



THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

Research Commons

<http://researchcommons.waikato.ac.nz/>

Research Commons at the University of Waikato

Copyright Statement:

The digital copy of this thesis is protected by the Copyright Act 1994 (New Zealand).

The thesis may be consulted by you, provided you comply with the provisions of the Act and the following conditions of use:

- Any use you make of these documents or images must be for research or private study purposes only, and you may not make them available to any other person.
- Authors control the copyright of their thesis. You will recognise the author's right to be identified as the author of the thesis, and due acknowledgement will be made to the author where appropriate.
- You will obtain the author's permission before publishing any material from the thesis.

Reconstructing Data Provenance from Log Files

A thesis
submitted in fulfilment
of the requirements for the degree
of
Doctor of Philosophy
at the
University of Waikato
by
Yu Shyang Tan



THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

Department of Computer Science
Hamilton, New Zealand
2017

Abstract

Data provenance describes the derivation history of data, capturing details such as the entities involved and the relationships between entities. Knowledge of data provenance can be used to address issues, such as data quality assurance, data audit and system security. However, current computer systems are usually not equipped with means to acquire data provenance. Modifying underlying systems or introducing new monitoring software for provenance logging may be too invasive for production systems. As a result, data provenance may not always be available.

This thesis investigates the completeness and correctness of data provenance reconstructed from log files with respect to the actual derivation history. To accomplish this, we designed and tested a solution that first extracts and models information from log files into provenance relations then reconstructs the data provenance from those relations. The reconstructed output is then evaluated against the ground truth provenance. The thesis also details the methodology used for constructing a dataset for provenance reconstruction research.

Experimental results revealed data provenance that completely captures the ground truth can be reconstructed from system-layer log files. However, the outputs are susceptible to errors generated during event logging and errors induced by program dependencies. Results also show that usage of log files of different granularities collected from the system can help resolve logging errors described. Experiments with removing suspected program dependencies using approaches such as blacklisting and clustering have shown that the number of errors can be reduced by a factor of one hundred. Conclusions drawn from this research contribute towards the work on using reconstruction as an alternative approach for acquiring data provenance from computer systems.

Acknowledgements

This study has been, without doubt, one of the biggest challenges I have undertaken in my life thus far. Its completion is by no means a sole effort and would not have been possible without the following people.

First and foremost, I would like to thank my supervisors. Dr. Ryan Ko, for all the guidance, advice and support given throughout the years of my study. Dr. Geoffrey Holmes, for offering your wisdom, both academically and socially. I have grown because of your advice and am confident it will continue to benefit me as I further my career. William Rogers for the technical insights given when it comes to the work I have done. Many times, it is your insight that helped me to overcome the frustrating issues encountered.

Next, I would like to thank CROWs from the Cyber Security Lab, especially, Mark Will, Jeff Garae and Craig Scoon for the times together. My time in New Zealand was alot easier and bearable because of you guys. Thank you Mark and Jeff for being my any-time-idea-bouncing-boards. I hope I have not caused any cracks from all the stress balls thrown at you guys. Keep on TPM-ing! I would also like to thank the postdocs of the lab, Dr. Harris Lin and Dr. Sivadon Chaisiri for all the help, support and advice. Starbucks! Special thanks to Baden Delamore and Dr. Raja Naeem Akram. I am glad to have known you guys. The dinner discussions and FIFA sessions have been interesting and have certainly broadened my horizons.

I would also like to extend my thanks to two groups of people. My family members for your unwavering confidence in me and support. Friends back home, especially Weiqing, Wing Sze and Viling, for the mental support and encouragement. Without the support given from these two groups of people, I would not have been able to overcome the many diffi-

cult times.

Finally, I would like to thank the Department of Computer Science and the Faculty of Computing and Mathematical Sciences for the support given. I would also like to acknowledge the STRATUS project for the financial support given during the final year of my study.

Thank you everyone!

Table of Contents

List of Figures	xi
List of Tables	xvii
Nomenclature	xix
1 Introduction	1
1.1 Background	1
1.2 Motivation	3
1.3 Log files and Provenance	4
1.4 Thesis Question	7
1.5 Scope of Research	8
1.6 Requirements	12
1.7 Thesis Structure	13
2 Literature Review	15
2.1 Extracting Information from Logs	15
2.2 Modelling Provenance Relations	18
2.2.1 Contextual Abstraction	20
2.2.2 Structural Abstraction	23
2.3 Provenance Reconstruction	28
2.3.1 Reconstruction in Provenance	28
2.3.2 Digital Forensics	34
2.3.3 Workflow Planning	39
2.4 Evaluating Provenance Reconstruction	41
2.5 Dataset for Provenance Reconstruction	44
2.6 Summary	45

