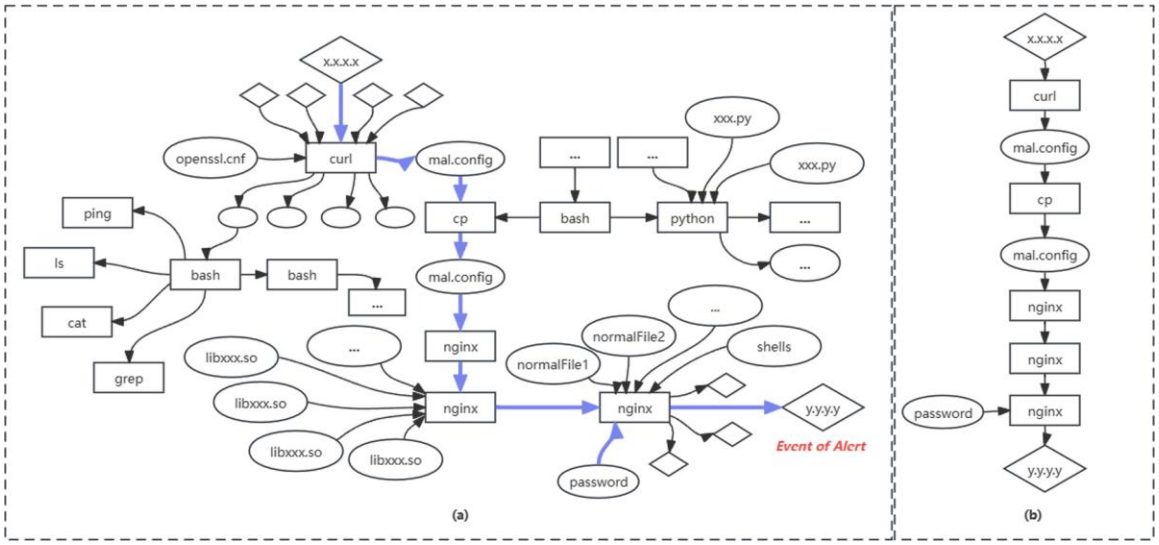

Fig. 1. Partial provenance graph of an attack.

miss suspicious entry nodes and lose key event sequences.

To address these challenges, we propose the GETSUS scheme based on key insights. GETSUS supplements coarse-grained audit logs with fine-grained Process Trace (PT) features, enhancing the contextual information of process nodes prone to dependency explosions, eliminating false dependencies, and resolving the dependency explosion problem. When a TDS generates an EOA, GETSUS identifies suspicious entry nodes related to the EOA. By performing forward tracing from these suspicious entry nodes, GETSUS obtains a suspicious provenance graph containing all event sequences leading to the EOA, assisting security analysts in conducting threat alert investigations. This paper makes the following contributions:

We successfully prevent false dependencies, and resolve the dependency explosion problem by integrating PT features with coarse-grained audit log records.

To facilitate the completion of threat alert investigations, we introduce the concept of a suspicious provenance graph, which fully retains the event sequence leading to the EOA. Compared to the original provenance graph, the suspicious provenance graph is reduced by approximately 4000 times while ensuring semantic completeness.

We have refined the existing algorithms, which enhances the ability to distinguish between suspicious and normal entry nodes.

We conduct a detailed and comprehensive evaluation. The results demonstrate the high efficiency of GETSUS.

2 Scheme (GETSUS) for constructing effective provenance graph

2.1 Overview

Figure 2 illustrates the workflow of GETSUS. When the existing TDS issues an EOA, GETSUS utilizes PT features to construct a provenance graph free of dependency explosions. GETSUS then optimizes this provenance graph, eliminating irrelevant edges and nodes (such as temporary files and library files). Based on the optimized provenance graph, GETSUS calculates and propagates suspiciousness values to identify suspicious entry nodes. By performing forward tracing analysis from these suspicious entry nodes, GETSUS outputs the suspicious provenance graph.

