Integrity checking and abnormality detection of provenance records

Sheung Chi Chan
University of Edinburgh

Ashish Gehani
SRI International

Hassaan Irshad
SRI International

James Cheney
University of Edinburgh
The Alan Turing Institute

Abstract

Data provenance is a kind of meta-data recording inputs, entities and processes. It provides historical records and origin information of the data. Because of the rich information provided, provenance is increasingly being used as a foundation for security analysis and forensic auditing. For example, system-level provenance can help us trace activities at the level of libraries or system calls, which offers great potential for detecting subtle malicious activities that can otherwise go undetected. However, most of these security related applications of provenance data require completeness and correctness of the provenance collection process. This cannot be guaranteed in some cases because some provenance recording modules collect information from some unreliable sources. We present work in progress on provenance graph integrity checking and abnormal component detection using ProvMark, the provenance expressiveness benchmarking tool. We also discuss possible applications of the ProvMark tool in aid of the quality checking of provenance data.

1 Introduction

There exist several different provenance systems in the literature, PASS [9], Hi-Fi [11], SPADE [6], OPUS [1], LPM [2], Inspector [12], and CamFlow [10] covering a variety of operating systems from Linux and BSD to Android and Windows. These tools aim to collect provenance information from different sources in the operating system and generate high-level provenance graphs after post-processing. The post-processing aims to make the resulting provenance graph records more suitable for different applications, including forensic audit, security and online analysis. Provenance systems typically also provide configuration settings allowing customization of the processing actions resulting in provenance suitable for particular applications.

Although provenance is increasingly being used as a basis for security analysis or forensic auditing, security analysts still face a daunting task. To make use of provenance in these settings, one must make sure the provenance collected and processed by different provenance systems does not leak key information about execution or introduce impurities into the resulting provenance records. In other words, security analysis requires high quality provenance data in terms of completeness and correctness. Chan et al. [5] proposed an expressiveness benchmarking approach to elucidate the relationship between an operating system activities and the resulting provenance graphs. Their approach aims to generate provenance benchmarks to describe system call behaviour and provide objective criteria for analysing the expressiveness of resulting provenance data by comparing the quality of the provenance data in terms of correctness and completeness. Chan et al. further propose a tool called ProvMark [4] to automate the expressiveness benchmarking approach. ProvMark is helpful for understanding the relationship between operating system kernel activities and the high level provenance graph.

This paper proposes an application and extension of ProvMark for integrity checking and abnormality detection of provenance records. This paper aims to define the problem and present results of initial manual experiments with a case study to provide an introductory demonstration of how ProvMark's approach can help to discover integrity problem and abnormal behaviour in provenance generation because of unreliable source and communication channels. The integrity checking aims to answer the following questions.

- Are there any missing records in the provenance graphs?
- Are there any irregular records in the provenance graphs?
- What records are missing in the provenance graphs?
- What activity is different from a normal execution?

The structure of the rest of this paper is as follows. Section 2 provides some background of the ProvMark tool and discusses how unreliable sources and erroneous post-processing can lead to incomplete or incorrect provenance records. Section 3 discusses possible new automated modules of ProvMark for integrity checking and abnormal detection. Section 4 describes a case study of source unreliability (Linux Audit System), in addition to manual experiments to show how ProvMark can help to discover integrity problems and abnormal behaviour in the resulting provenance records. Section 5 discusses the usefulness of the presented application of ProvMark and possible extension of ProvMark in terms of a new automated comparison module. Finally, Section 6 concludes and discusses ongoing and future steps.

2 Background

The main purpose of this paper is to investigate ways to check the quality of provenance data by adopting the provenance expressiveness benchmark approach used by the automated tool ProvMark. We use certain manual experiments to demonstrate the feasibility of this approach.

Chan et el. [4, 5] proposed *provenance expressiveness benchmarking* and then developed an automated system ProvMark to handle the benchmarking process. Their major contributions include the identification of how different system call behaviours contribute to the final provenance graph results and generate provenance benchmarks for each of the system calls automatically. These provenance benchmarks show how three different provenance systems each handle the same set of operations in a different manner. They also act as a basis to identify what information is processed by each provenance system in the final provenance graph if certain activities happened in the execution period. The motivation for this approach is the lack of standardization of provenance collection and processing across different provenance systems. Developers of those provenance systems can then use these provenance benchmarks to compare provenance data qualitatively for different applications.

