Applying Provenance in APT Monitoring and Analysis

Practical Challenges for Scalable, Efficient and Trustworthy Distributed Provenance

Graeme Jenkinson
Lucian Carata
Nikilesh Balakrishnan
Thomas Bytheway
Ripduman Sohan
Robert N. M. Watson
University of Cambridge
firstname.lastname@cl.cam.ac.uk

Jonathan Anderson
Brian Kidney
Memorial University
firstname.lastname@mun.ca

Amanda Strnad
Arun Thomas
BAE Systems Inc
firstname.lastname@baesystems.com

George Neville-Neil
Neville-Neil Consulting
gnn@neville-neil.com

Abstract

Advanced Persistent Threats (APT) are a class of security threats in which a well-resourced attacker targets a specific individual or organisation with a predefined goal. This typically involves exfiltration of confidential material, although increasingly attacks target the encryption or destruction of mission critical data. With traditional prevention and detection mechanisms failing to stem the tide of such attacks, there is a pressing need for new monitoring and analysis tools that reduce both false-positive rates and the cognitive burden on human analysts.

We propose that local and distributed provenance metadata can simplify and improve monitoring and analysis of APTs by providing a single, authoritative sequence of events that captures the context (and side effects) of potentially malicious activities. Provenance metadata allows a human analyst to backtrack from detection of malicious activity to the point of intrusion and, similarly, to work forward to fully understand the consequences. Applying provenance to APT monitoring and analysis introduces some significantly different challenges and requirements in comparison to more traditional applications. Drawing from our experiences working with and adapting the OPUS (Observed Provenance in User Space) system to an APT monitoring and analysis use case, we introduce and discuss some of the key challenges in this space. These preliminary observations are intended to prime a discussion within the community about the design space for scalable, efficient and trustworthy distributed provenance for scenarios that impose different constraints from traditional provenance applications such as workflow and data processing frameworks.

CCS Concepts • **Information systems** → **Data provenance**; • **Software and its engineering** → **Distributed systems organizing principles**; • **Security and privacy** → *Distributed systems security*

Keywords Advanced Persistent Threat, APT, provenance, distributed systems, security

1. Introduction

The motivating use case for our work is security, specifically forensic analysis and real-time monitoring of Advanced Persistent Threats (APT). The term APT refers to a class of security threats where a well-resourced attacker (typically a nation-state or criminal actor) targets a specific individual or organisation with a predefined goal[1]. Frequently that goal is the exfiltration of confidential material, although increasingly attacks target the encryption or destruction of mission critical data. APT-style attacks target enterprise IT Systems: large-scale distributed systems consisting of heterogeneous servers and end-user devices running diverse applications.

The *intrusion kill-chain* (Hutchins et al. 2011), which models potential attack stages, is commonly employed to defend against advanced persistent threats. In this paper we describe how provenance metadata might support monitoring and analysis of APT-style attacks as an attacker progresses through the stages of the intrusion kill-chain.

[1] This is in contrast to traditional attacks where attackers seek to compromise somebody, but do not particularly care who that is.

The application of provenance to monitoring and analysis of APT-style attacks introduces the provenance system to a whole host of distributed system concerns. Consider, for example, an analyst's effort to understand both how an attack took place, and the persisting impact of the attack – not just within a single host, but across many hosts, some of which are by definition in a compromised and untrustworthy state. Collecting and querying provenance data to serve such analyses requires that the provenance system address a number of challenging issues, such as:

- How do we monitor provenance across a heterogeneous set of machines with different applications running on different operating systems (OSes)?
- How does the provenance system determine causal relations between distributed events, for example whether sensitive file data read from a file server via NFS is sent over the wide-area network using HTTP?
- How do we scale the provenance system to handle increasingly large volumes of data as more systems are monitored at fine granularities?
- How can we provide security properties such as integrity, authenticity and non-repudiation under the assumption that the system is in a partial state of compromise?
- How can we balance competing goals of fault tolerance, performance, and security when implementing distributed tracing?

