AFRL-RY-WP-TR-2019-0337

TRACKING AND ANALYSIS OF CAUSALITY AT ENTERPRISE-LEVEL (TRACE)

Gabriela Ciocarlie
SRI International

MARCH 2020
Final Report

Approved for public release; distribution is unlimited.

See additional restrictions described on inside pages

STINFO COPY

AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with The Under Secretary of Defense memorandum dated 24 May 2010 and AFRL/DSO policy clarification email dated 13 January 2020. This report is available to the general public, including foreign nationals.

Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RY-WP-TR-2019-0337 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

CHARLES P. SATTERTHWAITE, DR-03
Program Manager,
Resilient and Agile Avionics Branch
Spectrum Warfare Division

DAVID G. HAGSTROM, DR-04
Chief, Resilient and Agile Avionics Branch
Spectrum Warfare Division

JOHN F. CARR, DR-04
Chief, Spectrum Warfare Division
Sensors Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

*Disseminated copies will show "//Signature//" stamped or typed above the signature blocks.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
March 2020	Final	16 July 2015 – 20 December 2019

4. TITLE AND SUBTITLE
TRACKING AND ANALYSIS OF CAUSALITY AT ENTERPRISE-LEVEL (TRACE)

5a. CONTRACT NUMBER
FA8650-15-C-7562

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
61101E

6. AUTHOR(S)
Gabriela Ciocarlie

5d. PROJECT NUMBER
1000

5e. TASK NUMBER
N/A

5f. WORK UNIT NUMBER
Y1B2

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

SRI International
333 Ravenswood Ave
Menlo Park, CA 94025-3453

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory
Sensors Directorate
Wright-Patterson Air Force Base, OH 45433-7320
Air Force Materiel Command
United States Air Force

Defense Advanced Research Projects Agency
(DARPA/IAO)
675 North Randolph St.
Arlington, VA 22203

10. SPONSORING/MONITORING AGENCY ACRONYM(S)
AFRL/RYZC

11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S)
AFRL-RY-WP-TR-2019-0337

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with The Under Secretary of Defense memorandum dated 24 May 2010 and AFRL/DSO policy clarification email dated 13 January 2020. This report is available to the general public, including foreign nationals.

14. ABSTRACT
We report on our work in developing the TRACE framework, which combines novel host-level tracking techniques with a proven enterprise-wide tracking system. Specifically, TRACE aimed to enable the detection and investigation of advanced persistent threat (APT) attacks in an enterprise environment using provenance and supports both what-provenance and howprovenance. Our design and implementation provided both logging and provenance propagation primitives. Its host-level provenance tracking component monitors host execution and collects both what- and how-provenance for individual host systems at the granularity of program execution units [22]. The enterprise-wide provenance tracking component builds upon the SPADE engine [10], which has been proven to be scalable and high-performance, and QuickGrail [7], which provides advanced query capabilities. In the course of four years and five engagements, we developed, evaluated, and refined TRACE to provide improvements on performance, scalability, and fidelity. During this time, the system call coverage increased (from 47 to 66 syscalls), while the time and space overhead reduced by over one and two orders of magnitude, respectively. In addition, we found that the TRACE instrumentation stack provided TA2 teams sufficient evidence to detect 80% of the attack stages across all evaluations, being one of the top-performing TA1 systems in the program. Our work was disseminated in 13 top-tier publications (ACSAC 2015, NDSS 2016, ASPLOS 2016, NDSS 2017, Usenix Security, NDSS 2018, Usenix ATC 2018, ACSAC 2018), and received best paper awards at both the Network and Distributed System Security Symposium (NDSS 2016) and Usenix Security 2017. The team also graduated three PhD students who contributed to TRACE.

15. SUBJECT TERMS
advanced persistent threats, provenance tracking, program analysis, network artifacts, program instrumentation

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT:	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON (Monitor)
a. REPORT	b. ABSTRACT	c. THIS PAGE			Charles Satterthwaite
Unclassified	Unclassified	Unclassified	SAR	35	19b. TELEPHONE NUMBER *(Include Area Code)* N/A

TABLE OF CONTENTS

Section **Page**

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

Table **Page**

1. SUMMARY

1.1 Goals

Our objective has been to develop a framework for tagging and tracking of provenance and causality across an enterprise at scale, maintaining a low impact on runtime performance. We created TRACE, which combines novel host-level tracking techniques with a proven enterprise-wide tracking system. Specifically, TRACE aimed to enable the detection and investigation of advanced persistent threat (APT) attacks in an enterprise environment using provenance information.

There are two broad classes of techniques for provenance tracking: audit logging [12], [15], [19], [22], [36] and provenance propagation [5], [6], [16], [17], [29], [35], [37], [40]. Both approaches have pros and cons, and neither is a complete solution for enterprise-wide APT detection and forensics. While audit logging captures both what- and how- provenance; propagation can only capture the former. Audit logging has low runtime overhead because it does not require expensive per-instruction set operations; however, it suffers from low accuracy and high space overhead (for storing event logs). While the propagation-based approaches are more accurate for certain attacks, such as information leak, they suffer from high runtime overhead and difficulty in handling program control dependencies.

TRACE was developed as a practical enterprise-wide provenance-tracking framework that captures both what- and how-provenance by leveraging the advantages of both approaches and overcoming their respective limitations. We designed our system to support the execution of both what- provenance and how-provenance queries. For example, given a corrupted dataset x, two what-provenance queries are: (1) “What is the source/entry point of x?” and (2) “which other files in the enterprise were derived from (and corrupted by) x?” A sample how-provenance query is: “Construct a causal graph showing the events/entities that led to the corruption of x and those that have been further corrupted by x.”

Our design and implementation provided both logging and provenance propagation primitives. Its host-level provenance tracking component monitors host execution and collects both what- and how- provenance for individual host systems at the granularity of program execution units [22]. The enterprise-wide provenance tracking component builds upon the SPADE engine [10], which has been proven to be scalable and high-performance, and QuickGrail [7], which provides advanced query capabilities. It collects provenance from individual entities such as hosts and constructs the distributed enterprise-wide causal graph that can be queried. SPADE provides the underlying distributed storage and processing infrastructure allowing issuance of APT-related queries spanning the whole enterprise.

Throughout the four years and five engagements, we developed, evaluated, and refined TRACE to provide improvements on performance, scalability, and fidelity. During this time, the system call coverage increased (from 47 to 66 syscalls), while the time and space overhead reduced by over one and two orders of magnitude, respectively. In addition, we found that the TRACE instrumentation stack provided TA2 teams sufficient evidence to detect 80% of the attack stages across all evaluations, being one of the top-performing TA1 systems in the program.

Finally, our work was disseminated in 13 top-tier publications (ACSAC 2015, NDSS 2016, ASPLOS 2016, NDSS 2017, Usenix Security, NDSS 2018, Usenix ATC 2018, ACSAC 2018), and received both a Network and Distributed System Security Symposium (NDSS 2016) and a Usenix Security 2017 distinguished paper awards. The team also graduated three PhD students who contributed to TRACE.

1.2 Team

As prime contractor, SRI International (SRI) assembled a four-organization team with a strong record in information flow tracking, program analysis, data provenance, and cyber-attack (including APT) detection and response. Purdue University and University of Georgia offered substantial expertise in program dependence analysis, causality analysis, provenance tracking, audit logging, binary analysis, memory forensic analysis, and advanced execution engine development. University of Wisconsin offered expertise in adapting techniques from programming languages to various problems in information security, such as malware detection and intrusion prevention, and graph databases for efficient provenance querying.

2. INTRODUCTION

Tagging and tracking of provenance and causality across an enterprise at scale and with flexible granularity-while maintaining a low impact on runtime performance requires a combination of core technologies that are integrated in a holistic fashion. TRACE combines novel host-level tracking techniques with a proven enterprise-wide tracking system to achieve this goal.

TRACE enables the detection and investigation of APT attacks in an enterprise environment using provenance information. Provenance captures multiple aspects of information about an entity: what the entity's origin is; how the entity is derived; and when it originated. In the context of APT defense, entities with trackable provenance information are of various granularity; they include processes, network connections, files, and data items within files. The what-provenance of an entity *e* is the set of other entities that have causally influenced *e*'s value/state (e.g., if a file's content comes from a number of network connections, then its provenance contains the IDs of the corresponding sockets); whereas, the how-provenance of entity *e* consists of events and their causal ordering - organized as a causal graph - demonstrating how.

The primary objective of TRACE is the application of scalable and fine-grained information-flow tracking techniques for real-time, enterprise-wide APT detection. Although an impressive body of academic research has laid the foundations for information-flow tracking and provenance analysis [5], [6], [16], [34], [39], and their applications for malware detection at the enterprise-level [8], [11], [13], [14], [17], [23], [29], [35], such systems have not been widely deployed due to intrinsic limitations that arise from intractability and imprecision of these techniques at scale. To further motivate TRACE, we first discuss the related solutions to program-level causality tracking. Such solutions mainly fall into two categories: audit logging and provenance propagation.

Host-Level Audit Logging. Solutions in this category [11], [14], [17], [35] record system-level events (e.g., syscalls) during execution and then causally connect events during attack investigation. They treat processes as subjects; files, sockets, and other passive entities as objects; and assume causality between subjects and objects involved in the same syscall event (e.g., a process reading a file). Audit logging is a built-in function in Linux that incurs much lower overhead than per-instruction provenance propagation. There are also techniques [8], [13], [23], [29] that focus on the analysis of causal relationships between the captured events. These techniques propose various optimizations in building causal graphs, narrowing down attack-relevant events, and querying events from a large amount of system event data. While their optimizations can improve the effectiveness of attack investigation, they still suffer from the limitations of audit logging: dependence explosion and storage overhead.

(1) *Dependence explosion* [21] is a major limitation of audit logging. For a long-running process, an output event is assumed to be causally dependent on all preceding input events, and an input event is assumed to have causal influence on all subsequent output events. Such conservative assumptions create excessive false-positive causal relations, making it difficult to reveal the true causality.

(2) *High storage overhead.* A preliminary study [22] shows that audit logging easily generates gigabytes of log data per host every day. This is particularly problematic for APT defense, as APT malware tends to lurk in the victim host for a long period of time.