ProvMark works by comparing the provenance resulting from monitoring a *background program* that performs background activity such as process initialization and termination with a *foreground program* which also performs the additional target activities. Another key contribution of the original ProvMark tool is the adoption of answer set programming to help solve the hard graph/sub-graph isomorphism problems of property graphs [3]. This allows a more flexible way to compare provenance graphs with large numbers of elements including property labels. In this paper, we discuss possible adoptions and extensions of such approach in the integrity checking and abnormality detection approach.

Provenance systems tend to source information from different system components. These components include the Linux Security Module (LSM), Linux Audit Framework, Windows Process Monitors or even the LLVM compiler framework or network socket from some distributed hosts. Most of these components are located in the kernel where the provenance recording system has no direct access. In this situation, those components rely on relaying services to pass the information back to some reachable location for processing. This setting may also pose certain problems. The reliability of the intermediate relaying components, like the audit dispatcher, may also affect the integrity of the information. In addition, these information passing channels may be polluted by adversaries which could also affect the final provenance record. As a whole, unreliable sources of information, data relaying channels and man-in-the-middle attacks affect the source information passed to a provenance system. Last but not least, it is also possible that certain faulty updates or alterations of the provenance systems themselves can lead to integrity and quality problems of the resulting provenance record. These possible pollutants of the provenance record affect the provenance quality and act as possible unreliable factors towards the application of provenance. For example, an adversary can purposely alter the information from the relaying channels to hide traces of its actions. Security analysis based on those affected relaying channels would fail to detect this malicious behaviour. In addition, if an audit log record for the opening of certain sensitive files has been dropped because the Audit Log buffer is full, then the system will fail in tracing further actions on that file because the reference has been lost. These settings open a vulnerability for violating certain security constraints and should be detected.

In this paper, we introduce certain small manual experiments to simulate unreliable source of provenance collection to demonstrate the feasibility of the integrity checking and abnormality detection by ProvMark. Evaluation and analysis of ProvMark tool are not covered in this paper.

3 Automatic provenance quality checking

As mentioned, end-users and the developers of provenance systems can use ProvMark generated provenance benchmarks for qualitative comparison of suitability for different applications. Unfortunately, ProvMark does not fully automate the further application of the generated benchmark. The users of the ProvMark tools need to manually compare the generated provenance benchmark for the data qualitative analysis and other applications. For this reason, we aim to extend the ProvMark tool by studying possible extensions and automation for the application of those provenance benchmark, especially for the integrity checking and abnormality detection process. As mentioned above, we design small manual experiments to demonstrate the feasibility of such automation and show that further work could be done to extend ProvMark in solving more realistic applications. In addition, we also take advantage of ProvMark's provenance graph comparison approach for the identification of the differences between the two graphs in order to understand what information is missing in a provenance collection process. This requires the graph edit distance and sub-graph isomorphism matching approach automated by the ProvMark tool. We assume that the incompleteness and incorrectness factors in this situation come from unreliable sources of provenance capture or purposeful alteration by an adversary during the provenance collection period. We also assume that the ProvMark tool does truly reflect the behaviour of the provenance generation processes.

In this paper, we aim to demonstrate a possible extension of the ProvMark tool. We make use of the provenance benchmark generation and the comparison approach for integrity checking and abnormality detection. The major way to do so is to create a complete graph as a control graph model. The control model is referring to a graph which has been verified manually by the provenance system developer as being the correct and complete graph generated by a tool for describing a known set of activity sequences. Thus the control model is assumed to be correct and act as the model answer on how this specific provenance tool describes certain activity sequences. Currently, those valid graphs still require experts' verification but this may also be automated in a later stage. In this situation, we are assuming that the input is deterministic and does not contain other variations which affect the determination process. Whenever integrity checking is needed, we compare the newly generated provenance graph with the control graph and identify the difference sub-graph. The resulting provenance sub-graph identifies the missing (or additional) elements from the newly generated provenance graph that represent either loss of records or existence of abnormal records. ProvMark can generate such a correct control graph for certain activity sequences. The above mentioned approach is similar to the regression testing application suggested by the ProvMark developers, but it works in a slightly different manner. Regression testing applications are more concentrated on checking for differences in behavior resulting from code changes, rather than integrity violations or abnormal behavior resulting from missing or altered audit records.