Table of Contents

3	Constructing a Dataset for Provenance Reconstruction Research	49
3.1	Requirements for Dataset	49
3.2	Evaluating Public Datasets	51
3.2.1	Provenance Datasets	51
3.2.2	Network and System Logs	53
3.3	New Zealand Cyber Security Challenge 2015	56
3.4	Setup for Data Collection	59
3.5	NZCSC'15 Dataset	61
3.6	Use Cases from the NZCSC'15 Dataset	67
3.7	Limitations of the NZCSC'15 Dataset	72
3.8	Summary	74
4	Modelling Log Events as Provenance Relations	77
4.1	Modelling Log Events to Provenance Relations	77
4.2	Assumptions Made	79
4.3	Data Flow Provenance Model	80
4.3.1	Entity	81
4.3.2	Activity	83
4.3.3	Modelling Data Channels	88
4.4	Aggregating Activities	89
4.4.1	Discovering Patterns	89
4.4.2	Parsing	97
4.4.3	Verifying the Patterns	99
4.5	DFPM and OPM	100
4.6	Summary	106
5	Reconstructing Data Provenance	109
5.1	The Data Provenance Reconstruction Problem	110
5.2	Algorithm for Reconstructing Data Provenance	112
5.2.1	Sort Phase	114
5.2.2	Find Instance Phase	116
5.2.3	Reconstruction Phase	117
5.2.4	Agents and Objects	121
5.2.5	Reducing the Number of Iterations on Relation List	123

5.3	Analysis of Algorithm	125
5.4	Context-based Provenance Reconstruction Algorithm . .	128
5.5	Abstracting the Reconstructed Provenance	129
5.6	Summary	131
6	Experiments	133
6.1	Methodology for Evaluating Data Provenance Reconstruc- tion	133
6.2	Experimental Setup	136
6.2.1	Data Pre-processing	136
6.2.2	Description of Experimental Setup	141
6.3	Reconstructing Data Provenance from Different Granu- larities	144
6.3.1	D1 - System Log Files	145
6.3.2	D2 - Application Log Files	150
6.3.3	D3 - System and Application Log Files	151
6.4	Dependency Explosion	154
6.4.1	Understanding the Dependency Explosion Problem .	154
6.4.2	Blacklisting Dependencies	155
6.4.3	Process-access Based	157
6.4.4	Clustering	162
6.5	Summary	167
7	Conclusion and Future Work	169
7.1	Conclusion	169
7.1.1	Summary of Contributions	171
7.2	Future Work	172
7.2.1	Translation between Log and Provenance Domain . .	173
7.2.2	Resolving the Dependency Explosion Problem	173
7.2.3	Extending the Data Flow Provenance Model	174
A	List of Attacks Carried Out in Challenge	175
B	Scenarios for Designing Finite State Automata	177
C	Experimental Results for Process-access based Pruning	183

Table of Contents

D Graph Plots for Determining Number of Clusters **207**

Bibliography **213**

List of Figures

1.1	Two main aspects of practical research in provenance . . .	2
1.2	Examples of provenance adapted from examples discussed by Gil and Miles [2013]	6
1.3	A possible graph visualisation of event logs	7
1.4	Layered view of a computer system derived from the multi-layered architecture view by Silberschatz et al. [2005] . . .	12
1.5	Illustration of the proposed data provenance reconstruction workflow	14
2.1	Comparing log formats of different kernel logging tools . .	17
2.2	Comparing messages between application and kernel logs	19
2.3	Summary of the different types of abstraction in provenance and approaches used to implement them	21
2.4	Difference between granularity layers caused by “contextual abstraction”	21
2.5	Difference between granularity layers caused by structural abstraction	24
2.6	Counter use-cases for structural abstractions without using <i>a priori</i> knowledge	25
2.7	Structural abstraction using <i>a priori</i> knowledge such as the function call tree	26
2.8	Timeline and focus of the series of provenance challenges [Moreau et al. 2010; Groth 2014]	29
2.9	Case where linearity of timeline is not intuitive for deducing relationship between events	36
2.10	Difference between traditional approach to computing true positives and approach proposed by Papineni et al. [2002]	43

List of Figures

2.11 Structure illustrating the categories of work reviewed . . .	46
3.1 Schematics of the setup used for Round 2 of NZCSC'15 . .	58
3.2 Environment separation between data logging and video capture	61
3.3 An example of an entry in the transcript	62
3.4 Flowchart depicting the video transcribing process for system and user generated events	65
3.5 Examples of the video transcript of the ground truth	66
3.6 Converting event in transcript to ground truth provenance	67
3.7 Use-case 1—Backing up /home/tech/PrivateFiles/PrivateFile1 to remote host, Technician VM	68
3.8 Use-case 2—Modification of /root/.ssh/authorized_keys file in order to gain remote shell access through ssh	69
3.9 Use-case 3—Injecting Malicious Payload via Database . . .	70
3.10 Use-case 4—Data provenance showing the derivation of the file /usr/local/apache2/htdocs/red0.php	71
3.11 Mapping between the log files and transcript in NZCSC'15 dataset to granularity level of computer system	75
4.1 Illustration of the adopted adaptor-based architecture for processing log files	78
4.2 Illustration of granularity layers used in the log and provenance domain	79
4.3 Difference between having <i>createfile</i> and using the <i>create</i> activity to represent file creation at the application layer .	87
4.4 Example showing how <i>open</i> and <i>close</i> can be used to group relations into logical groups	88
4.5 Segmentation of system call trace for each scenario	90
4.6 Illustration of the extraction of patterns from modelled provenance relations in the workload phase	91
4.7 FSA for ' <i>createfile</i> ' activity	93
4.8 FSA for ' <i>access</i> ' activity	94
4.9 FSA for ' <i>modify_rw</i> ' activity	94
4.10 FSA for ' <i>modify_w</i> ' activity	94