To facilitate understanding of GETSUS's function, Figure 1 illustrates its role in a real-world case. Here, diamonds represent socket nodes, rectangles represent process nodes, and ellipses represent file nodes. The edges between nodes represent causal relationships between two system entities. Figure 1(a) shows a partial provenance graph of a data leakage attack. Figure 1(b) shows the suspicious provenance graph which identifies event sequence leading to the EOA. Compared to the graph in Figure 1(a), the suspicious provenance graph directly reveals the attack sequences.

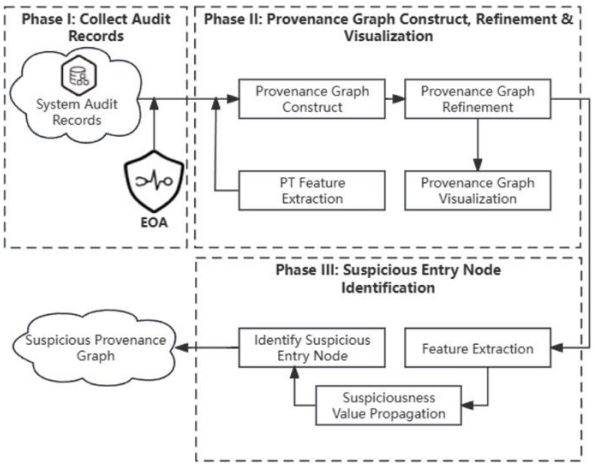

Fig. 2. Overview of GETSUS.

2.2 Scheme

In this section, we elaborate on some of the design details of GETSUS. In Section 2.2.1, we provide a detailed introduction to some of the key concepts. In Section 2.2.2, we discuss how GETSUS collects audit

logs and uses PT features to create a provenance graph. In Section 2.2.3, we describe how GETSUS identifies suspicious entry nodes. In Section 2.2.4, we explain how GETSUS obtains the suspicious provenance graph.

2.2.1 Basic concept

A Provenance Graph is composed of system entities and the causal relationships between them. System entities include processes, files, and sockets. Causal relationships, such as read, write, clone, etc., represent interactions between entities and are obtained by parsing system audit logs. GETSUS uses system calls to represent interactions between entities. An event ε is composed of a triplet $\{SRC, DST, REL\}$, where $SRC, DST \in \{Process, File, Socket\}$, and $REL \in \{read, write, \dots\}$. Table 1 shows various attributes in entities.

Table 1. Part critical information of system entities.

Entity	Attributes	Shapes
Process	pid, ppid, exe, args, ...	Rectangle
File	path, fd, ...	Ellipse
Socket	ip, port, ...	Diamond

Edges in the provenance graph represent system events, indicating interactions between system entities. An event sequence is composed of multiple causally related, ordered events ε. An event sequence of length n can be represented as $P = \{\varepsilon_1, \varepsilon_2, \varepsilon_3, \dots, \varepsilon_n \mid \varepsilon_i.DST = \varepsilon_{i+1}.SRC\}$. An event ε can have multiple event sequences. Event sequences can reflect the causes leading to ε and reveal the consequences triggered by ε. GETSUS uses 32 system calls to represent the interaction relationships. Some of them are displayed in Table 2.

Table 2. Representative system call.

Process/Process	fork, clone, execve, kill, vfork
Process/File	open, pipe, dup, close, mkdir, unlink
Process/Socket	connect, recvfrom, sendto, sendmsg, recvmsg, getpeername

2.2.2 Provenance graph construction & refinement

GETSUS utilizes Auditbeat to collect system logs. Most provenance analysises based on system call logs can only track coarse-grained information flows in the system, conservatively assuming that the output of a node depends on all its previous inputs. This leads provenance-based attack investigations into the predicament of dependency explosions. As shown in Figure 3(a), the curl process accesses a large number of Socket nodes and downloads numerous files. In Figure 3(a), the config or info files depend on all preceding socket nodes in chronological order, creating false dependencies. Inspired by Palantir [15], GETSUS uses PT as a feature to solve the dependency explosion problem. PT is a hardware feature enabled by modern Intel CPUs, which effectively records the execution trace of a process and recovers instruction-level data flows. PT provides fine-grained provenance information, and when combined with coarse-grained system audit log records, it can eliminate false causal relationships. As shown in Figure 3(b), PT can annotate the real dependency of the curl process and eliminate irrelevant nodes and edges.

