Linux security modules and whole-system provenance capture

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

ABSTRACT

Data provenance describes how data came to be in its present form. It includes data sources and the transformations that have been applied to them. There have been several different OS provenance capture tools in the past. However, only CamFlow and its predecessor Linux Provenance Modules use Linux Security Modules which is a framework that allows the Linux kernel to support a variety of computer security models while avoiding favouritism toward any single security implementation. This paper examines the relationship between LSM and CamFlow. the two key research questions that are addressed: 1) Does the latest version of Linux LSM capture all security related information flows? 2) Given the results of (1) and the data captured by CamFlow in its LSM hooks, can we prove that an intrusion will be reflected as a different in the provenance graphs. This is an important question that either validates or refutes the use of kernel provenance for intrusion detection.

I. INTRODUCTION

Provenance is the chronology of ownership, custody or location of a historical object. These documents are used to guide the authenticity and quality of an item. The term is used in a wide range of fields including science and computing. In computing...

Data provenance has wide range of applications ranging from dependibility (reliability and security) of the system to reproducibility of computational experiments.

Security is a major concern since there is no permanent fix to detect intrusions. Its a race between attackers and defenders. Data provenance can help detect such intrusions in the kernel. However, in order to make sure that all the intrusions are getting detected, we need a way to capture the entire data flowing in the system which is why we use CamFlow.

Camflow is a practical implementation of whole-system provenance capture that can be easily maintained and deployed. They use Linux security modules as the underlying framework to capture the data flows..

In this paper, we examine if the whole system provenance capture mechanism developed by Camflow can be utilized for intrusion detection.

We first focus on a methodology proposed by Georget et al. to verify if indeed every security related flow goes through an LSM hook. The methodology proposed by Georget et al. had been designed for linux kernel v4.3 to ensure that all security flows are passing through the hooks. We gaurantee the same for v4.20 which is the last formal release.

A second challenge in determining if the use of kernel provenance can help in intrusion detection, we prove that a security breach is reflected in the provenance graph it produces.

Key contributions
The key contributions of our work include: 1) analysing the callgraphs obtained from static analysis of the linux kernel to see if all paths are being tracked 2) a formalism to automate the process of analysing the callgraphs for the future versions 3)ensuring that

II. BACKGROUND

There have been multiple provenance capture systems proposed before this PASS, HiFi. However, they had problems 1) struggled to keep abreast with current OS releases 2) Did not have whole system provenance capture guarantees. Camflow provides easy maintainability because of its adoption of LSMs and Net-filters.

LSMs are security frameworks which were initially built for accomodationg differnt access policies such as MAC and DAC. It was however not built for information flow policies. This is why ensuring if it captures all the information flow paths is important. Georget et al. does so...

¡¡¡¡¡¡¡ HEAD **Georget et al methodology** Georget performs static analysis using a four step approach.

1) The model designed by Georget to represent system calls and their execution paths does not describe the C source code. They instead use an internal representation called GIMPLE[].
2) Each system call is represented by a control flow graph (CFG).

3) The paths in these graphs model the execution paths in the program as defined by the classical graph theory[].
4) The system calls are analysed one at a time.
5) Each system call contains multiple functions. These functions are inlined into the system calls to reduce the analysis to

======= The purpose of information flow control is to monitor the way in which information is disseminated in the system once it is out of its original container. This is unlike access control which can only enforce rules on how whose containers are accessed. Several scientific and technical challenges exist in ensuring complete information flow. One of them being the large Linux kernel code base. Georget tackles this issue in his work.

In order to improve the state of art of the information flow systems, Georget et. al developed a plugin for the GCC compiler to easily extract and visualize control flow graphs of kernel functions...

Camflow which utilizes LSM for the whole-system provenance capture. It collects the provenance data and constructs provenance graphs from the collected data. Now that we are aware that Georget's methodology ensures the placement of hooks such that complete information flow is possible, we prove/disprove that the violations are reflected in the provenance graphs. ¿¿¿¿¿¿¿ becc8df1a0d27778850c5b0b894284758ab5ad42

III. Methodology

IV. Results

V. Limitations

VI. Related Work

VII. Conclusion

VIII. Discussion

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 1016, Piscataway, NJ, USA, 2017. IEEE Press.

[4] GERWIN KLEIN, "Operating system verificationAn overview,"Sadhan a Vol. 34, Part 1, February 2009, pp. 2769. 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. 4251.

[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. 4356..

[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. 259268.

[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. , "Camflow information flow patch. In https://github.com/CamFlow/information- flow- patch. .

[14] Chris Wright, Crispin Cowan, Stephen Smalley, James Morris, Greg Kroah-Hartman. , "Linux Security Modules: General Security Support for the Linux Kernel. Proceeding Proceedings of the 11th USENIX Security Symposium Pages 17-31
August 05 - 09, 2002 .

[15] PASQUIER, T. , "CamFlow development. In https://github.com/CamFlow/camflow-dev.

[16] INRIA , "The Kayrebt Toolset In http://kayrebt.gforge.inria.fr/