4.11 FSA for 'datatransfer' activity	95
4.12 FSA for 'copy' activity	95
4.13 FSA for 'merge' activity	96
4.14 Results for applying the FSA on the NZCSC'15 system log files	101
4.15 Events snippet from a LAF log file, simplified to show only the relevant fields	102
4.16 OPM provenance resulting from modelling snippet of events from LAF log file	103
4.17 Provenance of the snippet of events in Figure 4.15 modelled using DFPM	104
4.18 Illustration of the extraction and modelling of log events into provenance relations	106
5.1 Illustration of source and derivative provenance from e	111
5.2 Illustration of the content of data-in and data-out arrays from Agent2's perspective	112
5.3 Illustration of the hypothetical use case that will be used in the explanation of the reconstruction algorithm	113
5.4 From unsorted relations to list of rg , illustrated based on use case shown in Figure 5.3	115
5.5 State of $DP(e)$ at the end of find instance phase (with reference to the example use case)	118
5.6 Finding a match for a data-in rg on Agent. Dotted arrows represent ruled out provenance relations	118
5.7 Illustrating the concept of data-in and data-out rg at agents along a path	119
5.8 Progress of reconstructing the example use case after processing $P1$	120
5.9 Progress of reconstructing the example use case after processing $P3$	120
5.10 Difference in the interpretation of provenance relations in terms of data flow direction between Object and Agent entities	123

List of Figures

5.11	Example scenario for why considering causal order alone cannot accurately determine relevancy of a <i>rg</i> . Letters in round brackets under each edge represents the content of the data communicated at each edge	124
5.12	Overlapping of results (dotted bars) for each iteration in the path reconstruction process	124
5.13	Comparison between the amount of relations and entities for system log files in the CSC'15 dataset	126
5.14	Example of the aggregation process	131
5.15	Illustration of the ground truth and reconstructed provenance graph	131
5.16	From log files to reconstructed provenance	132
6.1	Illustration of a pair-wise provenance relation and the possible outcomes of comparison between the ground truth and reconstructed provenance	135
6.2	Illustration of the procedure for kernel log entry generation	137
6.3	Log sequence error in Linux Audit Framework (LAF) logs .	137
6.4	Impact of system load on logging	139
6.5	Logic sequence error due to system call being interrupted before it can be timestamped correctly	140
6.6	Workflow for reconstructing the data provenance and generating the ground truth from the NZCSC'15 dataset . . .	142
6.7	Sample screenshot showing how erroneous and relevant entities interconnects to form a single connected graph . .	146
6.8	Comparing the data provenance reconstructed from dataset D2 to the ground truth	150
6.9	Duplicated provenance relations in data provenance reconstructed using dataset D3	152
6.10	Dependencies directly connected to relevant process nodes in the reconstructed data provenance graph	155
6.11	Indirect dependencies explosion when reconstructing derivation provenance	156
6.12	Chained dependency explosion through shared system resources	156

6.13	Comparing precision of the reconstructed output between using system log files with no pruning and blacklist pruning (Y-axis is in log scale)	157
6.14	Explosion of dependencies connected to a process—the dependency explosion problem may lead to file dependencies used for the initialisation and execution of a process to be included into the data provenance	158
6.15	Histograms showing the number of dependencies sharing the same number of processes accessing them for the different system log files	160
6.16	Comparing precision of the reconstructed output for process-access based pruning with blacklist and no pruning (Y-axis is in log scale)	162
6.17	Percentage of variance measured for different values of k for system log file used in the reconstruction of <i>use-case 2</i>	164
6.18	Summary of precision for pruning each cluster from system log file, for each use case (Y-axis in log scale)	165
6.19	Comparing precision of reconstructed data provenance for clustering based pruning with other discussed approaches (Y-axis in log scale)	167

List of Tables

3.1	Overview of discussed datasets	52
3.2	Summary of tools used and logs collected on each VM type	60
3.3	Overview of the set of use cases	72
4.1	Composition of concepts for layers in DFPM in BNF form	83
4.2	Abstracting system layer activities to application layer	92
4.3	Abstracting application layer activities to user layer	92
4.4	The 13 relations defined in Allen’s interval algebra	97
4.5	Mapping concepts between OPM and DFPM	105
6.1	Results of reconstruction for each use case using the system only set	145
6.2	Examples of near matches between the ground truth and the reconstructed provenance	147
6.3	Improvement in recall values after adapting the evaluation to the inconsistencies observed in the log files	148
6.4	Results of the reconstruction using only application layer log files	149
6.5	Results of reconstruction different use-cases using both system and application layer log files	152
6.6	Results of reconstruction using blacklist-pruned system log files	157
6.7	Summary of results for process-access based pruning	159
6.8	Attributes of the best-performing cluster for each use case	164
6.9	Attributes of clusters that resulted in zero recall (no ground truth)	165