(3) *Offline analysis.* Prior work that introduced [21] and refined [25] the notion of an execution unit requires both a forward and backward pass over the logs. This precludes their use in the streaming setting, where online analysis must be performed for real-time detection.

Per-Instruction Provenance Propagation/Tracking (I-PT). I-PT techniques [5], [6], [16], [34], [36], [39] involve fine-grained causality tracking by monitoring the execution of individual instructions. Their application to host-wide causality tracking in production environments has been hindered by the following limitations.

(1) *Substantial runtime overhead.* Because they track the execution of individual instructions and propagate (potentially) large provenance sets, I-PT techniques usually incur substantial runtime overhead. State-of-the-art implementations without hardware support incur a slow-down [16], which is undesirable for production environments.

(2) *Lack of implicit flow handling.* Many I-PT techniques have difficulty handling implicit flow, which is information flow through control dependences [27] (usually induced by program predicates). A naive solution that propagates provenance via all control dependences may lead to a high false-positive rate (up to 97% for SPEC2000INT programs [4]), with the consequence that each output is causally related to almost all inputs.

(3) *Inflexible provenance granularity.* I-PT techniques [5], [6], [16], [34], [36], [39] often assume one default provenance granularity (e.g., each input byte, input syscall, or file). However, APT defense may require multiple provenance sources with varying granularity (e.g., individual emails instead of an entire inbox file).

Network-level Provenance. There is a line of work in network-level provenance [37][38][40][41] that focuses on distributed systems. In particular, [37] monitors and records causal dependencies in the SDN (software defined networks) environment. TRACE supports both host-level and network-level provenance, providing comprehensive causal relationships across systems. As an infrastructure, TRACE and existing provenance tracking tools are complementary. In other words, one may leverage existing provenance tracking techniques to enhance the capabilities of TRACE.

3. METHODS, ASSUMPTIONS, AND PROCEDURES

3.1 Host Provenance Tracking

Many long-running programs share a common property: their execution is dominated by an event-handling loop, which usually handles an independent request on each loop interaction. This is confirmed by our study of more than 100 open-source applications, including 51 server applications (e.g., web, mail, media, ssh-daemon, remote desktop, version control) and 43 client-side applications (e.g., browser, email client, multimedia, messenger). They are written in various languages (C, C++, Java, Python) and may involve threads/processes. Hence, our host monitoring component incorporates a new technique called unit-based selective instrumentation (UBSI) that builds on the notion of a unit [21]. Specifically, it identifies borders and dependences between such loop iterations – defined as execution units – so that causal graphs may be constructed that contain only causally related entities/events.

First, static analysis is performed offline to recognize the execution and data unit structures of a program. Next, inter-unit dependences are identified. Selective program points (e.g., unit boundaries or unit dependence-inducing instructions) are then instrumented. During execution, system calls and unit-related events are captured by the provenance tracking runtime.

Reverse Engineering Unit-Inducing Loops. To detect the unit-inducing loops, UBSI leverages three observations: (1) such loops tend to be at the top level; (2) their loop bodies must make some I/O system calls; and (3) their loop bodies dominate the execution time. We have developed dynamic analysis techniques using PIN tool [24] to pinpoint unit-inducing loops in applications [20]. We then use source code and binary instrumentation techniques [20] to instrument the loop entry and exit points such that special log entries are generated to indicate unit boundaries.

Reverse Engineering Inter-Unit Dependencies. In some cases, a unit by itself may not fully cover the sub-execution that handles an independent input. Instead, a few inter-dependent units together constitute a semantically independent sub-execution. In practice, there are memory dependencies across unit boundaries. However, only some of them – called workflow dependencies – are helpful in connecting units that belong to the same sub-execution. Examples include the dependencies caused by the enqueue and dequeue operations of a task queue. In prior work [21], inter-unit dependencies were identified via a number of training runs of the target binary, which suffers from incomplete discovery of dependencies. To remedy this problem, we developed a static analysis method to detect shared data structures accessed by multiple threads. It identifies and instruments the instructions that induce dependencies through the statically identified shared structures. The interdependent units can then be detected and clustered.

Notable Improvements. The following improvements were made to UBSI to support the needs of TRACE:

- Source-code-based UBSI was integrated with SPADE by extending its Audit Reporter module to identify unit boundaries and unit dependences at runtime. This technique reduced the space and post-processing overhead. Specifically, the most recent version of Firefox

(version 54.0.1) at the time was updated with unit instrumentation to support JavaScript execution and Firefox extensions.

- Memory consumption in SPADE's Audit Bridge was optimized, allowing us to maintain a large number of units and their dependencies in memory. A runtime caching technique was developed to store memory access records. It allows us to directly emit unit dependence information instead of recording every memory access history.
- In addition, UBSI was refined to reduce the volume of events generated while still capturing the dataflow between units. In Engagement 1, UBSI reported read and write events on memory by units. The mentioned events were used to identify dependencies between units. Due the nature of the Firefox application, there were too many read and write events by units. This is shown in Figure 1(a). In Engagements 2 and 3, an abstraction of unit dependency was developed. SPADE's Audit Reporter was extended to automatically convert read and write into unit dependencies events between units, as shown in Figure 1(b). Although UBSI was still responsible for reporting the memory read and write events, this greatly reduced the number of final provenance events from UBSI, as shown in Table 1. In Engagements 4 and 5, a final update was made. We migrated unit dependency identification upstream to UBSI instrumentation. This significantly reduced the runtime overhead. Specifically, we cached memory read and write events, and only reported confirmed dependencies between units. This eliminated unnecessary inter- process communication (IPC) between instrumented applications and the Linux Audit subsystem. Additionally, the update in Engagement 4 reports only the last dependency between any two units to further minimize the runtime and space overhead. This also simplified the provenance graph, as shown in Figure 1(c).

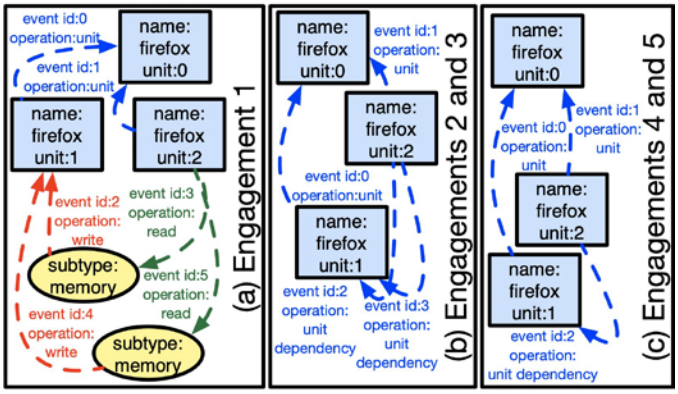

Figure 1. UBSI Activity Representation

Table 1. Time and Space Overhead Measurements of UBSI Across Engagements Using Firefox 54.0.1

Benchmarks	Time Overhead		Space Overhead	
	Eng. 1-3	Eng. 4-5	Eng. 1-2	Eng. 3-5
JetStream2.0 [1]	351%	22.6%	6,952%	7.1%
Octane2 [2]	414%	16%	3,398%	10%
SpeedoMeter2.0 [3]	591%	15.99%	1,603%	11.94%
Average	452%	18%	3,984%	10%

3.2 Enterprise-wide Provenance Tracking

SPADE [9] is an open source software infrastructure that provides support for provenance auditing in distributed environments. TRACE uses it as an integration framework. SPADE uses a graph-based data model consisting of vertices and directed edges, each of which can be labeled with an arbitrary number of annotations. The model uses the following two vertex and five edge types from the Open Provenance Model (OPM) [31] ontology:

- **Two Vertex Types:** (1) Controlling *Agent*, executing *Process* (blue rectangles in Figure 1), and (2) *Data Artifact* (yellow ovals in Figure 1).
- **Five Edge Types:** Defining (1) which process *used* which artifact (green arrows in Figure 1), (2) which artifact *wasGeneratedBy* which process (red arrows in Figure 1), (3) which process *wasTriggeredBy* which other process (blue arrows in Figure 1), (4) which artifact *wasDerivedFrom* which other artifact, and (5) which process *wasControlledBy* which agent.

TRACE extends the ontology to handle the constructs described in the following sections. SPADE has been architected to decouple the collection, filtration, storage, and utilization of provenance metadata, as illustrated in Figure 1. A novel **provenance kernel** mediates between producers and consumers of provenance information, and handles the persistent storage of the records. The kernel handles buffering, filtering, and multiplexing incoming metadata from multiple **provenance reporters** via a non-blocking interface; supports multiple **provenance stores**; and responds to concurrent queries from provenance consumers. The kernel also supports modules that operate on the stream of provenance graph elements, allowing the aggregation, fusion, and composition of provenance elements to be customized with **provenance filters** [10].

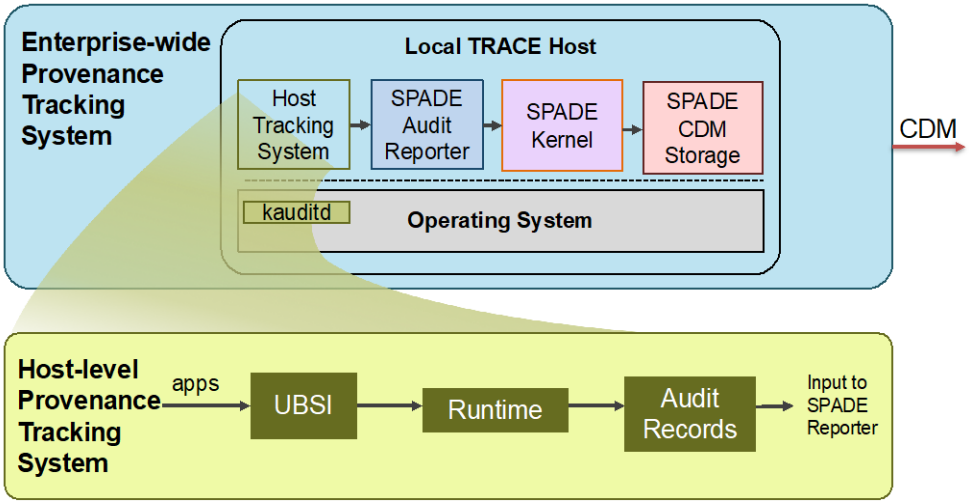

Figure 2 Overview of the Components and Dataflow in the TRACE Provenance Tracking System

