

Provenance-based Intrusion Detection: Opportunities and Challenges

Xueyuan Han
Harvard University

Thomas Pasquier
University of Cambridge

Margo Seltzer
Harvard University

Abstract

Intrusion detection is an arms race; attackers evade intrusion detection systems by developing new attack vectors to sidestep known defense mechanisms. Provenance provides a detailed, structured history of the interactions of digital objects within a system. It is ideal for intrusion detection, because it offers a *holistic*, attack-vector-agnostic view of system execution. As such, provenance graph analysis fundamentally strengthens detection robustness. We discuss the opportunities and challenges associated with provenance-based intrusion detection and provide insights based on our experience building such systems.

1 Introduction

System security continues to be an arms race between intruders and defenders. In this arms race, attackers adapt in response to defense mechanisms and *always* win. Defeating attackers requires rethinking traditional defeat- and exploit-based mitigation techniques, which lack complete security coverage [16]. We propose taking a holistic, attack-vector-agnostic view of system execution.

We claim that provenance is the ideal data to use for such a task and that provenance graph-based analysis is the ultimate means towards achieving complete security coverage. Provenance refers to meta-data describing how digital objects came to be in their current state. It provides a complete, structured view of what happened on the system [4] by presenting complex dependencies and causality relationships between digital objects as a directed acyclic graph (DAG). As such, it is well suited for intrusion detection. An intrusion manifests in anomalous interdependencies among data objects that deviate from those found in non-malicious execution. In fact, in attack causality analysis [10], provenance has long been used to explain intrusions (§ 3).

Provenance graph analysis strengthens adversarial robustness, because the graphs exhibit long-range correlations and dependencies allowing for causal reasoning about intrusions [2] (§ 2). Such causal reasoning enables detection of sophisticated attacks, such as network attacks, that remain undetected for long periods of time. In prior work [8], we reduced a host-based intrusion detection problem to a graph-based

anomaly detection problem, in which graph analysis identified structured execution traces that represented an intrusion. However, intrusion detection on provenance graphs requires analyzing dynamic, attributed, streaming graphs, which are rarely studied in the literature. Given the development of fine-grained, whole-system provenance capture systems [15], this task becomes even more challenging as the graphs rapidly become extraordinarily large [4]. However, we can use domain-specific knowledge of provenance graphs to simplify the challenge of identifying anomalies, making it an easier problem than general-purpose graph analysis suggests. For example, since execution history is immutable, we can assume that provenance graphs only increase in size (i.e., there are never deletions). This property allows us to incrementally and progressively reason about causality without needing to look backwards.

2 Applicability

Data provenance has seen use in areas such as databases and computational sciences. While it now also appears as part of real-time security analysis [4], most approaches are variations of dynamic taint analysis of provenance data. While simple and effective on their own merits, they are limited to constraining information flows within a system (e.g., data loss prevention, access control, and regulatory compliance); little work has been done to detect intrusions from outside the system [8, 15].

Host-based anomaly detection systems define some baseline normal behavior and then classify as abnormal any behavior that significantly deviates from the baseline. The approach is predicated on the assumption that intrusions are highly correlated to abnormal behavior. Many existing systems use unstructured collections of multidimensional data (e.g., audit logs) to detect outlying points in a high-dimensional feature space, formulating intrusion detection as point-based outlier detection to leverage various learning and data mining techniques. Provenance, however, is structured graph data that represents relationships between a digital item (i.e., data entity), a transformation on that item (i.e., activity), and agents (i.e., persons and organizations) associated with the item and the transformation. Hence, unlike the prior work, we formulate the host-based intrusion detection problem as a graph-based anomaly detection problem defined as follows [2]:

Definition 1. *The graph-based intrusion detection problem is to identify components of the graph that are significantly different from those in a learned model of the graph.*

Using a provenance graph-based approach to intrusion detection is suitable for various reasons:

