Completeness-Security Gap in LSM-based provenance capture mechanisms?

Aarti Kashyap *Electrical and Computer Engineering*
University of British Columbia
Vancouver, Canada
kaarti.sr@gmail.com

***Abstract*—Information flow control at the Operating system level has been an active area of interest for a long time. Using information flow control techniques we can describe how data came to be in its present form by gathering metadata that describes the history of each object being processed on the system. This is called as data provenance. Data provenance has many uses, from forensics and security to aiding the reproducibility of scientific experiments. The existing implementations for Linux are based on the Linux Security Modules (LSM) framework which implements hooks at specific points to control operations on kernel objects and a set of opaque security fields in kernel data structures for maintaining security attributes. However, while past work on verification of LSMs addressed the reliability of information flow control system built on LSMs, no one has addressed the security of information flow control system built on LSMs. In this work, we show that the current LSM interface captures all security-related flows in the kernel. Next, given the provenance captured at these points, we explore if the security violation shows up as an anomaly in the provenance graph.**

***Index Terms*—Data provenance, Whole-system provenance, Linux kernel**

I. Introduction

System security is race between the attackers and the defenders. The attackers, adapt their attack model based on the defense mechanisms being deployed on the systems. The designers of the systems can build a completely correct system using formal methods such as theorem proving and model checking [4]. However, this only proves the correctness properties of the systems such as " Making sure that the system is following the correct protocol " or " the shared memory allocation is done efficiently". The attacker can make sure that there is no violation in these properties and still manage to attack the system. In order to beat the attacker in this game, several security based mitigation techniques are proposed, which lack complete security coverage. [5]. Hence, in order to obtain a full or a wider coverage of the system, using provenance-based techniques is an ideal way to proceed. [6]

Provenance has many different definitions when used in different contexts. The simplest way to define provenance is a formal set of documents, to understand the beginning of something's existence and origin. These documents can be used to guide authenticity or quality of the item. In a computing context, data provenance represents, in a formal manner, different relationships between entities (data items), activities (data transitions) and agents (which cause the transition). In other words, it can be understood as a formal set of documents which help in understanding the data existence and it's flow to trace its integrity (quality). [1]

Information flow tracking is a security mechanism designed to monitor how sensitive information spreads in a system. In data provenance context, information flow tracking is used to track the data and is be further used to put limits on the dissemination of a piece of sensitive data once it's out of it's container of information. This allows high level policies such as " my banking information will not be sent outside my system " or " my banking information will not be mixed with my wife's banking information" to be enforced easily. [2]. An explicit information flow is defined as the copy, usually partial, of the content of one container of information to another. [3]

Data provenance with a completeness property ensures that all the information flows of the data within the system are being recorded. Pasquier et. al [6] in his work on Run-time Analysis of Whole-system provenance ensures the completeness and accuracy of the provenance capture mechanism. They further use Georget et al. [3] formalism to show that all the information flows between the kernel objects are properly recorded. However, this does not prove that all the security related flows are also being captured by the system.

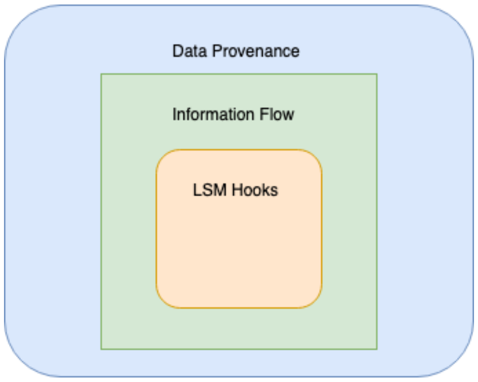

Fig. 1. High-level Architectural view. A structured way to formalize whole-system data provenance is to formalize if all the information flows are being captured. In order to make sure that all information flows are being captured we need to make use of LSM to insert hooks at every point in the kernel where a user-level system call is about to result in access to an important internal kernel object.

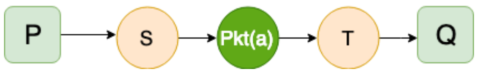

Fig. 2. A simple provenance DAG: a process P sends packet Pkt(a) to process Q using the sockets S and T.