Provenance systems are frequently exposed to distributed systems concerns, even when the sources of provenance aren't themselves distributed systems. For example, in designing and developing OPUS (Observed Provenance in User Space) (Balakrishnan et al. 2013), we have addressed concerns ranging from the reliable collection and communication of provenance records, to the scalability of graph databases. Drawing upon our experiences of designing, developing and using OPUS this paper will present an overview of the key challenges for provenance in monitoring and analysing APTs.

2. Background

Detection and prevention of APT-style attacks has proven to be particularly challenging, with attackers frequently maintaining access to compromised systems over many months or even years, while hiding their presence within normal background activity. With traditional mechanisms proving ineffective, defensive strategies have increasingly focused on monitoring. When looked at in isolation, it is often difficult to determine whether a given event (commonly referred to as an *Observable*), such as a HTTP connection to a specific host, is malicious or benign. Existing commercial approaches — referred to as Security Information and Event Management (SIEM) systems — provide context to discrete events by integrating security logs from across an enterprise IT system. However, such solutions are notorious for high false-positive rates and imposing a high cognitive burden on human analysts. Further, tracking complex attacker behaviours within a single host, or over many hosts, is challenged by inconsistent trace behaviours on different systems, poor log granularity within hosts, and lack of explicit data-oriented linkage, requiring analysts to manually re-impose causal semantics on traces. We believe that provenance may be able to assist substantially with these problems.

In data-processing pipelines, provenance provides the context for improving confidence in computed outcomes. Likewise in security applications, we believe that the context provided by provenance metadata could be used by analysts in determining why a given event happened, and to help distinguish background (expected) use of the system from potential threat indicators.

For provenance to be effective, it must be captured at the right level of abstraction for attacker operation. For example, if the threat model assumes attacks that target a device's firmware or the System Management Mode (SMM), then an in-kernel provenance system risks missing important events. However, the same system will prove effective in identifying attacks vectored through the compromised networked applications. While surface behaviours resulting from specific exploits may vary significantly (e.g., system-call interactions), many attacker behaviours are intrinsic to APT-style attacks; examples of stable behaviour include: establishing a Command and Control channel and lateral network movement. The stability of these behaviours makes them particularly well-suited to deriving potential *indicators of compromise*. Provenance offers greater access to those intrinsic behaviours – for example; by allowing end-to-end information flows to be tracked, regardless of their specific route or system calls used to propagate the information.

In general, the attacker will seek to provide persistent "hands on keyboard" access to the target network by establishing a Command and Control (C2) channel. *Beaconing* (or calling home) over known C2 channels is frequently monitored (by security mechanisms such as an Intrusion Detection System) as an *indicator of compromise*. Once such an indicator of compromise is detected, there are commonly two questions that security analysts seek to answer:

1. What happened prior to the indicator of compromise? For example: What was the first point of entry? How did the attacker successfully evade earlier attempts at detection?
2. What happened after the initial compromise? For example: What further systems did the attacker access? Did the attacker exfiltrate or delete confidential data?

Answering such questions is essential in order to both improve the organisation's defensive posture and return the system into a trustworthy state.

During APT-style attacks, an attacker may move laterally through the target network seeking to elevate their privileges.

Prior to undertaking lateral movement, attackers frequently orientate themselves within the network by executing common system-administration tools such as `ifconfig` or `nmap`. Port scanning is another commonly employed indicator of compromise. Provenance can help backtrack the attackers lateral movements through the network in order to identify the privileges acquired and ensure that the attacker is denied further access to the target network.

In addition to assisting analysis once an indicator of compromise is detected, provenance might also be used to derive additional indicators of compromise. A security analyst may query provenance metadata to determine uncommon or unique behaviours across the enterprise IT system; for example, unique or uncommonly written files. Similarly, provenance may be able to directly identify potential malicious outcomes, such as exfiltration of sensitive data. Thus provenance metadata can, potentially, guide a detailed forensic analysis in order to identify new indicators of compromise.