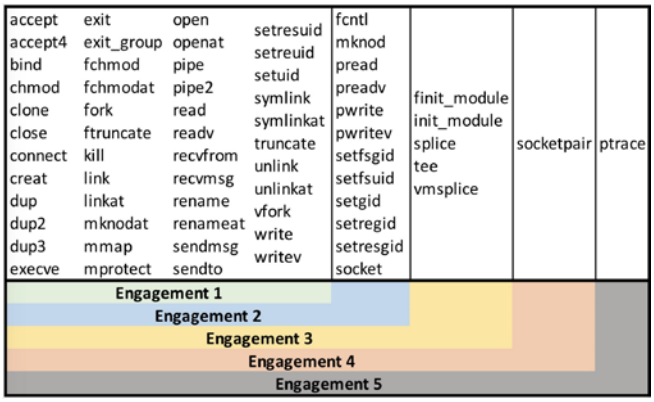

Figure 3. System Calls Captured by TRACE System from Engagement 1 (47) to Engagement 5 (66)

The TRACE system uses two sources of information about system activity that were used to output provenance in a Common Data Model (CDM) format. The two sources include:

- **Linux Audit.** This component of the operating system kernel was configured to log all system calls specified in Figure 3. Calls from all processes across the host were audited except for those that are executed by the TRACE system.
- **UBSI Instrumentation.** Linux Audit does not provide information regarding the dataflows through memory in and between threads. This dataflow was captured by UBSI instrumented programs. UBSI instrumentation divides a program into individual components called units that are iterations of program loops. It then audits reads and writes by units of memory locations shared between threads.

SPADE's Audit Reporter was extended to process the two sources (mentioned above) to generate provenance. A single audit event received contains the following information:

- **Event.** The system call number, arguments, and return value.
- **Process.** Relevant process identifiers, including pid and ppid.
- **User.** Identifiers related to process ownership, such as uid, gid, euid and egid.
- **Supplementary Information.** Optional information per event, such as a remote network address or a filesystem path.

SPADE's Audit Reporter uses the information in the event records to build an overview of the system in terms of the following provenance objects:

- **Process.** The subject or object of the system call.
- **Unit.** The subject performing read or write on a memory address.
- **Artifact.** The objects affected by the system call. The objects through which dataflow was reported were: (1) **Local Filesystem Artifacts**: regular file, named pipe, unix socket; (2) **Memory**: Address allocated in a thread or affected by a unit; (3) **Network**: Local or remote address; (4) **IPC**: unnamed pipe; and (5) **Unidentified**: Unknown objects whose type could not be determined.

SPADE's Audit Reporter maintains state for each of the provenance objects and internal mappings between them to generate an accurate representation that reflects the model in the operating system kernel.

Integrating Host Activity. Our approaches abstract portions of the information flow graph, emit provenance at multiple resolutions, and use information flow labels for units of various types. To handle such fine-grained provenance elements generated by the *unit-based selective instrumentation*, we extended SPADE's Audit Reporter.

Each Reporter module in SPADE utilizes the same interface to the SPADE Kernel. The domain semantics are captured as annotations on the vertices and edges. For example, existing reporters transform the audit logs of macOS, Linux, Android, and Windows into provenance event streams by emitting OPM vertices with annotations about users, processes, and files, and edges with annotations of types and other details of performed operations.

Persistent Storage. The kernel commits the integrated provenance (and subsequently retrieves when necessary) through a uniform provenance storage interface that allows graph elements to be inserted and retrieved. Each storage subsystem implements this interface and leverages its native functionality to best implement the required functions. For example, the implementation of the storage interface for the Neo4j graph database [33] leverages the underlying system's native support for path queries to answer provenance queries. The implementation of the storage interface for JDBC-compliant databases, such as MySQL and H2 [32], uses self-joins to provide the equivalent functionality.

Network Awareness. SPADE supports provenance queries about distributed computation. It models a network connection as a pair of **network artifacts** connected by used and *wasGeneratedBy* edges. Network artifacts are distinguished by the property that each endpoint can independently construct the same artifact without explicit coordination. This property allows for complete decentralization of provenance collection, while still ensuring that subgraphs from different hosts can be reassembled. SPADE implements network artifacts with this property by combining the time the connection was initiated with the IP addresses and TCP or UDP ports of the two endpoints.

The system is completely decentralized; each computer maintains the authoritative repository of the provenance gathered on it. Flows between processes, files, and network connections are recorded at each host, and the resulting metadata is stored locally. Applications are oblivious to the provenance collection and metadata distribution.

Space/Performance Optimizations. First, a number of abstractions are applied to the events generated by UBSI-instrumented application to reduce the volume of data published. Abstracted relationships were reported after all underlying events were processed. Although such abstraction can eliminate the assurance of total ordering between events, our evaluation over five adversarial engagements demonstrates that it does not impact the forensics capability of TRACE. By reporting the abstracted event in place of the last underlying event, the detection logic can be sure that all relevant underlying events have already been processed. To simplify analysis, we made updates in our system to report abstracted relations as early as possible. Second, unit-related records were previously stored using statically allocated buffers during stream processing in SPADE's Audit

Bridge (between Linux Audit and SPADE's Reporter). This was changed to use dynamic sizes, which dramatically reduced the memory footprint of the corresponding process. In addition, de-duplication strategies were implemented in data structures to reduce memory usage. Third, each process can be controlled by a different agent, letting its data structure be linked to a separate instance tracking its user, group, etc. In practice, the diversity of controlling agents over time is low. This was exploited by defining a new kinetic data structure (that associates agents with spans of temporal operation). This significantly reduced unit-related memory requirements.

Accuracy and Fidelity Improvements. To enable cross-host information flow tracking, we implemented network artifacts and used iptables to generate Linux Audit network filter records. When network connections are established (through application system calls), local endpoint details are not reported in the system call records of Linux Audit logs. Cross-host tracking requires this information for forensics across hosts. We developed a new Linux kernel module, called *netio*, that collects local endpoint details and reports them via the Linux kernel's Audit subsystem. SPADE's Audit Reporter was updated to (i) detect and report when the *uid* of a process in a system call record differs from the last known *uid* of that process; (ii) process the local endpoints reported by the *netio* kernel module; (iii) collect host configuration information from the runtime and report it; (iv) add support for reporting when file paths relate to directories, links, character devices, and block devices.

Finally, specific processes, such as those associated with managing application workloads and collecting telemetry, were whitelisted. Recovery logic was added to handle incomplete unit-related records that may arise during abstraction of heap memory reads / writes when constructing inter-unit dependencies.

3.3 Other Relevant Work

For the host-level provenance tracking, our team explored other relevant approaches (Figure 4) that led to publications in top-tier conferences:

- **Multiple Perspective attack Investigation (MPI)** [26]: MPI uses a semantics-aware program annotation and instrumentation technique to partition execution based on the application specific high-level task structures. It avoids training, generates execution partitions with rich semantic information, and provides multiple perspectives of an attack. Specifically, MPI takes the annotations and automatically instruments (a large number of) program locations that denote unit boundaries through static program analysis. The analysis handles complex threading models in which the executions of multiple tasks/units interleave. The instrumentation emits special syscalls upon unit context switches so that the application-specific task/unit semantics is exposed to the underlying provenance tracking systems. MPI allows annotating multiple task/unit structures simultaneously so that a forensic analyst can inspect an execution from multiple perspectives (e.g., tab and domain perspectives for Firefox).
- **Lightweight Dual Execution (LDX)** [18]: Causality inference, such as dynamic taint analysis, determines whether an event *e* is causally dependent on a preceding event *c* during execution. Given an execution, LDX spawns a slave execution, in which it mutates *c* and observes whether any change is induced at *e*. To preclude nondeterminism, LDX couples the executions by sharing syscall outcomes. To handle path differences induced by the

perturbation, we developed a novel on-the-fly execution alignment scheme that maintains a counter to reflect the progress of execution. The scheme relies on program analysis and compiler transformation.

- **Model-based Causality Inference (MCI)** [19]: We developed a model-based causality inference technique for audit logging that does not require any application instrumentation or kernel modification. MCI leverages LDX that can infer precise causality between system calls, but requires doubling the resource consumption such as CPU time and memory consumption. For each application, we use LDX to acquire precise causal models for a set of primitive operations. Each model is a sequence of system calls that have interdependences, some of them caused by memory operations and hence implicit at the system-call level. These models are described by a language that supports various complexity such as regular, context-free, and even context-sensitive. In production run, a novel parser is used to parse audit logs (without any enhancement) to model instances and hence derive causality.

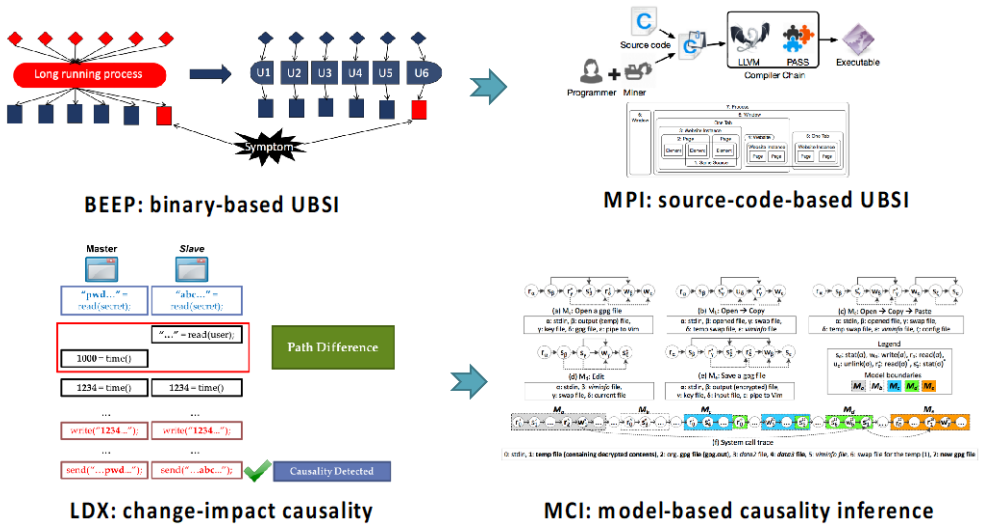

Figure 4. Different Approaches Explored During the Program on Host-Level Provenance Tracking

The TRACE team also explored approaches for cost-effective audit logging:

- **Kernel-supported Cost-effective Audit Logging (KCAL)** [25]: The Linux Audit system is widely used as a causality tracking system in real-world deployments for problem diagnosis and forensic analysis. TRACE also relies on the Linux Audit system. However, its performance could be improved. We performed a comprehensive analysis on the Linux Audit system and found that it suffers from high runtime and storage overheads due to the large volume of redundant events. To address these shortcomings, we proposed KCAL, an in-kernel cache-based online log-reduction system to enable high-performance audit logging. It features a multi-layer caching scheme distributed in various kernel data structures, and uses the caches to detect and suppress redundant events. Our technique is designed to reduce the runtime overhead caused by transferring, processing, and writing logs, as well as the space overhead caused by storing them on disk. Compared to existing log reduction techniques that first generate the huge raw logs before reduction, our technique avoids generating redundant events at the first place.