Nomenclature

API	Application Program Interface
BNF	Backus-Naur Form
CPU	Central Processing Unit
FSA	Finite State Automata
IDS	Intrusion Detection Systems
OPM	Open Provenance Model
VM	Virtual Machine
FIFO	First-In-First-Out

Chapter 1

Introduction

1.1 Background

In day to day life we often encounter objects with an uncertain background, regardless of whether it is at work or during personal time (e.g. an unmarked parcel, unmarked document). An intuitive response is to question the object's origins (e.g. where is this object from? How did it get here?). One approach is to analyse the object's derivation history – information that depicts how the object arrived at its current state. However, in most situations, this piece of information does not accompany the object and as such, has to be obtained through other means.

In computer science, an object's derivation history is generally known as *provenance of the object* or simply as *provenance*. In a survey on provenance, Carata et al. [2014] conceptualised provenance as a graph that captures the relationship between entities (e.g. people, process or other objects) that are involved in the object's derivation process. This concept is shared by many other discussions on defining provenance, such as those by Cruz et al. [2009] and Moreau [2010]. Due to its relational properties, generating provenance requires specialised logging mechanisms that focus on capturing relational information [Allen et al. 2010b; Carata et al. 2014].

Practical provenance research can be broadly categorised into two aspects: the applications of provenance and the collection and management of provenance. Figure 1.1 briefly illustrates the relationship between these two aspects. Research on the collection and management of

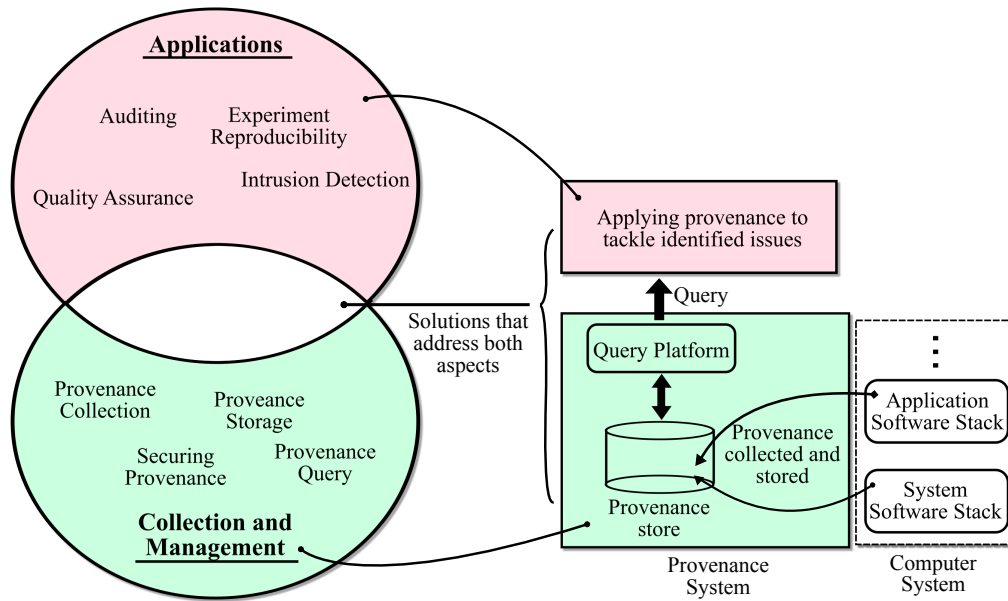


Figure 1.1: Two main aspects of practical research in provenance

provenance focuses on how it can be collected, stored and queried [Cruz et al. 2009; Biton et al. 2008; Ko and Will 2014]. The end results are provenance systems, such as SPADE [Gehani and Tariq 2012] and Komadu [Suriarachchi et al. 2015], that abstract tasks related to the collection and management of provenance from consumers (e.g. researchers or solutions using provenance).

On the other hand, research on the applications of provenance focuses on how it can be used to tackle issues ranging from data quality assurance to experiment reproducibility and system security [Gotz and Zhou 2009; Biton et al. 2008; Dumitras and Neamtiu 2011]. The provenance required to drive these researches and solutions are queried from existing provenance systems.

However, the provenance required may differ based on the context of the research. For example, work by Gotz and Zhou [2009] looks at analysing user activities using provenance depicting high-level user behaviours while Biton et al. [2008] looks at understanding complex workflows using workflow provenance. Existing provenance systems may not

be able to capture all of the required provenance. As a result, solutions such as those proposed by Gotz and Zhou [2009] integrate customised solutions for capturing the required provenance into their proposed provenance management frameworks. Regardless, we can establish that how provenance can be used is inherently dependent on how it can be collected. Such a dependency relationship hints at the importance of provenance collection.