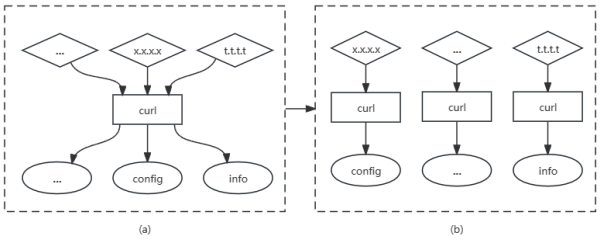

Fig. 3. The function of PT features.

GETSUS employs Palantir [15] to extract PT features, encompassing three steps: static binary analysis, runtime monitoring, and provenance analysis. First, this approach statically analyses program binary files to obtain the Control Flow Graphs (CFGs) and Call Graphs (CGs), followed by taint summarization. Leveraging the taint summary, this approach conducts taint propagation, and ultimately associates the PT trace with the audit logs through the system call sequence, yielding audit logs enriched with fine-grained information. Given that the extraction of PT features involves static binary analysis of the process, to enhance efficiency, we only perform static binary analysis on processes that are susceptible to dependency explosions, such as apache, nginx, firefox, etc. GETSUS generates a provenance graph devoid of false dependencies by parsing the optimized audit logs.

The majority of audit records are not essential when conducting causal analysis. To eliminate the irrelevant edges and nodes, GETSUS draws upon recent studies [13, 16] to trim the provenance graph from two aspects without losing semantic integrity. 1). The provenance graph typically contains many parallel edges between two nodes, resulting from the system's tendency to complete data transmission by proportionally invoking the corresponding system calls multiple times. Therefore, GETSUS amalgamates parallel edges occurring within a brief time frame. 2). The provenance graph is replete with temporary files and other irrelevant files, such as configuration files and library files. These files do not embody any semantic relationships and offer no assistance to subsequent causal analysis, necessitating their removal.

Upon trimming the provenance graph, GETSUS obtains a preliminarily reduced version. As GETSUS is solely concerned with the parts related to the EOA, it initiates a backward trace from the EOA using depth-first search, yielding a partial graph that encompasses all components dependent on the EOA.

2.2.3 Suspicious entry node identification

Inspired by DepImpact [14], GETSUS initially extracts temporal similarity, data flow similarity, and importance

features of the edges. Utilizing the AFK-MC2 algorithm [17], GETSUS clusters the edges, then assigns weight values to the edges via the RSLDA algorithm [18]. Starting from the EOA, GETSUS propagates suspiciousness values, ultimately identifying suspicious entry nodes.

GETSUS employs timestamps to represent the timing of events. GETSUS posits that events closer in time to the EOA are more likely to be suspicious events, hence they possess a higher temporal similarity, $S_t(e)$. Here, t_e denotes the timestamp of event e, and t_{EOA} represents the timestamp of the EOA. The calculation formula for $S_t(e)$ is as follows:

$$S_t(e) = ln\left(\frac{1}{|t_{EOA} - t_e|} + 1\right) \quad (1)$$

Similarly, events with a data flow more akin to the EOA are more likely to be suspicious events. GETSUS assigns a higher data flow similarity, $S_d(e)$, to such events. The specific formula is as follows, where d_e represents the data flow size of event e.

$$S_d(e) = \frac{1}{|d_{EOA} - d_e|} \quad (2)$$

The importance feature, $S_i(e)$, of an edge is quantified by the in-degree and out-degree of the edge's destination node and its calculation formula is as follows:

$$S_i(e) = \frac{OUT(v)}{In(v)} \quad (3)$$

Where, $e = \{u, v\}$ is event e. $OUT(v)$ and $IN(v)$ represents the outgoing edges and incoming edges of the destination node v respectively. If the only incoming edge to node v is edge e, this implies in terms of dependency that edge e is indispensable to node v. Therefore, to denote the importance of edge e, GETSUS assigns a higher importance value to this edge.