4. RESULTS AND DISCUSSION

During the DARPA program, our system was evaluated in a series of five engagements. During each engagement, attacks were launched by the TA5 team and were analyzed by TA2 performers. We summarize the outcome of the engagements below to highlight the efficacy and utility of the TRACE system in uncovering APT attacks. The goal of each attack was to surreptitiously exfiltrate information from the host being targeted. All attacks involved a composition of the following components:

- **Host Access.** Two strategies were leveraged to gain unauthorized access to TRACE hosts.
    - **Stolen Credentials.** Two applications (ssh and VNC) were used to access the host using stolen credentials.
    - **Vulnerable Applications.** Synthetic vulnerabilities were inserted that were then exploited to gain access to the host using Firefox and the Pine email client.
- **Reconnaissance.** Different attacks used combinations of following approaches to gather host information (i.e., executing commands and malicious programs, reading and writing system files on the host and scanning for other hosts to exploit):
    - **System utilities used to gather system information.** whoami, hostname, uname, ifconfig, netstat, lsusb, lspci, lshw, date, openssl, ps, tcpdump, arp, sysinfo, route and lsmod.
    - **System files read or written.** /etc/hosts, /etc/network/interfaces, /proc/cpuinfo, /proc/meminfo, /proc/version, /etc/passwd, /etc/shadow and /etc/sudoers.
    - **Malicious programs downloaded by the attacker and executed.** A network port scanner was downloaded to gather information about other hosts in the subnetwork.
- **Host Hijacking.** The following methods were used to execute attacker's programs for enabling reconnaissance.
    - **System variable update.** Benign programs were tricked into loading a malicious shared library by setting the LD PRELOAD variable to the malicious shared library.
    - **In-memory malicious code execution.** Vulnerable programs were exploited by the attacker to execute malicious code in memory. Two programs were targeted by the red team in the engagements: Firefox and pine.
    - **Code injection.** ptrace system call was used to inject malicious code into a process either by writing to its memory or by loading a shared library into a process.
- **Privilege Escalation.** A Linux kernel module was loaded into the host by the red team, prior to each engagement, that enabled non-root processes to gain root privilege without user authentication. Whenever any process interacted with the kernel module in a special way, the interacting process gained root privilege. An example of this was the kernel module named elevateme. When this kernel module was loaded, it created the character device /dev/elevateme. If some program in user space wrote to the device, the kernel module checked if the string written matched AAAA. If it matched, then it changed the privilege of the process to root by calling commit creds (prepare kernel cred(0)).

Table 2 presents the overall performance evaluation results across all engagements. **ff-down**: Downloaded file using firefox; **scp-down**: Copied to host using scp; **email- down**: Downloaded automatically as a result of opening an email in a vulnerable email client; **direct-exec**: Executed from filesystem manually; **mmap-exec**: Executed in memory address space of a hijacked process; **auto-exec**: Executed automatically from filesystem because of a vulnerability in an

application; **reconn**: Activity performed by attacker to gather host information; **conn-in**: Attacker gains access to the host; **conn-out**: Attacker exfiltrated information from the host; **priv-escal**: Attacker escalated the privileges of a hijacked process on host to root **Coloring**: **white**: Detected by at least one team; **light green**: Not detected by any team but present in data; **red**: Not detected by any team and not found in data; **orange**: Not detected because SPADE was killed; gray: Not detected by any team and not present in data. TA5 team hypothesized cause was external misconfiguration.

Table 2. Overall Performance Evaluation

Eng.	Attack	Access Method	1	2	3	4	5	6	7
1	Bovia	Firefox	conn-in 1/3	ff-down 2/3	direct-exec 2/3	reconn 2/3	conn-out 2/3		
	Pandex	ssh login	scp-down 1/3	conn-in 2/3	direct-exec 2/3	reconn 0/3	conn-out 0/3		
	Bovia-Stretch	Firefox	conn-in 2/3	ff-down 2/3	direct-exec 2/3	priv-escal 1/3	mmap-exec 2/3	reconn 2/3	conn-out 2/3
2	Bovia-Simple	ssh login	conn-in 3/3	direct-exec 3/3	reconn 3/3	conn-out 3/3			
	Pandex-Drakon	Firefox	conn-in 0/3	ff-down 0/3	mmap-exec 0/3	priv-escal 1/3	reconn 0/3	conn-out 0/3	
	Pandex-Micro	Firefox	conn-in 2/3	ff-down 2/3	direct-exec 2/3	reconn 2/3	conn-out 2/3		
	Drakon-Netrecon	Firefox	conn-in 0/3	ff-down 0/3	mmap-exec 0/3	priv-escal 0/3	reconn 1/3	conn-out 0/3	
3	Drakon-In-Memory	Firefox	conn-in 1/3	mmap-exec 1/3	priv-escal 2/3	ff-down 1/3	direct-exec 1/3	reconn 1/3	conn-out 2/3
	Drakon-Pass-Manager	Firefox extension	conn-in 0/3	ff-down 1/3	direct-exec 1/3	priv-escal 1/3	reconn 1/3	conn-out 1/3	
	Phishing-Email-Link	Firefox	conn-in 0/3	reconn 1/3	conn-out 0/3				
	Phishing-Attachment	Pine	conn-in 0/3	email-down 2/3	auto-exec 2/3	reconn 2/3	conn-out 2/3		
4	Azazel-A	ssh login	conn-in 2/2	direct-exec 0/2	reconn 1/2	conn-out 1/2			
	Azazel-B	ssh login	conn-in 0/2	direct-exec 0/2	reconn 0/2	conn-out 0/2			
	Drakon-Ptrace	Firefox	conn-in 0/2	priv-escal 1/2	mmap-exec 0/2	reconn 1/2	conn-out 1/2		
	Drakon-Lib-Inject	Firefox	conn-in 0/2	ff-down 1/2	priv-escal 1/2	mmap-exec 1/2	direct-exec 1/2	reconn 1/2	conn-out 1/2
	VNC	VNC login	conn-in 0/2	scp-down 1/2	direct-exec 1/2	reconn 1/2	conn-out 1/2		
	Metasploit	ssh login	scp-down 1/2	conn-in 1/2	direct-exec 2/2	reconn 0/2	conn-out 1/2		
	Stolen-Credential	ssh login	scp-down 0/2	conn-in 0/2	priv-escal 1/2	direct-exec 0/2	reconn 0/2	conn-out 0/2	
	Drakon	Firefox	conn-in 0/2	priv-escal 1/2	mmap-exec 1/2	reconn 1/2	conn-out 1/2		
5	Multiple-Performers	ssh login	conn-in 0/1	direct-exec 0/1	reconn 0/1	conn-out 0/1			
	Drakon	Firefox	conn-in 0/1	mmap-exec 0/1	priv-escal 0/1	mmap-exec 0/1	reconn 0/1	conn-out 0/1	
	Azazel	ssh login	scp-down 1/1	conn-in 0/1	direct-exec 0/1	reconn 0/1			

4.1 Adversarial Engagement 1

TRACE was run on Ubuntu 14.04 64-bit host. The host was setup with a vulnerable Firefox web browser (version 42) and a kernel driver to simulate a privilege escalation vulnerability. For the first adversarial engagement, our system was configured to track 47 system calls and collected 111 GB of data in a period of 4 days, capturing 1,485,144,886 events.

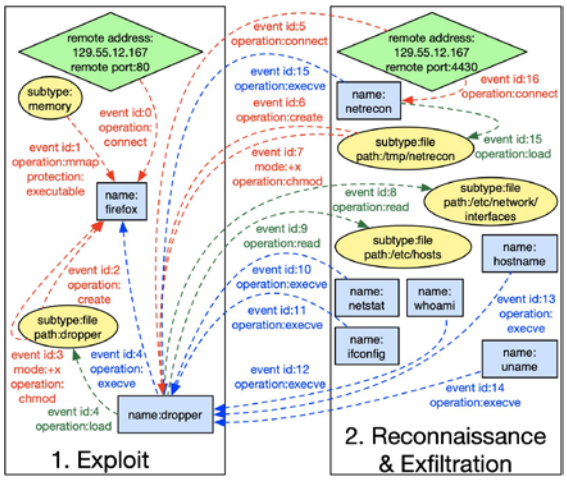

Figure 5. Normalized Bovia Attack Graph for Engagement 1

1) *Bovia:* Firefox navigated to a compromised website. The compromised website exploited a vulnerability in Firefox and took control of the Firefox process. The hijacking of the Firefox process was not captured by TRACE because it was done in memory; TRACE can see only allocation of memory, not its modification. The hijacked process downloaded a malicious binary and executed it. The malicious program connected to the attacker, which acted as a reverse-shell. The attacker gathered user, host, and network information closed the channel.

■ *All stages of the attack were present in TRACE data. Of the three TA2 teams, one was able to detect all 5 stages in the attack, a second team was able to identify 4/5 attack stages, and the third team did not detect this attack.*

2) *Pandex:* The attacker used ssh to log into the host using stolen credentials. This connection was used to install Dropbear SSH server, which was used for future connections by the attacker. The attacker later connected to the Dropbear SSH server on the compromised host to gather system information and exited after that. Activity by the attacker to gather host information was not detected and was not present in the data along with its exfiltration.