As suggested by the ProvMark developers, ProvMark aims to provide a benchmark to assist the solving of the problem. Although the applications can in theory be extended to handle abnormal behaviour by adversaries or faulty updates, we concentrate the discussion on the unreliable sources of information and potentially lossy relaying channels in this paper. We make use of one of the provenance systems, SPADE with its Linux Audit reporter, as an example to demonstrate the feasibility of such approach and act as the base for the development of the automated module. The reason for choosing this combination for our case study is because of the unreliability of the Linux Audit System providing the source of information for this combination. To decrease the overhead of the logging process, the Linux Audit System allows configuration using *audit rules*. From the original Linux Audit Daemon configuration manual [7], it has mentioned a *max_log_file* parameter which allows the system administrator to limit the size of the log file and also use the *max_log_file_action* to configure how the Linux Audit System handles the additional log when the log file is full. The key problem here is how the administrator handles situations in which storage space or communication buffer space is exhausted. Although the default action of *max_log_file_action* is to keep the additional log in buffer, it is possible to configure the system to drop the oldest log record entries or simply ignore new audit log records. Also, the buffer may be full and fail to store additional log records. These possible settings may result in loss of log records and make the Linux Audit System an unreliable (and incomplete) source for system and kernel activities.

Morrison [8] provides an analysis of the whole Linux Audit System and summarizes some standards and mechanisms for the prevention of audit data loss and the protection of the integrity of those data. Although these methods do help to reduce the loss rate by different configuration enforcement including the maximum buffer size settings, some of them still require certain tradeoffs. These tradeoffs include immediate halting of system applications or denial of later activities. Thus a certain level of integrity detection is still required to detect random loss or altered audit log records. In addition, SPADE settings allow us to provide static audit records as source of information. This allows us to set some manual experiments and test cases for demonstrating the ability of ProvMark in this application and the feasibility of the automated module. It is worth mentioning that SPADE can be configured to obtain information from different sources, but we are just concentrating on the single source of information from the Linux Audit System as it is one of the obvious unreliable sources of information which allow us to demonstrate the feasibility of the approach before considering more complicated cases.

4 Case Study on Linux Audit System

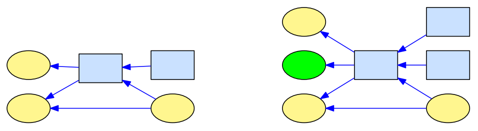
Linux Audit is one of the major sources of information for understanding and tracking different kind of activities and data exchange in the kernel. This makes it a perfect target for both system analysts and adversaries to gather information. By default, the Linux Audit System monitors over 300 system calls and generates log records for every execution and data exchange. Some of the system activities may span multiple log records. For example, a simple file open activity will span two or three log records. One of them records the file path and certain permission and another one of them assigns a new file descriptor to the file itself. It may also have an additional record for storing the actor's permission of this file open activity. If the mapping log record of the file descriptor has been lost, then later read / write activity referring to this file descriptor will not be traced back to the target file. This may result in incomplete records related to further action to this file. Zeng et al. [13] have done some experiments on the demonstration of Audit overhead on both time and memory. They show that overhead for both execution power and storage are directly proportional to the number of activities required to be audited. They also suggested, in order to limit the overhead of the audit service, to customize the audit rules to either limit the number of target activities or the maximum amount of log record storage. Otherwise, a full system audit will fill up the storage memory with a huge set of log records quickly.

In this subsection, we create test cases to demonstrate the ability of ProvMark to discover of these missing records and abnormal records. The SPADE tool allows us to configure its Audit reporter by sourcing information directly from the live audit log records returned by the Linux Audit System. It also allows us to run the Audit reporter with a static log record input for testing purposes. We make use of this feature and create certain static log records manually for the case study. We first create a static log record with around 20 different system call activities as control record. Then we remove different numbers of system call activities from the control log record to create multiple test cases. We also altered system calls in the control log to demonstrate the abnormal record discovery. The different test cases are shown in Table 1.

Test Cases	# of syscall removed	# of syscall altered
TC#1	1	0
TC#2	2	0
TC#3	3	0
TC#4	0	1
TC#5	2	1

Table 1: Modification of log records for test cases

Making use of the benchmark generation of ProvMark, we treat the audit log as the source of information for the SPADE tool to collect provenance. We configure SPADE to read the static log records we provided instead of the live audit log record from the Linux Audit System. The control log record acts as the source for the execution of the foreground program, while the modified log record for each test case acts as the source for the execution of the background program. In our approach, we are assuming the control source model always contains the complete and correct provenance graph for the execution, thus when we are using it as the model answer for integrity and abnormal behaviour, it always contains more or equal number of graph elements. As a result, it fits the ProvMark terminology of the foreground program sources. These static log records are processed by SPADE and then post processed by ProvMark to generate provenance benchmarks. As the source of provenance collection is static, the result should be accurate across different trials thus we eliminate the side factors affecting our test cases. Literally, the generated provenance benchmark in each case should represent the difference between the foreground graph and the background graph which should describe the missing record. It is because the only difference between the control log and the test cases' log is the record we purposely removed. With the present of an altered activity, the approach will still work but the results are presented in a slightly different ways. As ProvMark is using the edit distance approach to identify the closest match, it is possible to locate the similar part of the two graphs and identify the different portion of the graphs while maintaining the least edit distance. The result will then show the graph


(a) TC#2 (b) TC#5

Oval:Artefacts *Rectangle:Processes*

Figure 1: Sample Provenance Graph Structure

structure in the control graph which is either missing or behaves differently in the target graph of the new execution. This application of the edit distance approach in ProvMark in aid of common structure discovery is described and proposed by Chan and Cheney [3] which also contains proof of the correctness of the edit distance approach automation.