The suspiciousness value of an edge signifies its causal correlation with the EOA: the greater the suspiciousness value, the stronger its relevance to the EOA. The contributions of the three feature values to the suspiciousness value are nebulous, and an evenly distributed method could yield inaccurate results. GETSUS normalizes the three feature values. Subsequently, GETSUS opts for the AFK-MC2 clustering algorithm [17] to cluster all edges into two groups and roughly categorizes one group of edges as suspicious and the other as normal. The AFK-MC2 algorithm introduces certain enhancements to the K-means algorithm, optimizing the clustering outcome.

After obtaining two groups of edges, GETSUS employs the Robust Sparse Linear Discriminant Analysis (RSLDA) method [18], projecting the feature vectors onto a plane. RSLDA is capable of extracting the most discriminative features for classification. Specifically, RSLDA primarily utilizes the $l_{2,1}$ norm, adaptively assigning larger projection weights to categories of significant features. This approach allows RSLDA to minimize information loss, retain more semantic information, and provide a more effective distinction between suspicious and normal edges. Through RSLDA, GETSUS obtains the optimal projection matrix, denoted as $\mathbf{W} = (w_t, w_d, w_i)^T$ and the unnormalized weight value W_e' of the edge e is as follows:

$$W_e' = w_t * S_t(e) + w_d * S_d(e) + w_i * S_i(e) \quad (4)$$

GETSUS assigns a suspiciousness value of 1 to the EOA. To ensure that the suspiciousness value of any node doesn't exceed the maximum value of its child nodes and EOA, GETSUS normalizes the weight value of the edge $e = \{u, v\}$ as follows [14]:

$$W_e = \frac{W_e'}{\sum_{e_i \in OutEdges(u)} W_{e_i}'} \quad (5)$$

After normalizing the weights of all edges, GETSUS propagates the suspiciousness value (SV) of the EOA based on the weight values. Each node will have a corresponding suspiciousness value, which is ultimately propagated to various entry nodes through different paths. The specific calculation method for the node suspiciousness value is as follows:

$$SV(u) = \sum_{e_i \in OutEdges(u)} W_{e_i} * SV(e_i.DST) \quad (6)$$

2.2.4 Suspicious provenance graph

GETSUS ranks all entry nodes based on their suspiciousness values. We posit that entry nodes with higher suspiciousness values have a stronger causal relationship with the EOA. GETSUS categorizes suspicious entry nodes into: file entry nodes, process entry nodes, and socket entry nodes, selecting a few nodes with the highest suspiciousness values from each category as suspicious entry nodes. GETSUS initiates a forward causal analysis from these suspicious entry nodes to the EOA. During this forward causal analysis, GETSUS records a portion of the traversed provenance graph, designating this as the suspicious provenance graph. The suspicious provenance graph encapsulates all event sequences leading to the EOA, assisting security analysts in conducting threat alert investigations.

3 Evaluation

We have constructed the GETSUS using C++ and Python. In this section, we conduct a comprehensive evaluation of GETSUS through four simulated attacks.

3.1 Experiment setup

To evaluate the effectiveness of GETSUS in real-world attacks, we simulate four types of attacks: data leakage, insider threats, phishing software, and watering hole attacks. The primary operating environment for GETSUS is the 64-bit distribution of Ubuntu Linux

Table 3. Statistical data of four simulated attacks.

Attack Scenario	#Node			#Event	#Edge	#Suspicious Edge
	Process	File	Socket			
Data Leakage	844	785	316	195512	4963	11
Insider Threat	616	934	139	86127	2496	8
Phishing Email	452	402	18	79848	1479	6
Watering Hole	856	1087	127	293464	3871	12

16.04.07 LTS. This environment is equipped with a 12th Gen Intel(R) Core (TM) i9-12900H 2.50 GH, 16GB of memory. All experiments are conducted on a single computer. GETSUS uses Auditbeat to collect system logs and approximately 7,000 lines of C++ code to parse the system logs and generate the provenance graph. GETSUS stores the provenance graph data in a Neo4j database and provides visualization capabilities.

During the simulation of the four attacks, we have manually marked the EOA, as well as the suspicious edges and entry nodes in advance. Table 3 summarizes the statistical data after parsing the four simulated attacks.