■ *All stages of the attack were present in TRACE data except the reconnaissance and the exfiltration by the attacker. Of the three TA2 teams, one was able to detect all three stages present in the data. The second team failed to detect the copying of the Dropbear SSH server to host, but detected its installation. The third team did not detect any stage of the attack.*

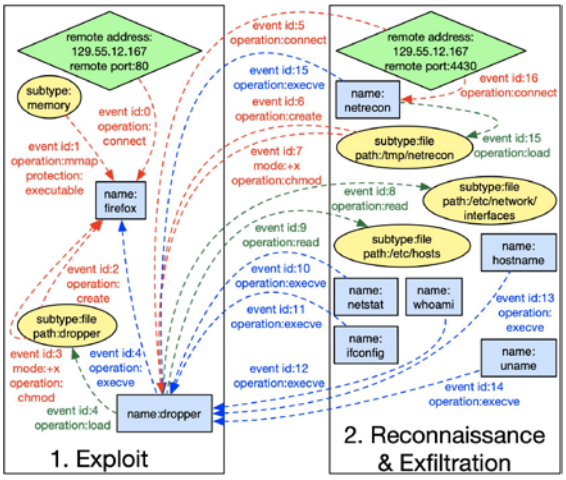

Figure 6. Normalized Pandex Attack Graph for Engagement 1

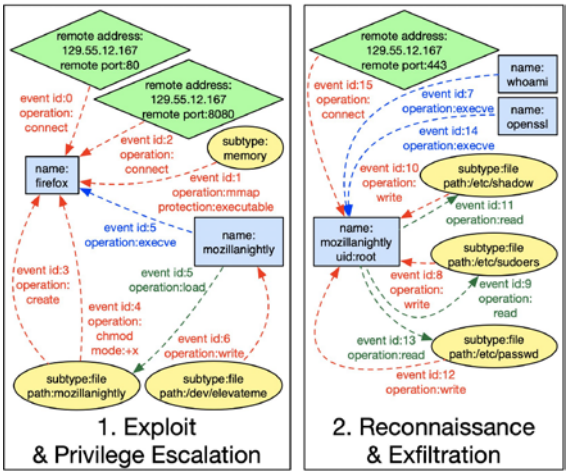

Figure 7. Normalized Bovia-Stretch Attack Graph for Engagement 1

3) *Bovia-Stretch*: Firefox navigated to a compromised website that exploited a vulnerability in Firefox and took control of the Firefox process. The hijacked process downloaded a malicious binary and then executed it. The malicious program connected to the attacker and acted as a reverse-shell. In this attack, unlike the Bovia attack, the malicious binary gained root privileges first by writing to a file. The file was created by a kernel module that was installed as benign setup for this attack. The attacker used a malicious program to read and write system files like /etc/passwd, texttt/etc/sudoers and /etc/shadow and closed the channel.

■ *All stages of the attack were present in TRACE data. Of the three TA2 teams, one was able to detect all stages in the attack, a second team was able to identify 6/7 attack steps, and the third team did not detect the attack.*

Engagement-1 Summary: TRACE provided evidence of 15/17 attack steps across 3 attacks. At least one of the TA2 teams identified each of those 15 steps.
Lessons: Based on these results, we identified 12 critical missing system calls and determined the need to optimize UBSI output to have direct connections between units for memory reads

and writes. We also concluded that an event-centric representation of provenance was more needlessly verbose.

4.2 Adversarial Engagement 2

TRACE monitored 59 system calls, and the total data collected for the three teams was 1.78 GB, 4.62 GB, and 1.46 GB respectively. This was a significant reduction from Engagement 1 and a testament to the optimizations that were implemented. The improvements made to the TRACE system included the following:

- **System Calls.** Twelve more system calls were identified and audited to generate a richer provenance graph. In addition, tracking of connect system call was improved. Specifically, the connect system call can complete without a definitive answer about whether it was successful or not. In Engagement 1, these ambiguous connect system calls were being ignored as failed system calls. Now provenance was reported for these ambiguous connect system calls also in favor of over-reporting rather than under-reporting. Performance
- **Enhancements.** An update was made in the UBSI instrumentation to report only dependencies between units rather than reporting the read of a memory address by a unit that was previously written to by a different unit. This, in theory, reduced the number of Linux Audit records generated by TRACE system at most by 50%.
- **Linux Filesystem**. In the first engagement, the permissions of Linux files were not reported in provenance unless permissions were updated directly through a system call. From this engagement onwards, permissions of files were reported always, which enabled us to detect any change in permissions by the red team in a secret way. In addition, filesystem paths were normalized to remove special symbols "." and ".." to simplify the generated provenance. This reduced the effort required by the analysis teams to connect dataflows.
- **Network Provenance.** TRACE was extended to include provenance tracking of UDP network traffic.

1) *Bovia-Simple*: This attack was the same as the Bovia attack from the first engagement.

■ *All stages of the attack were present in TRACE data. All three TA2 teams detected all 4 stages of the attack.*

2) *Pandex-Drakon*: In this attack, Firefox navigated to a compromised website. The compromised website exploited a vulnerability in Firefox and took control of the Firefox process. Firefox was hijacked by allocating executable memory, writing the malicious code to it, and transferring execution to it. The hijacked Firefox process then elevated its privileges to root. The elevated Firefox process did reconnaissance and exfiltrated that information to the attacker over the network. This attack went mostly undetected despite it being a variant of the Bovia-Stretch attack from Engagement 1, which was detected by analysis teams. The stages of the attack which were not detected by any team were also not found in the data. Retrospective analysis by the adversarial team led them to conclude that the reason for this miss was most likely in- correct configuration (process/user whitelisting) of the TRACE system.

■ *2/6 stages of the attack were present in the TRACE data. Of the three TA2 teams, only one team detected only the privilege escalation stage of the attack while the other two teams detected no stage of the attack. The red team hypothesized that the attack stages missed due to missing data were likely due to misconfiguration of the TRACE system.*

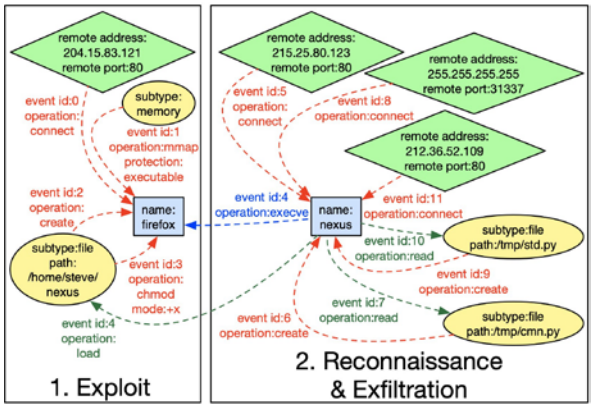

Figure 8. Normalized Pandex-Micro Attack Graph for Engagement 2

3) *Pandex-Micro*: Firefox navigated to a compromised website. The compromised website exploited a vulnerability in Firefox and took control of the Firefox process. The hijacked process downloaded a malicious binary and executed it. The malicious program connected to the attacker and acted as a reverse-shell. The attacker wrote python scripts to the host and executed them. The python scripts gathered network information and exfiltrated the /etc/passwd file to the attacker.

■ *All stages of the attack were present in TRACE data. Of the three TA2 teams, two teams detected all 5 stages of the attack. The third team did not detect any attack stages.*

4) *Drakon-Netrecon*: Firefox navigated to a compromised website. The compromised website exploited a vulnerability in Firefox and took control of the Firefox process. Firefox was hijacked by allocating executable memory, writing the malicious code to it and transferring execution to it. The hijacked Firefox process was seen writing to a file and, as a result, the process gained root privilege. This happened because of the kernel module installed on the host as part of benign activity. The hijacked process gathered system information such as process listings and host information, allocated executable memory again, and gathered network information from the system and sent that the attacker. Only one TA2 team was able to partially detect this attack. After looking in the data provided, we were able to detect the download of the malicious binary, escalation of privilege by the Firefox process, and the exfiltration but not the source of the attack, i.e., when the attacker exploited Firefox.

■ *4/6 attack stages were present in the TRACE data. Of the three TA2 teams only one team detected only the reconnaissance stage of the attack. Only one team was able to detect the reconnaissance stage of the attack, while none of the two other teams detected any stages of the attack.*

Engagement-2 Summary: TRACE provided evidence of 15/21 attack stages across 4 attacks. At least one of the TA2 teams identified 10 of those13 attack steps.
Lessons: We identified 5 critical missing system calls: splice, tee, and vm_splice for dataflow tracking and finit_module and init_module to detect kernel-module insertions. We found that elimination of explicit memory representation was insufficient for succinctly describing intra-process dependencies. This motivated the need for abstraction of multi-edges.

4.3 Adversarial Engagement 3

The following improvements were made to the TRACE system for this engagement.

- **System Calls**. Five more system calls were identified and audited.
- **Linux Filesystem**. Prior to Engagement 3, dataflow through directories, links, character devices, and block devices was reported as dataflow through Linux files. The change was made to capture dataflow in a more refined way by specifying the exact type. This enabled us to see that the red team was escalating process privileges using a character device.
- **Process Management (IPC, User Tracking, Whitelisting).** In prior engagements, SPADE's Audit Reporter did not keep any internal mapping about IPC objects through which Linux process data could be shared, e.g., Process A talking to child Process B, through a common unnamed pipe. In the resulting provenance graph, this would have been represented as Process A and Process B doing I/O on separate vertices in the graph i.e., disconnected. Though by using the information in the provenance graph, the analysis teams could draw a direct connection between Process A and Process B. In Engagement 3, an update was made to keep the mapping and draw a direct connection between Process A and Process B in SPADE's Audit Reporter.
  Some of the TA5 attacks involved the change of the user of a process to root in order to execute privileged activity. This was being done without the use of a direct system call. In prior engagements, SPADE's Audit Reporter generated provenance showing two unconnected processes, referring to the same Linux process, with different user identifiers. SPADE's Audit Reporter was updated to start monitoring for a change in the user identifier and start drawing a direct connection between the two processes in the provenance generated.
  In Engagement 2, it was noticed that Xvnc application used by the red team to perform attacks on the hosts caused significant spike in audit records. Since, the Xvnc application was control traffic (not categorized as part of the attack or benign activity), an ability was added in the SPADE's Reporter to whitelist such processes by name.
- **Network Provenance.** The native tools being used in the TRACE system did not always provide complete network dataflow information. For example, the local port of a network channel is not reported in a UDP transfer. To cover this gap, a kernel module was added to SPADE's Audit Reporter that hooked system calls related to network I/O (sendto, recvfrom, sendmsg, recvmsg, bind, connect, accept and accept4 to report complete information of a network endpoint.