Hence, as a part of this work we try to answer the question about complete and security related flows of the system using a whole-system provenance capture called Camflow. [7]

The first part of the challenge includes answering the question if the completeness formalism is enough to detect all security related flows. Xueyuan et al. uses whole-system provenance capture tool for fault-detection [8]. Pasquier et. al. [6] shows that it's possible to detect intrusions during run-time of the system. However, is it enough to detect all sorts of data leaks and/or intrusions is one question we try to answer as a part of this work. This analysis has been performed for Linux Kernel 4.3. We try to see if the formalism still holds for the updated version.

The second contribution of our work is, if indeed Camflow [7] is able to capture all security-related flows, do they all show up as an anomaly in the provenance-graph. The provenance-graph is constructed from the data captured from all the security-related flows in the system.

II. Background

We conduct an analysis to see if the current version of the information flow patch applied to CamFlow is capable enought to catch all security violations. In order to do so, we make use of three open-source tools/libraries/patches CamFlow, GraphChi and complete flow capture formalism proposed by inria.

A. CamFlow

As we have discussed before data provenance records the chronology of ownership, change and movement of an object or a resource. There are many provenance capture systems available including PASS [10], Hi-Fi [11], Linux Provenance Module [12] and CamFlow [9]. The capture mechanism is built on the Linux Security Modules (LSM). LSM is a framework that allows the kernel to support variety of computer security models while avoiding favoritism towards any of the security implementations.

We chose CamFlow for testing our hypothesis because it adopts the LSM architecture and support for NetFilters which makes it a maintainable practical whole-system provenance implementation .CamFlow maintains a patch based on Georget et al's [3]formalism which provides completeness guarantees. The fact that Cam-flow provides completeness guarantees, its a strong choice for researchers to use it as a fault-detection tool [8]. Pasquier et al. uses CamFlow to perform a run-time analysis of whole-system provenance to find intrusions [6]. However, since the completeness guarantees are specific to kernel version 4.3, we want to provide a formalism which shows that it holds for the updated kernel versions.

B. GraphChi

The representation of the provenance data is in form of a directed acyclic graph (DAG). Every node in the DAG represents an entity, an activity or an agent. Each directed edge represents interaction between each node. Fig.2 represents a simple example. In our context, entities are kernel objects, activities are tasks; and agents are users and groups. Fig 2 represents packet being sent from process P to Q from socket S to T.

We use GraphChi, a vertex-centric graph processing model to generate program models and to generate anomalies. The purpose of utilizing GraphChi is to answer that in case of security violations in the system, are the anomalies reflected in the graphs.

C. Previous formalism

Georget et al. [3] proposed a formalism to verify the property for complete capture of the information flows called *Complete Mediation.*

III. Methodology

IV. Evaluation

V. Conclusion

Discussion Topics

The first interesting discussion point is understanding the completeness-security gap if any. Currently the DAGs are generated with respect to the provenance data. However, this is not taking the timing aspect into consideration. Provenance capture mechanisms currently are data related. The way we can capture anomalies is by learning the data pattern from previous immutable data. However, if the attacker brings in the timing aspect, keeping the data intact can that cause damage? It depends on the application if data delays can cause harm.

Another interesting point to discuss would be the completeness of information flows means that it guarantees that all flows are being captured. So you need to place hooks at the relevant places which ensures that all information flow paths are being recorded. However, again this does not cover the event aspect of the system. What if instead of modifying the data with alone or modifying the data with respect to time, it modifies the event in a smart way such that the correlations created by the DAG cannot spot it.

Acknowledgment

This work was done as a part of our term project for CPSC 508. CPSC 508 is a graduate level operating systems course taken by Dr. Margo Seltzer.

References

[1] Lucian Carata, Sherif Akoush, Nikilesh Balakrishnan, Thomas Bytheway, Ripduman Sohan, Margo Seltzer, Andy Hopper, an "A Primer on Provenance," acmqueue, 2014.

[2] Daniel Crawl and Ilkay Altintas , A Provenance-Based Fault Tolerance Mechanism for Scientific Workflows.