- *Provenance captures complete access to security-sensitive kernel objects:* State-of-the-art provenance whole-system capture systems leverage the Linux Security Module (LSM) interface to record provenance for every security-related interaction, rather than intercepting system calls. They can be extended to verifiably monitor all information flows in a system [7].
- *Provenance makes explicit the relationships among objects:* One powerful feature of provenance is its native graphical representation to show system execution as interactions between data objects. However, such interdependencies are innate to every execution trace, even in seemingly unstructured audit data from logging systems such as `auditd`. In fact, there exist frameworks that reconstruct graph-based provenance from flat audit data to allow for reasoning about system execution [6]. However, this post hoc approach comes with a caveat: it is harder to ensure completeness or correctness of the graph built from flat audit data [17].
- *Intrusions result from unexpected interactions:* The entry point to a victim system may be a single, isolated event, but its effects must propagate for an intrusion to be fruitful to an attacker. For example, consider an insider attacker who wishes to steal sensitive information from a data server under his control. He first installs a malicious BASH script that discovers and collects all documents (i.e., a single entry point to the server). However, to successfully steal the information, he needs to either transfer it to a foreign machine or write it to an external storage device. The key to detecting the data leak is to connect the collection of the data to the transmission of the data, which in a provenance graph is clearly represented as a chain of dependencies between processes, files, and sockets.
- *Graph representation improves robustness:* graphs are generally more adversarially robust, i.e., it is harder for an attacker to camouflage her behavior to fit into the reference graph structures [2]. In fact, we claim that *the provenance graph of an intrusion must differ from that of a valid execution when we use an LSM-based whole-system provenance capture system.* As LSM places hooks on any execution path that generates an information flow [2], if the capture system records provenance on every such path, violations of security policies will be evident from the provenance graph. Moreover, the attacker must also have the knowledge of the substructures that are referenced by the IDS, which alone requires significant effort. For example, the attacker from the previous example may evade detection if each step is allowed when performed in isolation. An advanced attacker

can even fake the IP address of the foreign machine. However, when considering the chain of actions as a whole (i.e., an abnormal graph substructure), we can identify the intrusion.

3 Opportunities and Challenges

Analyzing dynamic, attributed graphs is difficult. Graph anomaly detection in this setting requires detecting changes over time, which in turn requires a formal notion of similarity defined specifically for the target domain [2]. With attributed vertices and edges, changes can occur both structurally and in labels. Provenance graphs further complicate the matter as each vertex and edge usually has a set of attributes (instead of a single type attribute), and the number of attributes varies depending on the type of the vertex/edge. To enable online intrusion detection, one also receives the provenance graph in a streaming fashion and must perform the analysis in realtime. However, provenance graphs are acyclic, thus having a topological ordering that simplifies computation. Events therefore can be partially ordered as they are streamed for analysis [15]. We can then efficiently reason over the vast amount of information contained in vertex and edge labels, which, combined with structural information, reflects various aspects of system execution. In the following sections, we discuss the main opportunities and challenges associated with provenance-based intrusion detection.

3.1 Opportunities

Opportunity 1: Provenance graph structures and labels encode the complete, historical context of system execution. A useful intrusion detection system learns detailed normal behavior from the past. Given flat audit data with no completeness guarantee, an IDS is limited by the data recorded in the audit logs. It is also difficult to obtain higher-order dependencies [18]. In some cases, the type of information it learns from is determined empirically by the attack vectors it is designed to detect. Such an ad hoc approach ultimately leads to the arms race described in § 1. In contrast, whole-system provenance provides a complete view of information flow that natively reflects higher-order correlations and long-range dependencies. Its graph structure also allows for graph-based analysis. We illustrate its benefits by describing the following principles that provenance analysis embodies.

- *Principle 1: Identify semantically meaningful substructures.* Provenance graphs can become large, obfuscating important events that require special attention. Complex system interactions within a task and between tasks further cloud understanding. Therefore, it is important to identify substructures/subgraphs that are semantically coherent (e.g., describing a single task within a program). Macko et al. [13] developed two centrality metrics to perform local clustering on provenance graphs for task separation. Generic metrics

used to discover communities are also applicable, albeit expensive in certain cases. Significant changes in those structures usually imply intrusions. For example, most control-data attacks alter the control flow of a program to execute injected malicious code. They typically start a new shell with the privilege of the victim process [5], which inevitably introduces unexpected vertices and edges in the provenance graph. Akoglu et al. [2] summarized various distance measures to detect structural anomalies in dynamic graphs.

• *Principle 2: Incorporate time.* The rate of provenance event creation is proportional to kernel object access rate. As each access to security-sensitive kernel object results in (at least) one edge in the graph, provenance graphs reflect this rate through the number of vertices and/or edges per unit of time. Although some benign workloads exhibit a high rate of provenance generation (e.g., building a kernel [15]), bursts of intense provenance generation frequently indicate an attack. For example, attackers exploit race conditions to deploy Time-of-Check-to-Time-of-Use (TOCTOU) attacks. The fairly recent Dirty COW attack (CVE-2016-5195), in which the Linux kernel's memory subsystem incorrectly handled copy-on-write (COW), granting write access to private read-only memory mappings, used two threads simultaneously bombarding the system with `madvise` and `write` system calls. These calls produce elements of the provenance graph at a rate rarely observed during normal behavior.