Using provenance to support analysis of APT-style attacks faces many of the same challenges as its application to grid computing and data processing workflows: dealing with distributed activities and ensuring security properties such as integrity and non-repudiation. The case for provenance in security has been made a number times before (Muniswamy-Reddy et al. 2006), (Zhou et al. 2010) and (Tan et al. 2013). However, our use case forces significant departures from much existing research in this space:

1. APT indicators of compromise commonly span broad system-level behaviours. Despite several existing provenance system working at the system-level (for example, (Muniswamy-Reddy et al. 2006) and (Gehani and Tariq 2012)), practical challenges remain, for example, providing efficient synchronization and naming primitives.
2. In contrast to monitoring provenance in a single application (such as Hadoop (Akoush et al. 2013)) or applications within a single language runtime (such as Java), system-level provenance systems generate significantly higher volumes of data resulting in potentially significant scalability challenges.
3. Monitoring provenance across a large enterprise IT environment exposes the provenance system to numerous distributed system concerns, such as fault tolerance and performance. These concerns are likely to be in tension with existing approaches of provided security properties such as non-repudiation and integrity (Hasan et al. 2009).
4. It should be assumed that the system is perpetually operating in a state of partial compromise. Compromise of the Trusted Computing Base (TCB) allows an attacker to make progress whilst repudiating their actions. How should provenance be analysed under such assumptions?

These challenges are discussed in more detail in the following section.

3. Practical challenges for provenance in APT monitoring

In this section we outline the key practical challenges that arise in the capture and analysis of provenance for the purposes of monitoring APTs.

3.1 Granularity of provenance

Consider the strategies that an attacker might use for infiltration and setup of a C2 channel. Those should be part of the threat model and inform how much detail is needed from the provenance subsystem: is provenance needed for every byte passing through a CPU? What about DMA transfers for disk and network I/O? Fortunately, *most* indicators of compromise will not be identified at such a low level but higher up in the stack, due to the existence of stable behaviour in the kill chain (delivery of malicious code to OS-facing files, C2 signaling from remote endpoints via OS-provided sockets).

Practical overhead and storage constraints will also dictate a trade-off between the amount of detail captured and perfect identification of low-level malicious behaviours.

At a minimum, the collected provenance should support direct backward and forward queries starting from indicators of compromise. This implies that the raw event log captured has enough information to track where data in the indicators comes from, and, once the intrusion point was detected, what other parts of the system were affected. In most scenarios, this also assumes the ability to query and reconcile events recorded on multiple hosts.

3.1.1 Distributed commit logs

Commit logs (append-only, total-ordered sequences of records) describe what happened and when (Souilah et al. 2009). Logs are widely used in distributed systems to apply changes in a consistent order or to replicate data. As a solution of maintaining consistent views about a large system, they are a data source from which provenance can be derived.

When used as part of an OS-level provenance capturing system, records in the commit log are typically describing changes in the state of kernel abstractions (processes, files, sockets). Enough extra metadata regarding timing and network communications is needed for reconstructing this state at time points in the past, across multiple machines. For example, this means an analyst should be able to get a consistent view of what the process tree was on each machine in a network or what sockets were active 2 days ago.

The challenge lies in integrating distributed commit logs in a secure provenance capture solution: The infrastructure needed to collect them is in itself a new distributed system. Mandatory collection of provenance metadata brings this system into the Trusted Computing Base (TCB)

3.1.2 Naming

Collecting fine grained records comes with the challenge of naming things in a meaningful way: there is a disconnect between the set of things an analyst can point to (packet flagged by an intrusion detection system, a file path on an NFS server) and the kernel-level abstractions identified in the commit log (processes, files, sockets).

The naming of such kernel-level abstractions is more problematic than it may first appear. For example, although it is "natural" to identify files by their path, doing so exposes the provenance system to subversion by attackers exploiting the race conditions present in the construction of filesystem paths. Like process IDs, paths are also insufficiently unique and may refer to unrelated files at different points in time[2].

The first part of a solution consists in providing unique identifiers to every kernel-level object. For example, using version 5 UUIDs with the namespace set to the type of object being named. While this provides supportable semantics, it does not necessarily provide useful semantics. For example, this does not solve the reverse lookup problem: given an object and a partial set of its properties (contents, path, endpoint information), how does one find the names used in the commit log to describe its different versions?