1.2 Motivation

Studies on techniques for collecting provenance by Allen et al. [2010a], Carata et al. [2014] and Coe et al. [2014] noted that state-of-the-art techniques either require modifying software running on the system (e.g. applications or the kernel) or manifest as software that actively monitors different parts of a system and generates provenance during runtime.

However, it is not always possible to collect provenance using the proposed techniques. For example, modifying existing software on production systems is generally considered intrusive and may cause instability in the system (e.g. introduce new vulnerabilities, incompatibility with other existing software or components). Likewise, introducing new software to the system for monitoring and generating provenance would require extensive testing to ensure compatibility with existing software and that it is secure. Deploying new software becomes even more difficult if the party requiring the provenance does not own the system (i.e. third party) as it would involve the issue of whether the software can be trusted. Another downside with existing provenance collection techniques is that they have to be in place and running while the relevant events are happening in order for the appropriate provenance to be captured. This is an issue for situations where provenance collection is an afterthought (i.e. forensic investigation) or when the solutions are not present or running when the event is happening. Without provenance, solutions that rely on access to provenance will not be able to function.

An alternative approach to acquiring provenance is to reconstruct it

from information sources commonly found on systems, such as event logs (i.e. log files)¹ and file meta-data. In comparison to collecting provenance, event logging is common in systems and in many cases, an integral part of the system's day to day operation. Security standards such as the Controlled Access Protection Profile (CAPP) [National Security Agency 1999] require tools that capture and record events describing the states or activities of the system to be deployed. Likewise, upon starting up, the operating system generates information that can be used to deduce the states and activities of the operating system [Ling 2013], in the form of log files and file meta-data. In their work on analysing different types of log files found on systems, Ghoshal and Plale [2013] and Ling [2013] discussed how the analysed log files contain information pertaining to events happening in the system and the involved objects. Having said that, the focus of information captured in log files differs from provenance. Provenance captures relational information describing the derivation history of an object while events in log files describe what is happening in the system at a fixed point. Section 1.3 further elaborates the difference between log files and provenance.

In recent work, Ghoshal and Plale [2013] have shown that it is possible to extract and model parts of provenance information (e.g. disjoint elements of the provenance graph) from application log files. However, the authors do not discuss how provenance that depicts the derivation history of objects can be reconstructed using the extracted information. Hence, this thesis focuses on investigating reconstructing data provenance from log files as an alternate approach to active data provenance collection.

1.3 Log files and Provenance

Provenance information can be stored either as metadata embedded in the object the provenance is describing (i.e. data or workflow) [Groth

¹In the context of this thesis, the term 'event logs' is used interchangeably with the term 'log files'.

1.3 Log files and Provenance

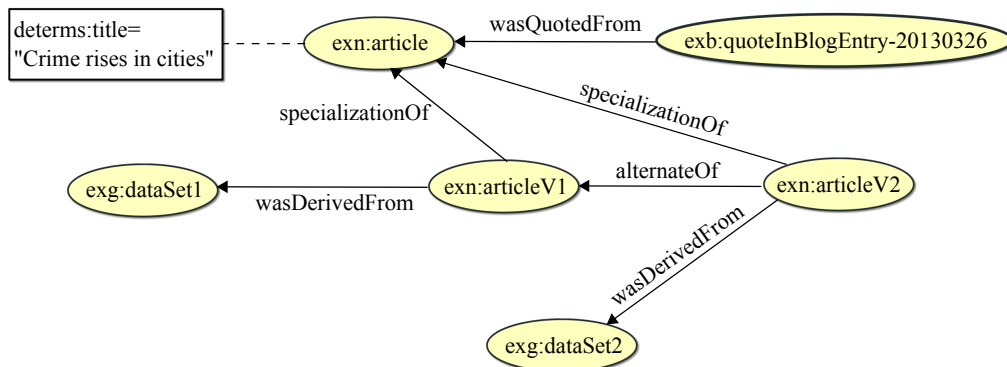
et al. 2012] or as a separate file. Storing provenance as a separate file arguably gives it a log-file flavour. Having said, the difference between provenance stored separately as a file and a log file is the focus of the information captured within each.

Events stored within log files can be seen as entries within a visitor sign-in book at a security guard's booth. Each visitor entry captures information of visitors who have passed through the security booth. However, just by looking at the entry, it is not certain as to where exactly the visitors have been or with whom the visitors have interacted after passing through the security booth. Similarly, log files are considered to be the result of logging mechanisms deployed at specific points within a computer system. Most of these mechanisms only capture events observed within their assigned scope and perspective. For example, a logger for an application only captures events happening in relation to the application and within the application's execution space (e.g. memory region or execution stack). Thus relationships between objects and processes outside the scope of the application (e.g. across other log files) are unknown.