• *Principle 3: Keep history in mind.* Advanced persistent threat (APT) attacks are usually a set of continuous, long-running processes that permeate the victim system. Noticing such attacks requires a holistic understanding of system execution starting from its initialization. In fact, any intrusion that requires retrospective analysis on previously processed portion of the graph can be discovered only if the detection system “remembers” history. However, the sheer volume of provenance data renders any attempt at a complete review impractical. One way to mitigate this needle-in-a-haystack problem is to incrementally build a concise yet comprehensive model that memorizes the historical context of the graph. For example, Lemay et al. [11] designed regular grammars for provenance DAGs to succinctly summarize the graph structure.

Opportunity 2: Provenance graphs are topologically and partially ordered. This property follows naturally from the fact that provenance graphs are DAGs and that they truthfully reflect the causal relationships of events that occurred on the system. We took advantage of this property and designed a real-time provenance analysis framework to enable semantically rich security services [8]. In particular, a vertex-centric graph framework facilitates provenance graph analysis with its correctness guaranteed by the two partial ordering properties: 1) once an outgoing edge to a vertex arrives, we know that we have observed all incoming edges to that vertex; 2) we receive all edges and vertices along a path in order.

Opportunity 3: Provenance graphs enrich attack attribution and sense-making. Attribution is an important feature that allows system administrators to quickly understand the source of an intrusion so that they can remedy the issue in a timely fashion and effectively control the damage. Many intrusion detection systems suffer from a high false positive rate. Attribution helps administrators quickly reject false positive alarms, effectively making the IDS more usable. Provenance graphs are causality graphs that naturally allow for sense-making, providing a causal chain of events for reasoning. For example, King et al. [9] designed a system that structures OS-level audit logs to automatically identify sequences of steps that occurred in an intrusion, starting from a single detection point.

3.2 Challenges

Challenge 1: It is difficult to obtain a good graph summary. From Opportunity 1 (§ 3.1), we see that a good graph summary should at least adhere to all the principles discussed. For principles that do not consider graph structures, we can learn trends via applications of machine learning or empirically, e.g., by finding and setting a threshold. However, the streaming nature of provenance data for online intrusion detection makes graph analysis challenging. One approach is to segment the graph using a time window, though one needs to determine an appropriate window size.

Challenge 2: Online intrusion detection requires efficient computation. Even with the framework described in § 3.1, the computation itself (e.g., to generate a good graph summary) must be efficient enough to detect an intrusion before it wreaks havoc on the system. Many intrusion detection systems require training on known datasets, which is often performed offline [3]. Efficiency therefore is usually a primary concern during deployment. Complicated graph algorithms, such as subgraph isomorphism, are often NP-complete and are suitable only for small graphs. Machine learning and data mining approaches on graphs, e.g., graph kernels, offer alternatives with polynomial or even linear time complexity.

Challenge 3: The complexity of the system makes provenance graphs difficult to understand. There exists a trade-off between the completeness of provenance and the succinctness of the resulting graph. With whole-system provenance capture, this trade-off becomes even clearer as a large number of underlying system dependencies are captured. For example, Liu et al. [12] showed that a simple `sshd` command can trigger a massive number of Linux commands that are used to update Linux environment variables, which results in a large provenance subgraph describing these activities. However, they also proposed an algorithm that takes into account factors, such as rareness and dataflow termination, to determine the priority of events during backward and forward tracking of a provenance graph.

4 Experience

In prior work [8], we presented a provenance-based intrusion detection system. As we refined our system [15], we identified idiosyncrasies that differentiate intrusion detection via provenance and via audit logs. In addition to the properties already discussed, provenance captures interactions across applications that are invaluable in intrusion detection.

Based on our prior experience, we identify the following keys to provenance-based intrusion detection:

- *Understand the provenance capture mechanism and the graph it produces:* It is important to understand what information is captured, how it is captured, and at what level of granularity. These all affect graph interpretation. For example, we have worked with capture systems that record both thread-level details [15] and process-only details [6]. They have fundamentally different underlying capture mechanisms, and therefore, we need to make different assumptions about the provenance graphs they generate, even when they are capturing provenance of the same system execution. More importantly, we need to make correct assumptions, which is fundamental to the correctness of any provenance graph analysis. Consequently, it is essential to specify the formalization of the graphs from different capture mechanisms, not to generalize.

Sometimes, existing provenance capture systems may not fulfill the needs of an IDS; jointly developing a provenance capture system and a provenance-based IDS is most likely to improve the performance of both systems.

- *Build datasets to benchmark IDSes:* The lack of labeled datasets is a serious obstacle to work in this area. As provenance capture mechanisms evolve, a plug-and-play system that can automatically rerun experiments is valuable. We use Vagrant to generate experimental data in a virtual environment [1]. However, labeling datasets is tricky [14]. One cannot simply label an entire provenance graph as an “intrusion”, since an IDS could mistakenly interpret a benign subgraph as an intrusion entry point. On the other hand, a provenance graph of seemingly normal system execution might contain unexpected execution errors, which, though not part of an intrusion, still deviate from specified normal behavior. This difficulty leads to misleading comparison metrics, such as precision, recall, and F-measure. Benchmarking IDSes remains an important open problem.