3.2 Effectiveness of GETSUS

An effective suspicious provenance graph should encompass as many event sequences leading to the EOA as possible, ensuring no loss of attack semantics, while simultaneously incorporating fewer normal edges to avoid an overly large provenance graph. Therefore, we quantify the effectiveness of GETSUS using the number of edges (#Edge) in the suspicious provenance graph and the false negatives (FN), where FN represents the number of unidentified suspicious edges. We compared GETSUS with NoDoze and DepImpact, and the results demonstrated that GETSUS has zero FN and fewer #Edges. This indicates that GETSUS can efficiently assist security analysts in threat alert investigations. Table 4 displays the final results. The FN of GETSUS in all four attacks is zero, indicating that the suspicious provenance graph includes all event sequences leading to the EOA. The smallest #Edges data indicates that GETSUS includes the fewest normal edges, demonstrating the best reduction in the original graph. The data shows that the suspicious provenance graph obtained by GETSUS is approximately 31% smaller than the suboptimal DepImpact and about eight times smaller than NoDoze. Compared to the original graph, GETSUS reduces approximately 200,000 audit logs to about 50 edges, a reduction of approximately 4000 times.

The subpar performance of NoDoze and DepImpact is understandable. The implementation of NoDoze requires the support of an event frequency database. Therefore, the effectiveness of NoDoze depends on whether its event frequency database can encompass all normal events. NoDoze bases its anomaly determination on a single frequency feature and ultimately merges the most suspicious fixed-length paths. Hence, in most cases, NoDoze may miss suspicious events due to its reliance on a single feature and may also miss suspicious entry nodes due to its fixed-length path. Consequently, NoDoze has the highest FN and #Edge. DepImpact's performance is inferior to GETSUS. As DepImpact uses coarse-grained audit information, the results from DepImpact exists false dependency relationships, leading to a higher number of #Edge and FN.

Table 4. Effectiveness of each approach.

Attack	NoDoze		DepImpact		GETSUS	
	FN	#Edge	FN	#Edge	FN	#Edge
Data Leakage	3	320	1	63	0	45
Insider Threat	1	241	0	53	0	38
Phishing Email	0	165	0	37	0	22
Watering Hole	4	386	2	74	0	54

3.3 Runtime performance

Table 5. Runtime of GETSUS.

Attack	Provenance graph Construct (s)	Suspicious provenance graph (s)
Data Leakage	8.148	4.531
Phishing Email	4.093	2.426
Watering Hole	3.884	3.271
Watering Hole	14.373	5.725

We measured the time overhead of GETSUS. As show in Table 5, GETSUS spent an average of 7.62 seconds constructing the provenance graph and an average of 3.988 seconds constructing the suspicious provenance graph. The results demonstrate that the time overhead of GETSUS is within a reasonable range.

3.4 Selection of suspicious entry nodes

GETSUS propagates the suspiciousness values of EOA to entry nodes. Among the three types of entry nodes, GETSUS selects those with higher suspiciousness values as suspicious entry nodes respectively. More suspicious entry nodes enable GETSUS to encompass more suspicious edges, resulting in a lower False Negative Rate (FNR), $FNR = FN/E_t$, where E_t

represents the total number of suspicious edges. However, selecting more suspicious entry nodes can lead to the inclusion of more normal edges, increasing irrelevant semantics and complicating the task for security analysts. Figure 4 indicates that when the number of suspicious entry nodes ranges from 1 to 3, GETSUS encompasses all key edges. If there is a desire to further increase the number of suspicious entry nodes, it will inevitably increase the amounts of normal edges and nodes.

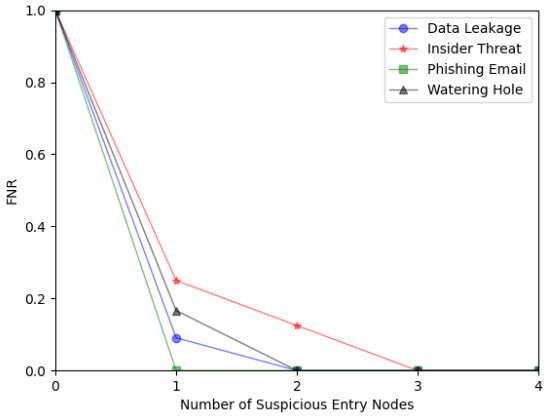

Fig. 4. FNR values using various amounts of suspicious entry.