[3] Laurent Georget, Mathieu Jaume, Guillaume Piolle, "Verifying the reliability of operating system-level information flow control systems in linux," Proceedings of the 5th International FME Workshop on Formal Methods in Software Engineering, FormaliSE '17, pages 10–16, Piscataway, NJ, USA, 2017. IEEE Press.

[4] GERWIN KLEIN, "Operating system verification—An overview,"Sadhan ¯ a¯ Vol. 34, Part 1, February 2009, pp. 27–69. © Printed in India

[5] Jonathan Pincus and Brandon Baker , "Mitigations for Low-Level Coding Vulnerabilities: Incomparability and Limitations ," 2004.

[6] Thomas Pasquier, Xueyuan Han, Thomas Moyer, Adam Bates, Olivier Hermant, David Eyers, Jean Bacon, Margo Seltzer, "Runtime Analysis of Whole-System Provenance ,"16 pages, 12 figures, 25th ACM Conference on Computer and Communications Security 2018.

[7] Thomas Pasquier, Xueyuan Han, Mark Goldstein, Thomas Moyer, David Eyers, Margo Seltzer, Jean Bacon, Practical Whole-System Provenance Capture , SoCC '17 Proceedings of the 2017 Symposium on Cloud Computing.

[8] Xueyuan Han, Thomas Pasquier, Tanvi Ranjan, Mark Goldstein, and Margo Seltzer, Harvard University
, "FRAPpuccino: Fault-detection through Runtime Analysis of Provenance ," HotCloud'17.

[9] PASQUIER, T. F.-M., SINGH, J., BACON, J., AND EYERS, D. , "Information flow audit for paas clouds. Cloud Engineering (IC2E), 2016 IEEE International Conference on (2016), IEEE, pp. 42–51.

[10] MUNISWAMY-REDDY, K.-K., HOLLAND, D. A., BRAUN, U., AND SELTZER, M. I. , " Provenance-aware storage systems. ," In USENIX Annual Technical Conference, General Track (2006), pp. 43–56..

[11] POHLY, D. J., MCLAUGHLIN, S., MCDANIEL, P., AND BUTLER, K , "Hi-fi: collecting high-fidelity whole-system provenance ," In Proceedings of the 28th Annual Computer Security Applications Conference (2012), ACM, pp. 259–268.

[12] Adam Bates, Dave (Jing) Tian, and Kevin R.B. Butler , "Trustworthy Whole-System Provenance for the Linux Kernel. 24th USENIX Security Symposium, 2015.

[13] PASQUIER, T. F.-M., SINGH, J., BACON, J., AND EYERS, D. , "Information flow audit for paas clouds. Cloud Engineering (IC2E), 2016 IEEE International Conference on (2016), IEEE, pp. 42–51.

[14] PASQUIER, T. F.-M., SINGH, J., BACON, J., AND EYERS, D. , "Information flow audit for paas clouds. Cloud Engineering (IC2E), 2016 IEEE International Conference on (2016), IEEE, pp. 42–51.

A. Equations

Number equations consecutively. To make your equations more compact, you may use the solidus (/), the exp function, or appropriate exponents. Italicize Roman symbols for quantities and variables, but not Greek symbols. Use a long dash rather than a hyphen for a minus sign. Punctuate equations with commas or periods when they are part of a sentence, as in:

$$a + b = \gamma \qquad (1)$$

Be sure that the symbols in your equation have been defined before or immediately following the equation. Use "(1)", not "Eq. (1)" or "equation (1)", except at the beginning of a sentence: "Equation (1) is . . ."

B. Figures and Tables

a) Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation "Fig. **??**", even at the beginning of a sentence.

TABLE I
TABLE TYPE STYLES

Table Head	**Table Column Head**		
	Table column subhead	***Subhead***	***Subhead***
copy	More table copy[a]		

[a]Sample of a Table footnote.

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity "Magnetization", or "Magnetization, M", not just "M". If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write "Magnetization (A/m)" or "Magnetization {A[m(1)]}", not just "A/m". Do not label axes with a ratio of quantities and units. For example, write "Temperature (K)", not "Temperature/K".