Costs incurred when creating unique identifiers should be proportional to the overall cost of object creation (in both time and space). Ephemeral objects such as IPC messages, for example, must to incur minimal overheads. Fortunately, these objects are confined within a single host and simpler, *locally* unique identifiers should suffice. In contrast, files require globally unique identifiers but they also amortise the cost of identifier creation over a longer period.

3.1.3 Fault tolerance

Failures are an inevitable port of ordinary system operation: application or system overload, reboots for maintenance, full disks, disk failures, kernel crashes, network outages, server reboots, and other disruptions must be tolerated by a distributed provenance system. Further, attackers may themselves go to substantial lengths to induce failure modes not just in the services provided directly by a system, but also its security monitoring and potential provenance capture, which must tolerate both non-malicious and malicious faults.

System-level provenance captures how the state of kernel-level abstractions change over time, for example, recording that process P_1 has forked process P_2. As the kernel has limited capacity to store an unbounded stream of records, provenance must be periodically sent to upstream consumers for either longer-term storage and/or further processing.

In the event of network failures or crashes, responsibility for ensuring the durability of provenance records should be limited to a single entity (the OS kernel). Similarly on recovery, it is the kernel that needs to ensure any locally stored provenance is transmitted while retaining its integrity and completeness. Inevitable tradeoffs between performance (achieved through asynchrony) and fault tolerance (reliant on synchrony) arise. This very much mirrors the problems encountered in databases as they try to maintain ACID properties, but now placed in an adversarial context.

3.1.4 Trust in provenance

Our confidence in this provenance is dependent on the provenance system's ability to answer the following questions:

1. Who created this proposition? And when was the proposition created?
2. Is the creator of this proposition trusted and, more importantly, are they trustworthy?
3. Does the truth of this proposition depend on any other propositions? If so what?
4. As further propositions become available, how does they change our conclusions – especially as we learn retrospectively that provenance sources were compromised?

Provenance-of-provenance allows determining the origin and validity of knowledge expressed within the provenance system. For example, the provenance system may record that a process P_1 created file f_1. System-level provenance can be captured using a variety of techniques, each having different implications for our confidence in that metadata:

Userspace - provenance captured in userspace (for example, through interposition of *library calls* (Balakrishnan et al. 2013)) is essentially a discretionary form of provenance monitoring. Although end-to-end encryption can be used to ensure the integrity and authenticity of provenance records, an attacker can repudiate their actions (by simply bypassing the provenance monitoring).

In-kernel - capturing provenance within the Trusted Computing Base (TCB) of the system enforces mandatory provenance monitoring (actions can only be repudiated by compromising the TCB). Enabling and disabling provenance monitoring (and copying provenance records from the kernel into userspace) typically requires OS privilege (that is, root access). Increasingly security primitives such as TLS are finding application in kernel-space (for example, in order to provide *zero-copy* semantics for webservers (Stewart et al. 2015)). Leveraging such mechanisms supports securing provenance records without further intervention from userspace components.

Below Ring 0 - hardware primitives can potentially support provenance capture in a number of ways. Trusted Computing primitives such as Intel SGX (Software Guard Extensions) can be used provide stronger non-repudiation (even in the presence of a compromised OS). And new hardware primitives could directly support provenance capture, for example providing an append only log for use by the kernel to store provenance records prior to sending over a network.

[2] This is also true for `inode` numbers used by the filesystem to identify a file on disk.

3.2 Provenance analysis

Once the primary data is captured, storing and post-processing it into a format suitable for analysis and query becomes important. From a high level perspective, this amounts to taking the sequence of low-level records in the log and transforming them into a provenance graph, in accordance to *known semantics*. From our perspective, this transformation should be governed by a model such as PVM (Balakrishnan et al. 2013), that formalizes the ways in which an existing graph is modified by each system-level event. For example, if a process forks and then the child writes to a file, the model needs to account for the fact that what is written may be influenced by what the parent read from other files (due to the shared pieces of state between related processes). Those elements have a clear influence on how the provenance graph looks, but are not explicitly present in the kernel event stream.