In contrast, provenance describes how an object evolves over time and contains information that allows an analyst to understand and attribute the evolution process. Evolution of the object may span logically across the execution space of multiple applications or systems. Information within provenance differs from those in log files as they each describe either a direct or indirect relationship between other objects and the object the provenance is describing. For example, provenance of a visitor would contain information on the places the visitor actually visited, the people the visitor interacted with and the sequence of activities the visitor went through before each activity.

Due to its relational properties, provenance is conceptualised and commonly visualised as a graph. Figure 1.2a illustrates such a graph, based on an example extracted from documentation on PROV-DM, a provenance model proposed by Moreau and Missier [2013]. Provenance can also be expressed in record format using tuple-like formats that can adequately express relationships between two objects, as shown in Figure 1.2b.

On the other hand, visualising events in a log file (e.g. an application



(a) Sample of a provenance graph adapted from Gil and Miles [2013]

```

entity(exn:article, [dcterms:title="Crime rises in cities"])
entity(exg:dataSet1)
entity(exg:dataSet2)
entity(exn:articleV1)
entity(exn:articleV2)
entity(ex:quoteInBlogEntry-20130326)

wasDerivedFrom(ex:quoteInBlogEntry-20130326, exn:article, [prov:type='prov:Quotation'])
specializationOf(exn:articleV1, exn:article)
specializationOf(exn:articleV2, exn:article)
alternateOf(exn:articleV2, exn:articleV1)
wasDerivedFrom(exc:articleV1, exg:dataSet1)
wasDerivedFrom(exc:articleV2, exg:dataSet2)

```

(b) Corresponding record format of the provenance shown above

Figure 1.2: Examples of provenance adapted from examples discussed by Gil and Miles [2013]

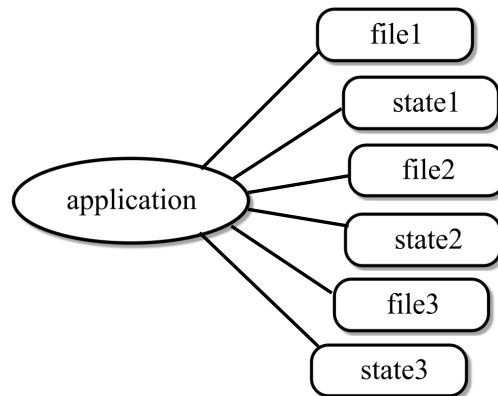


Figure 1.3: A possible graph visualisation of event logs

log) would result in a graph that shows the states and objects the application interacted with directly connected to the node representing the application. This is much like a ‘one-step’ provenance. An example is illustrated in Figure 1.3.

1.4 Thesis Question

Theoretically, disjointed pieces of provenance information modelled from different log files can be pieced together, by studying the *cause* and *effect* relationship between events, to produce a reconstructed provenance of an object. However, how well such reconstructed provenance accounts for the object’s known derivation history remains unclear. Addressing this uncertainty is critical to understanding whether reconstructing provenance from log files can be a viable alternative to acquiring provenance. Formally, this thesis seeks to address the following question:

“Can data provenance be reconstructed in an automated manner from log files found on a computer system and, if so, how does the reconstruction compare to the known derivation history of the data?”

However, it is difficult to obtain the entire derivation history of data objects for evaluation. As such, this research focuses on addressing the thesis question from a more practical perspective, where the reconstructed

provenance is compared against what is known of the derivation of the data. This will be discussed in detail in Chapter 3.

1.5 Scope of Research

Scope of Data Provenance

Surveys by Moreau [2010], Carata et al. [2014] and Simmhan et al. [2005] have discussed different usages of provenance, such as results reproducibility, data verification, workflow checking, data accountability and system security. Ultimately, its usage has an influence on the form and information required in the provenance and the type of dataset required to reconstruct the provenance.

In this thesis, we define data provenance as **the information that depicts the evolution process of a piece of data, including the entities and activities involved in the process**. *Data*, in this research, is viewed at a file level where a file object is treated as a digital container for data.

We consider a reconstructed data provenance to be sufficiently complete if:

- it captures all entities, including any other derivatives of the data, involved in the derivation process starting from the creation of the data
- it captures the relationships between entities and their order, such that the data provenance shows how the data and its derivatives arrive at their current state
- it does not contain information that does not explain or is related to how the data reaches its current state or any of its derivative

Data provenance that satisfies the two stated points would be able to explicitly show how the data is being changed in every step of its derivation history. By analysing the data provenance, analysts would be able

to reach a consistent and correct conclusion on what has happened to the data and how it reaches its current state [Carata et al. 2014; Moreau 2010].

Access to such data provenance would be useful for activities where determining when or how changes were made to the data or identifying the root cause of an error in the data is the primary focus [Zhou et al. 2010]. Digital data investigation [Carrier and Spafford 2004a], data accountability and data verification [Aldeco-Pérez and Moreau 2008] are some examples of data-oriented activities that can potentially benefit from having data provenance.