In Table 1, we describe the five test cases for our experiment. We are using the same control log record with 20 system calls for all five test cases. The control log record are a subset of a audit log record from real execution. Test cases TC#1 to TC#3 are used to demonstrate the integrity checking application of ProvMark. We remove different numbers of system calls from the control log record to form the static log record for each test case. In all three cases, ProvMark has successfully identified the missing patterns by showing the provenance patterns of the missing system calls as the resulting provenance benchmark. This is supported by the sub-graph isomorphism approach. Test case TC#4 is used to demonstrate the abnormality checking application of ProvMark. We purposely alter one of the system call activities and ProvMark does successfully identify the abnormal graph structure in the control audit log record. Test case TC#5 is a combined test case with both removal and replacement of system calls. We remove one system call record and replace another system call from the control log record. At last, ProvMark is still able to discover these changes by showing the graph structure of missing log records and the original audit log record that are being replaced. Both test cases TC#4 and TC#5 require the discovery or abnormal structure between the two graphs, which require the support by the edit distance approach to determine the matching components before filtering out the abnormal structure.

In the above experiment, we created different test cases that simulate possible loss of log records or both purposeful or non-purposeful alteration to the live log record from the Linux Audit System. For the integrity checking cases, the provenance result generated by ProvMark identifies which of the provenance patterns exist in the control log and are missing from the checking target. An empty provenance benchmark should be returned if there are no integrity (completeness) problems in the new provenance collection section. A similar approach can also be used for abnormality detection by creating a verified control audit log record.

Figure 1a shows the provenance graph structure describing the two missing system calls (*rename* and *open*) in test case TC#2 and Figure 1b shows the provenance graph structure describing the two missing system calls (*rename* and *open*) in test case TC#5. In addition, Figure 1b also shows the altered system call *clone* which has been altered to the *fork* system call in TC#5. By studying the resulting provenance graph structure, the users can understand what is abnormal in the target execution and provenance collecting session. In either of the shown test cases, the resulting provenance graph contains different provenance structure or misses some of the provenance structure. ProvMark compares the provenance graphs and identifies which parts are missing or different compared to the control graph. As we use the control graph as the foreground graph for the ProvMark process, the resulting provenance structures are sub-graphs of the control graph, which describe the expected provenance structure of the system calls that are either missed or modified in the provenance graph of the target session. For simplicity, only the structure of those provenance sub-graphs is shown in the figure.

5 Discussion

In this paper, we use static audit log files as a case study to demonstrate possible applications of ProvMark in the detection of provenance graph differences because of altered sources of provenance collection. This acts as the preliminary step to simulate unreliable sources of provenance collection which could be extended to real life integrity checking and abnormal activities detection. We use some of the small scale static examples as the source of information for our test cases. Although it shows good results and demonstrates the usefulness of ProvMark in aid of discovering integrity problems and abnormal activities, this approach is still immature and certain limitations exist which require future effort to make the application more mature for real life applications.

One of the limitation is the scalability problem. We are only using static audit log with around 20 system call activities for our case study. It shows an acceptable result and overhead in terms of storage and processing time. Similar to the limitations of the original ProvMark tool, one of the big problems to handle in this application is the comparison of graphs using the isomorphic graph/sub-graph comparison and matching problem mentioned in the literature of ProvMark. This is a hard question and remains as one of the biggest limitations of ProvMark to realistic applications. As we are demonstrating a possible application of ProvMark, this is an unavoidable limitation extending from the ProvMark tool. There are temporary remedies by limiting the size of the test targets and dividing the problem into smaller test cases, but the problem still remains for larger set of data or real time analysis.

Another possible limitation is on the settings of our simulation case study. We currently only consider static log records which are generated by removing and changing different number of log records in a single control log record. All the data we are using are unchanged throughout the experiments. Possible further work is to make the testing log record more dynamic by introducing certain random removals and alterations of log records to simulate random loss in Linux Audit System. Simulation of more sophisticated behavior would improve our understanding of the feasibility of this approach before moving to larger scale provenance graph analysis.