5 Conclusion

We propose to realize robust, attack-vector-agnostic intrusion detection through analysis on provenance graphs and identify opportunities and challenges specific to whole-system, provenance-based intrusion detection. While the concept of OS-level provenance is almost a decade old, formalization and theoretical studies of its graphs have not yet materialized. Applying whole-system provenance to intrusion detection [8] requires a formal understanding of provenance.

We invite fellow researchers in both theory and provenance communities to continue this exploration with us.

References

- [1] [n. d.]. Provenance datasets. ([n. d.]). <https://github.com/crimson-unicorn/dataset/tree/master/vulnerabilities>.
- [2] Leman Akoglu, Hanghang Tong, and Danai Koutra. 2015. Graph based anomaly detection and description: a survey. *Data Mining and Knowledge Discovery* 29, 3 (2015), 626–688.
- [3] Stefan Axelsson. 2000. *Intrusion detection systems: A survey and taxonomy*. Technical Report. Technical report.
- [4] Adam M Bates, Dave Tian, Kevin RB Butler, and Thomas Moyer. 2015. Trustworthy Whole-System Provenance for the Linux Kernel. In *USENIX Security Symposium*. 319–334.
- [5] Shuo Chen, Jun Xu, Emre Can Sezer, Prachi Gauriar, and Ravishankar K Iyer. 2005. Non-Control-Data Attacks Are Realistic Threats. In *USENIX Security Symposium*, Vol. 5.
- [6] Ashish Gehani and Dawood Tariq. 2012. SPADE: support for provenance auditing in distributed environments. In *Proceedings of the 13th International Middleware Conference*. Springer-Verlag New York, Inc., 101–120.
- [7] Laurent Georget, Mathieu Jaume, Frédéric Tronel, Guillaume Piolle, and Valérie Viet Triem Tong. 2017. Verifying the reliability of operating system-level information flow control systems in linux. In *Formal Methods in Software Engineering (FormalISE), 2017 IEEE/ACM 5th International FME Workshop on*. IEEE, 10–16.
- [8] Xueyuan Han, Thomas Pasquier, Tanvi Ranjan, Mark Goldstein, and Margo Seltzer. 2017. FRAPpuccino: Fault-detection through Runtime Analysis of Provenance. In *Workshop on Hot Topics in Cloud Computing (HotCloud '17)*. USENIX.
- [9] Samuel T King and Peter M Chen. 2003. Backtracking intrusions. *ACM SIGOPS Operating Systems Review* 37, 5 (2003), 223–236.
- [10] Samuel T King, Zhuoqing Morley Mao, Dominic G Lucchetti, and Peter M Chen. 2005. Enriching Intrusion Alerts Through Multi-Host Causality. In *NDSS*.
- [11] Mark Lemay, Wajih Ul Hassan, Thomas Moyer, and Nabil Schear Warren Smith. [n. d.]. Automated Provenance Analytics: A Regular Grammar Based Approach with Applications in Security.
- [12] Yushan Liu, Mu Zhang, Ding Li, Kangkook Jee, Zhichun Li, Zhenyu Wu, Junghwan Rhee, and Prateek Mittal. [n. d.]. Towards a Timely Causality Analysis for Enterprise Security. ([n. d.]).
- [13] Peter Macko, Daniel Margo, and Margo Seltzer. 2013. Local clustering in provenance graphs. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. ACM, 835–840.
- [14] Federico Maggi, Matteo Matteucci, and Stefano Zanero. 2010. Detecting intrusions through system call sequence and argument analysis. *IEEE Transactions on Dependable and Secure Computing* 7, 4 (2010), 381–395.
- [15] Thomas Pasquier, Xueyuan Han, Mark Goldstein, Thomas Moyer, David Evers, Margo Seltzer, and Jean Bacon. 2017. Practical whole-system provenance capture. In *Proceedings of the 2017 Symposium on Cloud Computing*. ACM, 405–418.
- [16] Jonathan Pincus and Brandon Baker. 2004. Mitigations for low-level coding vulnerabilities: Incomparability and limitations. (2004).
- [17] Devin J Pohly, Stephen McLaughlin, Patrick McDaniel, and Kevin Butler. 2012. Hi-Fi: collecting high-fidelity whole-system provenance. In *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM, 259–268.
- [18] Nong Ye et al. 2000. A markov chain model of temporal behavior for anomaly detection. In *Proceedings of the 2000 IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop*.