The model thus also provides guidance about how nodes and relationships in the graph should be interpreted from the perspective of somebody performing queries: a simple model might lead to lots of false-positive relationships in the graph (due to the *n*-by-*m* problem (Carata et al. 2014)); a more complex one could take into account data from taint-tracking or static analysis sources to provide more accurate descriptions while incurring larger overheads.

3.2.1 Provenance views

While the full provenance graph is a valuable output in terms of linking a large number of entities (files, processes, sockets) in a coherent manner, a number of other restricted views of the system can be constructed, either on the fly (i.e the process tree) or maintained as abstracted views given specific requirements (an abstracted view of the provenance graph where host-level details are de-emphasized but inter-host communication is summarized).

Such views are important in a distributed context: different hosts could hold different parts of the overall provenance graph, with others managing abstracted views that allow for hierarchical distributed queries (effectively indexing the available data) or for summary views created in order to facilitate specific tasks (identifying intrusion points, leaked pieces of information or lateral attacker movements). As an example, a view containing nodes representing hosts alongside limited metadata (process trees), and edges showing inter-host network communications can be used to efficiently query subsets of the graph when looking at how an attacker got from an entry point to a host containing target data.

Given the potential of the graph data to occupy significant disk space and other resources, parts of it could also be removed or stored at a coarser granularity. Depending on operational demands, the missing parts may be recreated from the original event stream as needed (if that is kept in compressed format). Typical log storage and rotation options, with fidelity decreasing for older data after analysis has not revealed any interesting activity, could also be applied.

3.2.2 Scalability

One of the main scalability challenges when considering raw event data transformation into provenance graphs is the way the model (in our case, PVM) manages to keep changes to the graph local: this would imply the ability of applying at least some of the events out-of-order, which allows for parallel processing. Processing the kernel-level events in real time poses its own challenges: our experience with using existing graph databases (like Neo4J) for provenance graphs has shown problems regarding data ingestion rates, as well as huge size amplification and memory consumption issues.

As an example, we present a 3GB kernel-level trace collected on a single host over 3 days. It contains 5.5 million events, and we are treating this as one of the smaller traces that we would like to process. Transforming this data into a Neo4J provenance graph in accordance to the initial PVM model leads to a 11GB database with 30 million nodes and 190 million edges. The process also requires more than 30GB of RAM. This single machine case is relevant as we want to maintain proportionality when moving to a distributed setting: each host should be able to keep up with processing its own events. It thus became clear that moving to a distributed system will require a significant rethinking of the overall data ingestion architecture, as well as strategies for coping with a workload that is both read and write-heavy (in graphs, adding a new node involves at least a read to find the location of a parent).

Beyond database optimisations, delaying the construction of the full graph and preferring to compute just of some of its views from the kernel event stream represents one of the possible solutions.

3.2.3 Pluggable versioning models

The use of unique identifiers (UUIDs) during monitoring simplifies provenance analysis. However, as unique identifiers lack any meaningful semantics about the object they identify (*pure names* in distributed-systems terminology) they are difficult for analysts to reason about. This is why the provenance system itself shouldn't be constrained to the identifiers obtained from the OS but be able to derive its own, akin to composite keys in database tables: the UUID coming from the kernel becomes just one attribute through which objects in the provenance graph can be referred to. Others are possible: path names, uuids generated by the model (PVM), hashes of file contents, etc.

When considered as a composite key, the set of object identifier attributes would allow end-users to define their own views on "what a new version means" or "what does uniques or equality mean". Similarly, this scheme allows for scenarios where a named entity is "equivalent" in some sense to another: perhaps it is a handle which represents the entity on the local host (NFS handles) or the relation between a socket and associated kernel-side buffers.

3.2.4 Dynamic trust in provenance

Knowledge involves belief, that is, an attitude that a subject holds towards a proposition. Belief is rarely static, and is influenced by evidence. Our belief in a given result may change over time as we determine that the evidence supporting the data is drawn into question. Unlike conventional uses of provenance, collecting provenance for security analyses must account for the presence of an active attacker. This attacker may seek to subvert the provenance monitoring system in order to disguise their attack. When the attacker's actions come to light, conclusions drawn from provenance (which was trusted but untrustworthy) may be invalid. The provenance system should support analysts in understanding the implications of changes in confidence in provenance.