Having said so, this thesis does not concern with the applications of the reconstructed data provenance. Our focus is purely on investigating the reconstruction of data provenance from log files obtained from a computer system.

Assumptions Made

In line with the motivation discussed in Section 1.2, the following assumptions are made with regards to the state of logging and the computer system:

1. we assume no provenance collectors or any other provenance systems are being actively deployed in the system.
2. we assume that all logging systems adopt its default configurations (i.e. *info* log level), where each logging system outputs information regarding the normal operations of the target of the logging.
3. we assume that only access to the log files are made available.
4. we assume that the log file are trusted and have not been tampered with.

The rationale behind the assumption of default configurations stems from our motivation that provenance acquisition is an afterthought. As a result, we assumed that no intentional steps have been taken to enrich the log files with information that could have enhanced the results

Chapter 1 Introduction

of the reconstruction. Assuming default logging configurations can also provide a base-line understanding on the data provenance that can be reconstructed from log files.

Definition of Computer System

In general, the term *computer system* can refer to different types of systems such as those within a private environment (e.g. private network of computers) or systems in an open environment (e.g. the Internet). However, challenges such as privacy, ownership and trust issues surrounding the dataset that can be used, are more complex when reconstructing provenance in an open environment. These issues, although important, are not directly relevant towards addressing the thesis question. As such, by *computer system* we refer to any system within a private environment, such as desktop computers or a cluster of computers connected within a private network.

Categorisation of Log Files

A study by Ling [2013] noted that application log files are not the only type of log files that can be collected off the system. Other types of log files such as kernel and activities log may also be found on a computer system. To study each type of log file separately in an exhaustive manner would require an extensive amount of data and effort. Instead, we broadly categorise log files according to where they are being generated in the system. The view of a computer system as a multi-layered architecture system, used in many modern day operating system textbooks such as those by Silberschatz et al. [2005] and Tanenbaum and Bos [2008], is adopted for this purpose.

Although views used by different textbooks may differ on the functionality and naming of each layer, they generally agree that an operating system can be structured into hierarchical layers between the hardware and the user.

Each layer is responsible for a specific set of tasks and the communication of information between layers. Such a layered view of the operating system is useful for debugging or even just simply for understanding how various components, software and data structures interact with one another.

Although this research concerns reconstructing data provenance from log files and not debugging, viewing computer systems as multi-layered architectures is still useful. As discussed later in Chapter 3, events generated by different logging mechanisms may differ in the level of detail depending on where the events are generated from. Hence, being able to objectively categorise events based on their granularity aids understanding and modelling of events with relation to provenance.

We broadly categorise the layers between the system kernel and the user into three layers. Log files are then categorised into one of the three layers, based on the point within a computer system in which the events are being observed and captured. The three layers in our multi-layered view are illustrated in Figure 1.4 and defined as follows:

- **User layer**—the user layer represents the user’s perspective. Events observed or generated at this layer relate directly to the activities carried out by a user, such as their behaviours and interactions. These events can possibly be produced through user intervention, such as manual notes and transcripts of screen recordings or automatically by components of an application (i.e. user interface).
- **Application layer**—the application layer sits between the user and system layer. It represents the space in which user-space applications run on a computer system. Log files generated in this layer are automatically generated by the logging functionality of applications or user-space logging mechanisms (e.g. logging mechanisms that do not require elevated privileges).
- **System layer**—the system layer represents the kernel space, where management of the devices such as the Central Processing Unit (CPU) and file system is carried out below the user-space. Log files generated in this layer either require elevated privileges granted

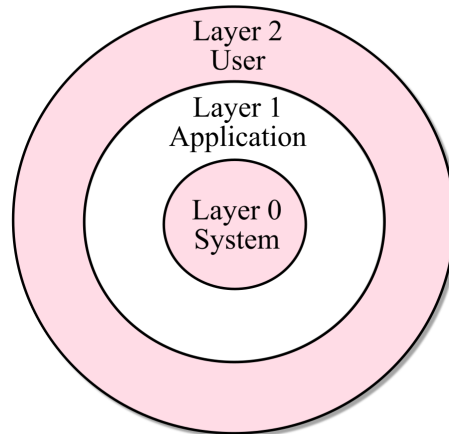


Figure 1.4: Layered view of a computer system derived from the multi-layered architecture view by Silberschatz et al. [2005]

to the logging mechanism or need to be generated natively by the devices (e.g. operating system kernel).

Layers beneath the system layer, such as the hardware layer, are not included in the view because obtaining events from these layers usually requires modifications (e.g. physical modification) to the system. Thus, our layered view is not concerned with layers beneath the system layer as it is uncommon for systems to log events beneath the system layer (i.e. hardware layer).

1.6 Requirements

To address the thesis question, a solution for reconstructing data provenance from log files is required. Drawing lessons from the work of Ghoshal and Plale [2013], we recognised such a solution will require the transformation from log events to provenance graph to be done in multiple steps. As such, the following will be required to achieve such a transformation:

RQ1 - a dataset for experimentation and evaluation. Such a dataset would require both the data from which data provenance can be

reconstructed (i.e. set of log files) and information on the actual derivation history of files (i.e. ground truth).