1) *Drakon-in-Memory:* Firefox navigated to a compromised website. The compromised website exploited a vulnerability in Firefox and took control of the Firefox process. Firefox was hijacked by allocating memory, writing the malicious code to it and transferring execution to it.

The hijacked Firefox process was seen writing to a character device and, as a result, the process gained root privileges. This happened because of the kernel module installed on the host as part of benign activity. The hijacked process downloaded a malicious binary and executed it. The malicious program gathered system information and sent it to the attacker.

■ *All 7 stages of the attack were present in TRACE data. Of the three TA2 teams, one team detected all stages of the attack. The second team detected 2/7 stages, and the third team did not detect any stages of the attack.*

2) *Drakon-Pass-Manager*: Firefox password manager extension was exploited by the attacker. The compromised extension downloaded a malicious binary and executed it. The malicious program connected to the attacker, which provided the attacker with a console. The attacker ran portscan and sent the information to the attacker. The connection which initiated the attack was not present in the data.

■ *5/6 stages of the attack were present in TRACE data. Of the three TA2 teams, one team detected all stages of the attack present in the TRACE data. The other two teams did not detect any stages of the attack.*

3) *Phishing-Email-Link:* Host fetched an email which was a phishing link. The user opened the email, navigated to the attached link, and entered confidential information, which was sent to the attacker. The connection that initiated the attack and the connection that exfiltrated the data were not present in the data.

■ *1/3 stages of the attack were present in TRACE data. Of the three TA2 teams, one team detected the reconnaissance stage of the attack, which was present. The other two teams did not detect any stages of the attack.*

4) *Phishing-Email-Attachment*: Pine fetched an email from the attacker with a malicious binary attached. The attached malicious binary was downloaded and executed due to a vulnerability in the pine client. The malicious program connected to the attacker, ran portscan, and sent the information to the attacker. The connection which initiated the attack was not present in the data.

■ *4/5 attack stages were present in TRACE data. Of the three TA2 teams, two teams detected all the present stages of the attack in the TRACE data. The third team did not detect any attack stages.*

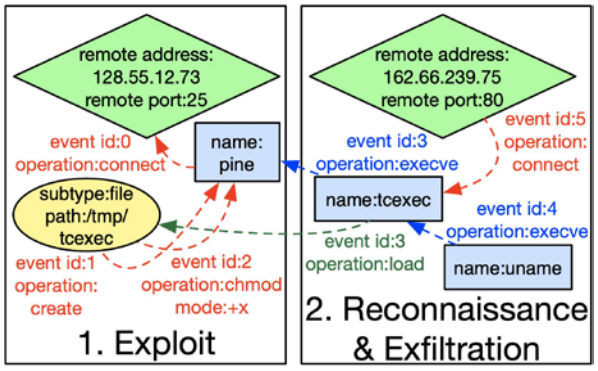

Figure 9. Normalized Phishing-Attachment Attack Graph for Engagement 3

Engagement-3 Summary: TRACE provided evidence of 17/21 attack steps across 4 attacks. At least one of the TA2 teams identified each of the 17 attack steps.
Lessons: Firefox 54's compartmentalized design led to significantly higher socket-based communication, motivating the need to track the socketpair syscall.

4.4 Adversarial Engagement 4

The following improvements were made to TRACE system for this engagement.

- **System Calls.** One additional system call, socketpair, was identified to be audited. The socketpair system call was added to track the flow of data between host processes through unnamed socket pairs. This was a refinement in the data model; i.e., the source and sink of the dataflow were now identified as unnamed socket pairs. Prior to this change, this dataflow was still captured, but it was not reported that the dataflow was through an unnamed socket pair. In addition, flags for clone system call were reported to aid analysis teams. This enabled the analysis teams to detect the information shared between host processes like memory area, file descriptor table, etc.
- **Performance Enhancements.** SPADE's Audit Reporter keeps internal data structures to retain state of processes running on the host. The life-cycle events of host processes (process start and process terminate) were used to do garbage collection in the internal data structures of SPADE's Audit Reporter.
  Prior to Engagement 4, the UBSI instrumented Firefox application reported dependency between two units each time they shared data through memory. This was changed to report only the last dependency between any two units. The effect of this change was to generate only one record instead of multiple records for dependencies between units.

1) *Azazel A/B:* This attack involved two hosts: Host A and Host B. Host A was exploited first, and then Host A was used to connect to Host B to exfiltrate information from it.

The attacker used ssh to log into Host A using stolen credentials. A malicious library was downloaded and added to the 'LD PRELOAD' shell variable. The update of the variable was not visible in TRACE output because no system call was used, but its effect was seen when a process loaded the library into its memory. For the attack, that program was nc. The hijacked nc process connected to the attacker without the user being aware. The attacker gathered system information and exits. Two hosts were involved in this attack. The attacker connected to Host A directly and

then connected to Host B from Host A. The connection between Host A and Host B was not in the data. No team detected the execution of the malicious binary on Host A and Host B, but it was present in the data provided. The reconnaissance activity on Host B was present in the data but was not detected by any team.

■ *All 4 stages of the attack were present in TRACE data. Of the two TA2 teams, one team detected 3/4 stages and the other team detected 1/4 stages of the attack.*

On Host B, the attacker logged in using ssh and a malicious binary was executed to exfiltrate host information.

■ *2/4 stages of the attack were present in TRACE data. Of the two TA2 teams, none of the teams detected any stages of the attack.*

2) *Drakon-Ptrace*: Firefox navigated to a compromised website. The compromised website exploited a vulnerability in Firefox and took control of the Firefox process. Firefox was hijacked by allocating memory, writing the malicious code to it, and transferring execution to it. The hijacked Firefox process was seen writing to a character device, and, as a result, the process gained root privileges. This happened because of the kernel module installed on the host as part of benign activity. The hijacked Firefox process used ptrace system call to inject shellcode into the sshd process. As a result of injected code into sshd process, it opened a connection to the attacker. The attacker gathered system information using the hijacked sshd process and exit. The initial connection was not present in the data. In this engagement, the red team used a new code injection technique for in-memory execution, which used the ptrace system call. TRACE was not auditing the ptrace system call, and therefore it went undetected.

■ *3/5 stages of the attack were present in the TRACE data. One TA2 team detected 3/5 stages of the attack, while the second team did not detect any stages of the attack.*

3) *Drakon-Lib-Inject*: Firefox navigated to a compromised website. The compromised website exploited a vulnerability in Firefox and took control of the Firefox process. Firefox was hijacked by allocating memory, writing the malicious code to it, and transferring execution to it. The hijacked Firefox process was seen writing to a character device, and, as a result, the process gained root privileges. This happened because of the kernel module installed on the host as part of benign activity. A Linux shared object file was downloaded by the hijacked process, which was then injected into the sshd process. The sshd process opened a connection to the attacker, gathers system information, and exit. The initial connection was not detected by any team because it was not present in the data. In this engagement, the red team used a new code injection technique for in-memory execution, which used the ptrace system call to make the target process load a malicious library. The ptrace event was not detected because TRACE was not auditing the ptrace system call, but its effect, the loading of the malicious library, was captured.

■ *6/7 stages of the attack were present in the TRACE data. One of the TA2 teams detected 6/7 stages of the present, while the second team did not detect any stage of the attack.*

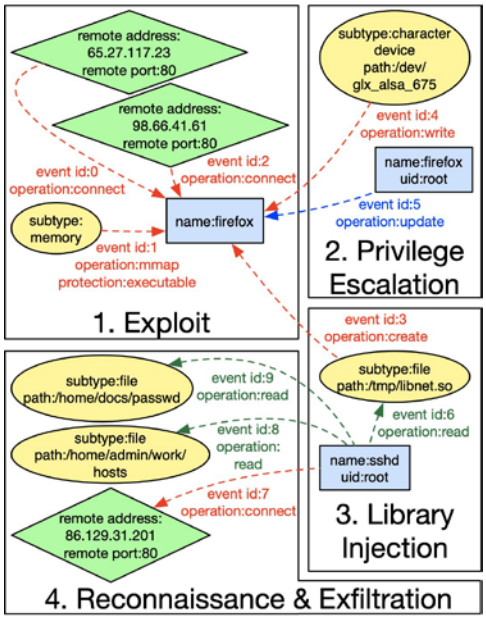

Figure 10. Normalized Drakon-Lib-Inject Attack Graph for Engagement 4

4) *VNC:* The attacker used vnc to log into the host using stolen credentials. A ransomware was downloaded and executed with sudo. The ransomware encrypted user files in the home directory. The connection that initiated the attack was not detected by any team because it was not present in the data.

■ *4/5 stages of the attack were in the TRACE data. One of the two TA2 teams detected all 4/5 stages of the attack present while the second team did not detect any stages of the attack.*

5) *Metasploit:* Metasploit was used to copy a malicious binary on the host. The malicious binary was executed and opened a connection to the attacker. The attacker gathered system information and exit. The reconnaissance activity by the attacker was not detected by any team because it was not present in the data.

■ *4/5 stages of the attack were in the TRACE data. One of the two TA2 teams detected all 4/5 stages of the attack present. The second team detected only one stage of the attack.*

6) Stolen-Credential-Attack: The attacker used ssh to log into the host using stolen credentials from another compromised machine. The attacker executed a malicious program called nodeup. The nodeup process used the kernel module (described previously) to elevate the privileges to root. The attacker killed SPADE and then performed system information to send to the attacker. But since SPADE was killed, no activity was recorded beyond escalation of privileges. The connection that initiated the attack was not detected by any team because it was not present in the data. The analysis teams also did not detect the copying of the malicious executable, but it was found in the data. The execution of the malicious executable, the reconnaissance, and the data exfiltration were all missed because as part of the attack the red team killed SPADE.

■ 2/6 stages of the attack were in the TRACE data. One of analysis teams was able to detect one stage of the attack, while the second team did not detect any attack stages.