4. Conclusions and further work

Using provenance metadata for APT monitoring and analysis is particularly challenging due to the number and variety of different applications, language runtimes and OSes that must be monitored. Provenance metadata must be captured at the same granularity as the attacker behaviour being monitored, so-called *indicators of compromise*. Typically indicators of compromise express system-level behaviours such as writing to a file or opening a connection to a remote host. System-level provenance closely matches the use case's requirements for providing broad view of the system behaviour without modifying applications, however the sheer volume of system-level metadata brings significant challenges for scalability.

Monitoring and analysis of distributed system-level provenance systems introduces significant practical challenges. Whilst many of these draw on existing distributed system concepts, existing open source implementations are frequently designed by Internet-scale business and rarely match the specific requirements of a distributed provenance system. Monitoring and analysis of system-level provenance analysis can be simplified through changes to the OS, for example the addition of unique identifiers to kernel-level abstractions. For these such changes to be acceptable to kernel developers, they need to be efficient and proportional to object creation costs (in both time and space).

As part of our work on the CADETS (Causal, Adaptive, Distributed and Efficient Tracing) we will continue adapting the OPUS system to an APT monitoring and analysis use case. As we implement the ideas presented in this paper we expect to further refine our ideas about the design space for provenance in modern, large-scale distributed systems.

Acknowledgments

This work is part of the CADETS Project sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8650-15-C-7558. The views, opinions, and/or findings contained in this paper are those of the authors and should not be interpreted as representing the official views or policies, either expressed or implied, of the Department of Defense or the U.S. Government.

References

S. Akoush, R. Sohan, and A. Hopper. Hadoopprov: Towards provenance as a first class citizen in mapreduce. In *Presented as part of the 5th USENIX Workshop on the Theory and Practice of Provenance*, Lombard, IL, 2013. USENIX. URL `https://www.usenix.org/conference/tapp13/hadoopprov-towards-provenance-first-class-citizen-mapreduce`.

N. Balakrishnan, T. Bytheway, R. Sohan, and A. Hopper. Opus: A lightweight system for observational provenance in user space. In *TaPP*, 2013.

L. Carata, S. Akoush, N. Balakrishnan, T. Bytheway, R. Sohan, M. Seltzer, and A. Hopper. A primer on provenance. *Queue*, 12(3):10:10–10:23, Mar. 2014. ISSN 1542-7730. doi: 10.1145/2602649.2602651. URL `http://doi.acm.org/10.1145/2602649.2602651`.

A. Gehani and D. Tariq. Spade: support for provenance auditing in distributed environments. In *Proceedings of the 13th International Middleware Conference*, pages 101–120. Springer-Verlag New York, Inc., 2012.

R. Hasan, R. Sion, and M. Winslett. The case of the fake picasso: Preventing history forgery with secure provenance. In *Procceedings of the 7th Conference on File and Storage Technologies*, FAST '09, pages 1–14, Berkeley, CA, USA, 2009. USENIX Association. URL `http://dl.acm.org/citation.cfm?id=1525908.1525909`.

E. M. Hutchins, M. J. Cloppert, and R. M. Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1:80, 2011.

K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer. Provenance-aware storage systems. In *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, ATEC '06, pages 4–4, Berkeley, CA, USA, 2006. USENIX Association. URL `http://dl.acm.org/citation.cfm?id=1267359.1267363`.

I. Souilah, A. Francalanza, and V. Sassone. A formal model of provenance in distributed systems. In *First workshop on on Theory and practice of provenance*, page 1. USENIX Association, 2009.

R. Stewart, J.-M. Gurney, and S. Long. Optimizing tls for high–bandwidth applications in freebsd, 2015.

Y. S. Tan, R. K. Ko, and G. Holmes. Security and data accountability in distributed systems: A provenance survey. In *High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on*, pages 1571–1578. IEEE, 2013.

W. Zhou, A. Haeberlen, B. T. Loo, and M. Sherr. Tracking adversarial behavior in distributed systems with secure network provenance. 2010.