4 Conclusion

We have developed the GETSUS scheme, which retains all event sequences leading to the EOA, and reduces the provenance graph by approximately 4000 times, effectively assisting security analysts in threat alert investigations. Initially, GETSUS employs PT features to supplement the audit logs with causal relationships, addressing the dependency explosion problem. Subsequently, GETSUS generates and optimizes the provenance graph. GETSUS assigns an appropriate weight value to each edge, propagates suspiciousness values backward from the EOA, and identifies suspicious entry nodes. By conducting forward tracing from these suspicious entry nodes, GETSUS ultimately outputs a suspicious provenance graph. The efficiency of GETSUS has been empirically validated.

Acknowledgements

We express our gratitude to the reviewers for their constructive comments on this paper.

References

1. A. Bates, D. J. Tian, K. R. Butler, T. Moyer, *Trustworthy Whole-System provenance for the linux kernel*, in 24th USENIX Security Symposium (2015).
2. M. N. Hossain, S. M. Milajerdi, et al., *SLEUTH: Real-time attack scenario reconstruction from COTS audit data*, in 26th USENIX Security Symposium (2017).
3. A. Goel, K. Po, K. Farhadi, Z. Li, E. De. Lara, *The taser intrusion recovery system*, in Proceedings of the twentieth ACM symposium on Operating systems principles (2005).
4. W. U. Hassan, S. Guo, et al., A. Bates, *Nodoze: Combatting threat alert fatigue with automated provenance triage*, in network and distributed systems security symposium (2019).
5. Y. Kwon, F. Wang, et al., *MCI: Modeling-based causality inference in audit logging for attack investigation*, in Network and Distributed Systems Security (NDSS) Symposium (2018).
6. T. Kim, X. Wang, N. Zeldovich, M. F. Kaashoek, *Intrusion recovery using selective re-execution*. In 9th USENIX Symposium on Operating Systems Design and Implementation (2010).
7. S. T. King, P. M. Chen, *Backtracking intrusions*, in Proceedings of the nineteenth ACM symposium on Operating systems principles (2003).
8. S. T. King, Z. M. Mao, et al., *Enriching Intrusion Alerts Through Multi-Host Causality*. In Ndss (2005).
9. Y. Liu, M. Zhang, et al., *Towards a Timely Causality Analysis for Enterprise Security*, in NDSS (2018).
10. S. Ma, X. Zhang, D. Xu, *Protracer: Towards practical provenance tracing by alternating between logging and tainting*, in 23rd Annual Network And Distributed System Security Symposium (2016).
11. P. Gao, X. Xusheng, Z. Li, F. Xu, S. R. Kulkarni, P. Mittal, *AIQL: Enabling efficient attack investigation from system monitoring data*, in 2018 USENIX Annual Technical Conference (2018).
12. K. H. Lee, X. Zhang, D. Xu, *High Accuracy Attack Provenance via Binary-based Execution Partition*, in NDSS (2013).
13. Z. Xu, Z. Wu, et al., *High fidelity data reduction for big data security dependency analyses*, in Proceedings of the 2016 ACM SIGSAC conference on computer and communications security (2016).
14. P. Fang, P. Gao, et al., *Back-Propagating System Dependency Impact for Attack Investigation*. In 31st USENIX Security Symposium (2022).
15. J. Zeng, C. Zhang, Z. Liang, *PalanTír: Optimizing Attack Provenance with Hardware-enhanced System Observability*, in Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (2022).
16. K. H. Lee, X. Zhang, D. Xu, *Loggc: garbage collecting audit log*, in Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (2013).
17. O. Bachem, M. Lucic, et al., *Fast and provably good seedings for k-means*, advances in neural information processing systems, 29 (2016).
18. J. Wen, X. Fang, et al., *Robust sparse linear discriminant analysis*, in IEEE Transactions on Circuits and Systems for Video Technology (2018).