7) *Firefox-Drakon:* Firefox navigated to a compromised website. The compromised website exploited a vulnerability in Firefox and took control of the Firefox process. Firefox was hijacked by allocating memory, writing the malicious code to it, and transferring execution to it. The hijacked Firefox process was seen writing to a character device, and, as a result, gained root privileges. This happened because of the kernel module installed on the host as part of benign activity. The hijacked Firefox process used ptrace system call to inject shellcode into the sshd process. As a result of injected code into sshd process, it opened a connection to the attacker. The attacker gathered host information using the hijacked sshd process and exit. The connection that initiated the attack was not detected by any team because it was not present in the data.

■ *4/5 stages of the attack were in the TRACE data. One of the two analysis teams detected all 4/5 stages of the attack present. The second team did not detect any stages of the attack.*

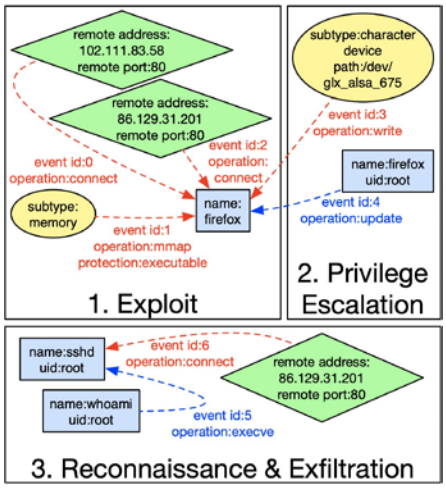

Figure 11 Normalized Drakon Attack Graph for Engagement 4

Engagement-4 Summary: TRACE provided evidence of 30/41 attack steps across 8 attacks. At least one of the TA2 teams identified 26/30 attack steps.
Lessons: 3/11 missed steps were attributed to targeted attacks by APT launched against TRACE. However, there was no self-protection implemented for this evaluation which made the system vulnerable to targeted attacks. Significant performance improvements were realized by moving unit-dependency upstream into Firefox. We determined that tracking ptrace calls was needed to detect code injections into sshd by the adversarial team.

4.5 Adversarial Engagement 5

The following improvements were made to the TRACE system for Engagement 5.

- **System Calls.** One additional system call, ptrace, was identified to be audited, bringing the total to 66. In Engagement 4, one of the hijack methods used by the red team was the use of ptrace system call to inject code into another process. Since TRACE system was

not auditing ptrace system call, the hijack went undetected, but the analysis teams were able to detect the attack based on other features. To address this gap, the auditing of the ptrace system call was added for Engagement 5. Despite the addition, no TA2 team detected the attack that used this hijack method, but, using the red-team report, we were able to find most of the attack graph.

- **Performance Enhancements.** When improving the performance of SPADE's Audit Reporter, a bottleneck was identified – the serialization and deserialization of internal data structures. In SPADE's Audit Reporter, internal data structures are backed by a persistent storage on disk as they can grow arbitrarily because of the nature of the data. The persistent storage necessitated the serialization and deserialization of data. Prior to Engagement 5, the serialization and deserialization approach being used was the default Java approach; i.e., using the Serializable interface. In Engagement 5, this was updated to use a custom serialization and deserialization approach. As a result of the new approach, on average, we observed 324% reduction in serialization time, 1436% reduction in deserialization time, and 854% reduction in the size of the serialized data.
- **Self Protection.** In Engagement 4, one of the TA5 team's attacks involved the termination of the TRACE system to prevent the analysis teams from detecting the attack. They succeeded in their goal; i.e., the TRACE system was terminated and the attack went undetected. To prevent this in Engagement 5, a feature was added to the TRACE system to prevent the termination of any component of the TRACE system unless authenticated. This prevented the attacker from terminating any TRACE system processes or unloading the SPADE kernel modules.

1) *Multiple-Performers*: The attacker used ssh to log into the host using stolen credentials from another compromised machine. The attacker copied /etc/passwd and /etc/hosts files and connected to another host. This attack was not detected by any team, but the complete attack was found in the data.

■ *All attack stages (4/4) were present in the TRACE data. There was only one TA2 team for this engagement and none of the stages were detected by them.*

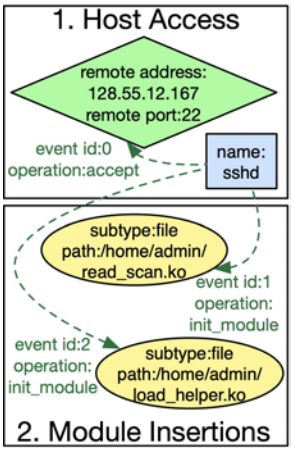

Figure 12. Normalized Attack Setup Graph for Engagement 5

2) *Firefox-Drakon*: Firefox navigated to a compromised website. The compromised website exploited a vulnerability in Firefox and took control of the Firefox process. Firefox was hijacked by allocating memory, writing the malicious code to it, and transferring execution to it. The hijacked Firefox process gained root privileges using a new technique, which was not detected. However, it was seen that the hijacked Firefox process' uid changes to '0'. The hijacked process injected shellcode into the sshd process using ptrace. The hijacked sshd process read /etc/passwd file and other system information. This attack was not detected by any team, but the attack was found partially in the data. The in-memory execution was not found along with the reconnaissance activity by the attacker.

■ *4/6 attack stages were present in the TRACE data. There was only one TA2 team, and none of the stages were detected by them.*

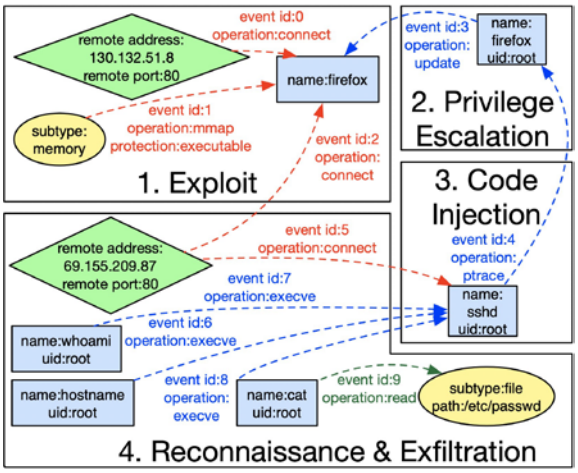

Figure 13. Normalized Drakon Attack Graph for Engagement 5

3) *Azazel:* The attacker used ssh to log into the host using stolen credentials. A malicious library was downloaded and added to the 'LD PRELOAD' shell variable. The update of the variable was not visible in TRACE output because no system call was used, but its effect was seen when a process loaded the library into its memory. For the attack, that program was nc. The intention was to have the nc process connect to the attacker without the user being aware. However, the hijacked nc process failed to connect to attacker and could not gather system information. The analysis teams did detect the copying of the malicious library to the host, but they did not detect any more activity on the host by the attacker. The initial connection by the attacker and the use of the malicious library was present in the data. The attacker failed to do reconnaissance, and therefore it was neither detected nor present in the data.

■ *3/4 stages of the attack were present in the TRACE data. There was only one TA2 team and it detected only one stage of the attack.*

Engagement-5 Summary: TRACE provided evidence of 11/14 attack steps across 3 attacks. There was only TA2 team and that was able to identify only 1/11 attack steps present in the data. 1/3 missed steps were attributed to a failed attack by APT launched against TRACE.
Lessons: There was a significant improvement in resilience due to self-protection techniques implemented prior to this evaluation.

5. CONCLUSIONS

The TRACE team developed, demonstrated, and delivered a wide range of deliverable results under the DARPA Transparent Computing Program. We substantially improved the state of the art by producing novel research and tools for provenance tracking at for enterprise-wide APT detection in real time. Specifically, TRACE is a highly-scalable system for stream-based, enterprise-wide provenance tracking that integrates three powerful capabilities: (i) unit-based instrumentation, (ii) distributed causality tracking, and (iii) efficient graph-based query analytics. During the course of the four-year project, TRACE was subject to a series of five adversarial engagements where the TA5 team launched APT attacks. The streaming CDM output produced by TRACE was fed as input to three TA2 analysis teams that implemented real-time APT detection logic. The TRACE instrumentation stack provided valuable forensic evidence to detect over 80% of the attack stages across all evaluations and our improvements led to multiple orders of magnitude reduction in time and space overheads for unit instrumentation. Our project produced:

- **Data:** As part of the five engagements, we produced provenance data for realistic APT scenarios, all of which will be available to the research community. This report details the findings for all the engagements and how our data was used by the TA2 teams in detecting the attacks performed by the TA5 team.
- **Software:** We produced a TRACE software package, which was part of the continuous integration framework developed by the TA3 team. During the five engagements, our system proved to be robust in terms of both performance and attack resilience. The main TRACE components are open source:
    - UBSI: https://github.com/kyuhlee/UBSI
    - SPADE: http://spade.csl.sri.com
- **Publications:** Our work was disseminated in 13 top-tier publications (ACSAC 2015, NDSS 2016, ASPLOS 2016, NDSS 2017, Usenix Security, NDSS 2018, Usenix ATC 2018, ACSAC 2018), and received both a Network and Distributed System Security Symposium (NDSS 2016) and a Usenix Security 2017 distinguished paper awards. Two other publications are in the review process.
    1. Accurate, Low Cost and Instrumentation-Free Security Audit Logging for Windows, Annual Computer Security Applications Conference (ACSAC 2015)
    2. ProTracer: Towards Practical Provenance Tracing by Alternating Between Logging and Tainting, Network and Distributed System Security Symposium (NDSS 2016), Distinguished Paper Award.
    3. LDX: Causality Inference by Lightweight Dual Execution, Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2016)
    4. Hercule: Attack story reconstruction via community discovery on correlated log graph, Annual Conference on Computer Security Applications (ACSAC 2016)
    5. DroidForensics: Accurate Reconstruction of Android Attacks via Multi-layer Forensic Logging, ACM Asia Conference on Computer and Communications Security (ASIA CCS 2017)

6. J-Force: Forced Execution on JavaScript, International World Wide Web Conference (WWW 2017)

7. Enabling Reconstruction of Attacks on Users via Efficient Browsing Snapshots, Network and Distributed System Security Symposium (NDSS ’17)

8. A2C: Self-Destructing Exploit Executions via Input Perturbation, Network and Distributed System Security Symposium (NDSS 2017)

9. MPI: Multiple Perspective Attack Investigation with Semantic Aware Execution Partitioning, USENIX Security Symposium (Security 2017), Distinguished Paper Award.