6 Conclusion

This paper demonstrates a possible extension of ProvMark in aid of integrity checking and abnormality detection of provenance graphs resulting from unreliable sources. With the provenance benchmarking approach, ProvMark is able to provide sources of information for qualitative validation of provenance. With the understanding of how each provenance system describes each of the system call activities in the operating system, a user is able to identify the behavioural differences between two executions. Users not only can get provenance benchmarks describing the difference between two executions. They can also make use of ProvMark to understand in detail what system call is missing or behaving abnormally by referencing the system call benchmarks that are generated by ProvMark for each of the supporting provenance recording systems.

This paper provides some basic tests on the way to demonstrate the usefulness of ProvMark in this direction. We also demonstrate how ProvMark can be extended to detect synthetically-constructed integrity or abnormality problems. This understanding provides a basis for developing a fully automated approach for integrity checking and abnormality detection in real time. We have provided some proof-of-concepts based on small manual experimental test cases to investigate the feasibility of such an approach. The experiment shows that ProvMark is able to aid the discovery of incompleteness and incorrectness in the provenance graph when the unreliable provenance sources lose or alter some of the information. Future work aims to extend the manual experiment to a new automated module for ProvMark to allow broader application to security and forensic applications on top of the formal mapping of activity behaviour and provenance graph structure.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant ACI-1547467. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. Chan and Cheney were also supported by ERC Consolidator Grant Skye (grant number 682315).

References

[1] Nikilesh Balakrishnan, Thomas Bytheway, Ripduman Sohan, and Andy Hopper. OPUS: A lightweight system for observational provenance in user space. In *Proceedings of the 5th USENIX Workshop on Theory and Practice of Provenance (TaPP 2013)*. USENIX Association, 2013.

[2] Adam M. Bates, Dave Tian, Kevin R. B. Butler, and Thomas Moyer. Trustworthy whole-system provenance for the Linux kernel. In *Proceedings of the 24th USENIX Security Symposium (USENIX Security 2015)*, pages 319–334, 2015.

[3] Sheung Chi Chan and James Cheney. Flexible graph matching and graph edit distance using answer set programming. In Ekaterina Komendantskaya and Yanhong Annie Liu, editors, *Practical Aspects of Declarative Languages*, pages 20–36, Cham, 2020. Springer International Publishing.

[4] Sheung Chi Chan, James Cheney, Pramod Bhatotia, Thomas Pasquier, Ashish Gehani, Hassaan Irshad, Lucian Carata, and Margo Seltzer. ProvMark: A provenance expressiveness benchmarking system. In *Proceedings of the 20th International Middleware Conference*, Middleware '19, page 268–279, New York, NY, USA, 2019. Association for Computing Machinery.

[5] Sheung Chi Chan, Ashish Gehani, James Cheney, Ripduman Sohan, and Hassaan Irshad. Expressiveness benchmarking for system-level provenance. In *9th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2017)*, Seattle, WA, June 2017. USENIX Association.

[6] Ashish Gehani and Dawood Tariq. SPADE: support for provenance auditing in distributed environments. In *Proceedings of the 13th International ACM/IFIP/USENIX Middleware Conference (Middleware 2012)*, pages 101–120, 2012.

[7] Steve Grubb. *auditd.conf(5) Linux User's Manual*. Red Hat, April 2016.

[8] Bruno Morisson. Analysis of the Linux audit system. Master's thesis, Information Security Group, Royal Holloway, University of London, 2014.

[9] Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, and Margo Seltzer. Provenance-aware storage systems. In *Proceedings of the 2006 USENIX Annual Technical Conference*, pages 43–56, 2006.

[10] Thomas Pasquier, Xueyuan Han, Mark Goldstein, Thomas Moyer, David M. Eyers, Margo Seltzer, and Jean Bacon. Practical whole-system provenance capture. In *Proceedings of the 2017 Symposium on Cloud Computing (SoCC 2017)*, pages 405–418, 2017.

[11] Devin J. Pohly, Stephen E. McLaughlin, Patrick D. McDaniel, and Kevin R. B. Butler. Hi-Fi: collecting high-fidelity whole-system provenance. In *Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC 2012)*, pages 259–268, 2012.

[12] Joerg Thalheim, Pramod Bhatotia, and Christof Fetzer. Inspector: Data Provenance using Intel Processor Trace (PT). In *Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 25–34. IEEE, 2016.

[13] Lei Zeng, Yang Xiao, and Hui Chen. Auditing overhead, auditing adaptation, and benchmark evaluation in Linux. *Sec. and Commun. Netw.*, 8(18):3523–3534, December 2015.