10. Kernel-Supported Cost-Effective Audit Logging for Causality Tracking, 29th USENIX Annual Technical Conference (ATC 2018)

11. JSgraph: Enabling Reconstruction of Web Attacks via Efficient Tracking of Live In- Browser JavaScript Executions, Network and Distributed System Security Symposium, (NDSS ’18)

12. MCI: Modeling-based Causality Inference in Audit Logging for Attack Investigation, Network and Distributed System Security Symposium (NDSS 2018)

13. Lprov: Practical Library-aware Provenance Tracing, Annual Conference on Computer Security Applications (ACSAC 2018)

6. REFERENCES

[1] Jetstream 2. https://browserbench.org/JetStream/index.html. Accessed: 2019-9-23.

[2] Octane 2.0. https://chromium.github.io/octane/. Accessed: 2019-9-23.

[3] Speedometer 2.0. https://browserbench.org/Speedometer2.0/. Accessed: 2019-9-23.

[4] T. Bao, Y. Zheng, Z. Lin, X. Zhang, and D. Xu. Strict control dependence and its effect on dynamic information flow analyses. In Proceedings of the 19th International Symposium on Software Testing and Analysis, ISSTA '10, pages 13–24, New York, NY, USA, 2010. ACM.

[5] E. Bosman, A. Slowinska, and H. Bos. Minemu: The world's fastest taint tracker. In Proceedings of the 14th International Conference on Recent Advances in Intru- sion Detection, RAID'11, pages 1–20, Berlin, Heidelberg, 2011. Springer-Verlag.

[6] J. Clause, W. Li, and A. Orso. Dytan: A generic dynamic taint analysis framework. In Proceedings of the 2007 International Symposium on Software Testing and Analysis, ISSTA '07, pages 196–206, New York, NY, USA, 2007. ACM.

[7] J. Fan, A. G. S. Raj, and J. M. Patel. The case against specialized graph analytics engines. In CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings, 2015.

[8] P. Gao, X. Xiao, Z. Li, K. Jee, F. Xu, S. R. Kulkarni, and P. Mittal. Aiql: Enabling efficient attack investigation from system monitoring data. In Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference, USENIX ATC '18, pages 113–125, Berkeley, CA, USA, 2018. USENIX Association.

[9] A. Gehani and D. Tariq. SPADE: Support for Provenance Auditing in Distributed Environments. In 13th ACM/IFIP/USENIX International Conference on Middleware, 2012.

[10] A. Gehani, D. Tariq, B. Baig, and T. Malik. Policy-based integration of provenance metadata. In 12th IEEE International Symposium on Policies for Distributed Systems and Networks, 2011.

[11] A. Goel, K. Po, K. Farhadi, Z. Li, and E. de Lara. The taser intrusion recovery system. In Proceedings of the twentieth ACM symposium on Operating systems principles, SOSP '05. ACM, 2005.

[12] P. Groth, S. Miles, W. Fang, S. C. Wong, K. P. Zauner, and L. Moreau. Recording and using provenance in a protein compressibility experiment. In HPDC'05, July 2005.

[13] M. N. Hossain, S. M. Milajerdi, J. Wang, B. Eshete, R. Gjomemo, R. Sekar, S. D. Stoller, and V. N. Venkatakrishnan. Sleuth: Real-time attack scenario reconstruction from cots audit data. In Proceedings of the 26th USENIX Conference on Security Symposium, SEC'17, pages 487–504, Berkeley, CA, USA, 2017. USENIX Association.

[14] X. Jiang, A. Walters, D. Xu, E. H. Spafford, F. Buchholz, and Y.-M. Wang. Provenance-aware tracing ofworm break-in and contaminations: A process coloring approach. In Proceedings of the 26th IEEE International Conference on Distributed Computing Systems, ICDCS '06. IEEE Computer Society, 2006.

[15] M. G. Kang, S. McCamant, P. Poosankam, and D. Ong. DTA++: Dynamic taint analysis with targeted control-flow propagation. In A. Perrig, editor, NDSS 2011, 18th Annual Network & Distributed System Security Symposium, Washington, DC, USA, Feb. 2011. Internet Society.

[16] V. P. Kemerlis, G. Portokalidis, K. Jee, and A. D. Keromytis. Libdft: Practical dynamic data flow tracking for commodity systems. In Proceedings of the 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments, VEE '12, pages 121–132, New York, NY, USA, 2012. ACM.

[17] S. T. King and P. M. Chen. Backtracking intrusions. In Proceedings of the nineteenth ACM symposium on Operating systems principles, SOSP '03. ACM, 2003.

[18] Y. Kwon, D. Kim, W. N. Sumner, K. Kim, B. Saltaformaggio, X. Zhang, D. Xu , LDX: Causality Inference by Lightweight Dual Execution, In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2016.

[19] Y. Kwon, F. Wang, W. Wang, K. H. Lee, W.-C. Lee, S. Ma, X. Zhang, D. Xu, S. Jha, G. F. Cretu-Ciocarlie, A. Gehani, and V. Yegneswaran. MCI: Modeling- based causality inference in audit logging for attack investigation. In NDSS, 2018.

[20] M. Laurenzano, M. Tikir, L. Carrington, and A. Snavely. Pebil: Efficient static binary instrumentation for linux. In Performance Analysis of Systems Software (ISPASS), 2010 IEEE International Symposium on, pages 175 –183, march 2010.

[21] K. H. Lee, X. Zhang, and D. Xu. High accuracy attack provenance via binary-based execution partition. In Proceedings of the 20th Annual Network and Distributed System Security Symposium, NDSS'13, 2013.

[22] K. H. Lee, X. Zhang, and D. Xu. Loggc: Garbage collecting audit log. In CCS'13, 2013.
[23] Y. Liu, M. Zhang, D. Li, K. Jee, Z. Li, Z. Wu, J. Rhee, and P. Mittal. Towards a timely causality analysis for enterprise security. In NDSS, 2018.
[24] C. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. Reddi, and K. Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. In PLDI'05, pages 190–200, 2005.
[25] S. Ma, J. Zhai, Y. Kwon, K. H. Lee, X. Zhang, G. Ciocarlie, A. Gehani, V. Yegneswaran, D. Xu, and S. Jha. Kernel-supported cost-effective audit logging for causality tracking. In 2018 USENIX Annual Technical Conference (USENIX ATC 18), pages 241–254, Boston, MA, July 2018. USENIX Association.
[26] S. Ma, J. Zhai, F. Wang, K. H. Lee, X. Zhang, and D. Xu. MPI: Multiple perspective attack investigation with semantic aware execution partitioning. In 26th USENIX Security Symposium, 2017.
[27] W. Masri, A. Podgurski, and D. Leon. Detecting and debugging insecure information flows. In Proceedings of the 15th International Symposium on Software Reliability Engineering, ISSRE '04, pages 198–209, Washington, DC, USA, 2004. IEEE Computer Society.
[28] S. McCamant and M. D. Ernst. Quantitative information flow as network flow capacity. In Proceedings of the 2008 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '08, pages 193–205, New York, NY, USA, 2008. ACM.
[29] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. N. Venkatakrishnan. Holmes: Real-time apt detection through correlation of suspicious information flows. 2019 IEEE Symposium on Security and Privacy (SP), pages 1137–1152, 2019.
[30] S. Miles, P. Groth, M. Branco, and L. Moreau. The requirements of recording and using provenance in e-science experiments. Journal of Grid Computing, 2006.
[31] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan, and J. V. den Bussche. The Open Provenance Model core specification (v1.1). Future Generation Computer Systems, 27(6), 2011.
[32] T. Mueller. H2 database engine. http://www.h2database.com.
[33] I. Neo Technology. Neo4j. http://neo4j.com.
[34] F. Qin, C. Wang, Z. Li, H.-s. Kim, Y. Zhou, and Y. Wu. Lift: A low-overhead practical information flow tracking system for detecting security attacks. In Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 39, pages 135–148, Washington, DC, USA, 2006. IEEE Computer Society.
[35] S. Sitaraman and S. Venkatesan. Forensic analysis of file system intrusions using improved backtracking. In Proceedings of the Third IEEE International Workshop on Information Assurance, IWIA '05. IEEE Computer Society, 2005.
[36] D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena. Bitblaze: A new approach to computer security via binary analysis. In Proceedings of the 4th International Conference on Information Systems Security, ICISS '08, pages 1–25, Berlin, Heidelberg, 2008. Springer-Verlag.
[37] H. Wang, G. Yang, P. Chinprutthiwong, L. Xu, Y. Zhang, and G. Gu. Towards fine- grained network security forensics and diagnosis in the sdn era. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18, pages 3–16, New York, NY, USA, 2018. ACM.
[38] Y. Wu, M. Zhao, A. Haeberlen, W. Zhou, and B. T. Loo. Diagnosing missing events in distributed systems with negative provenance. In Proceedings of the 2014 ACM Conference on SIGCOMM, SIGCOMM '14, pages 383–394, New York, NY, USA, 2014. ACM.
[39] M. Zhang, X. Zhang, X. Zhang, and S. Prabhakar. Tracing lineage beyond relational operators. In Proceedings of the 33rd international conference on Very large data bases, VLDB '07. VLDB Endowment, 2007.
[40] W. Zhou, Q. Fei, A. Narayan, A. Haeberlen, B. T. Loo, and M. Sherr. Secure network provenance. In Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, SOSP '11, pages 295–310, New York, NY, USA, 2011. ACM.
[41] W. Zhou, M. Sherr, T. Tao, X. Li, B. T. Loo, and Y. Mao. Efficient querying and maintenance of network provenance at internet-scale. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, SIGMOD '10, pages 615–626, New York, NY, USA, 2010. ACM.

7. LIST OF ACRONYMS, ABBREVIATIONS, AND SYMBOLS

APT	Advanced Persistent Threat
IPC	Inter-Process Communication
I-PT	Per-Instruction Provenance Propagation/Tracking
KCAL	Kernel-supported Cost-effective Audit Logging
MCI	Model-based Causality Inference
MPI	Multiple Perspective attack Investigation
OPM	Open Provenance Model
SPADE	Support for Provenance Auditing in Distributed Environments
SDN	Software Defined Networks
TA	Technical Area
TRACE	TRacking and Analysis of Causality at Enterprise
UBSI	Unit-Based Selective Instrumentation