RQ2 - a provenance model for modelling information extracted from log files into provenance information.

RQ3 - an algorithm or approach for reconstructing data provenance from the modelled provenance information.

RQ4 - a methodology for evaluating the reconstructed data provenance against the ground truth. Such a methodology can also serve as a platform on which the proposed approach can be evaluated against other approaches for data provenance reconstruction.

1.7 Thesis Structure

Figure 1.5 structures the solution for reconstructing data provenance from log files as a workflow, based on the list of requirements listed in Section 1.5. The intention is to modularise the solution for each requirement, such that outcomes from future research (e.g. new algorithms for reconstruction) are interoperable with the work done. The thesis is structured according to the proposed workflow so as to facilitate the description of this research.

Chapter 2 surveys and discusses work related to each of the requirements listed in Section 1.6. The goal is to provide an overview of the research done with respect to each of the requirements and their gaps. Emphasis is placed on the discussion of work related to **RQ2**, **RQ3** and **RQ4**. Detailed review of existing datasets for **RQ1** is done in Chapter 3 in order to keep the discussion in Chapter 3 concise.

Chapter 3 discusses the work done to meet requirement **RQ1**. The chapter first lays out a set of requirements for datasets suitable for provenance reconstruction research and shows the lack of publicly available datasets that meet the requirements. The methodology used in the gathering of a dataset for this research is then presented.

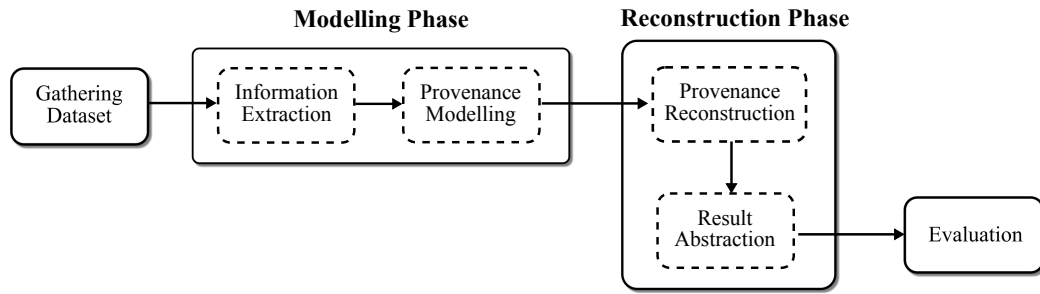


Figure 1.5: Illustration of the proposed data provenance reconstruction workflow

Chapter 4 discusses the model proposed for requirement **RQ2**. The chapter first introduces the proposed multi-layered provenance model. It then describes our approach to mapping provenance relations between the different layers in the proposed model.

Chapter 5 outlines the reconstruction algorithm designed to satisfy requirement **RQ3**. The problem of reconstructing data provenance from the modelled provenance relations is formulated and presented in the first section of the chapter. The rest of the chapter is devoted to describing how the algorithm works.

Chapter 6 presents and shows how the proposed evaluation methodology satisfies requirement **RQ4**. Using the proposed methodology, the thesis question is addressed. Problems in the reconstructed data provenance are also identified and discussed.

Chapter 7 discusses possible future work from this research and concludes the thesis.

Chapter 2

Literature Review

In this chapter literature related to requirements listed in Section 1.6 are reviewed. The structure of this chapter follows the order of the workflow for provenance reconstruction, shown previously in Figure 1.5. Work related to extracting information from log files are first discussed. Following which, provenance models for modelling the extracted information into provenance relations are reviewed. The review then focuses on work relating to provenance reconstruction. Finally, approaches for evaluating provenance reconstruction are discussed.

2.1 Extracting Information from Logs

To reconstruct data provenance from log files, entries in the log files have to be modelled into provenance relations. Information that can be used for identifying entities involved and inferring relationships between the entities have to be first extracted from the log entries.

One of the objectives of log analysis research is to extract patterns that describe expected behaviours of a system or application from the log files. These patterns can then be used to monitor for abnormalities during operation of the system. Research on extracting patterns from log files by Vaarandi [2003], Nagappan and Vouk [2010] and Lou et al. [2010] discussed how each log entry can be divided into two portions: message signature and parameters. The message signature is usually free-form text that describes the event represented by the log entry

while the parameters provide more specific contextual information such as the process ID, process name or state of the application. By extracting and analysing the message signatures, log entries can be classified into different types, thereby allowing patterns to be extracted from the sequence of logs. Techniques such as frequent item mining [Nagappan and Vouk 2010], source code analysis [Xu et al. 2009] and text analysis [Fu et al. 2009] have been proposed for the extraction of message signatures from log entries.

For the purpose of provenance modelling, techniques for extracting message signatures can be applied to log files for separating log entries into the two stated portions. The message signatures can be used for inferring relationships while the parameters can be used for identifying entities involved. However, techniques for extracting message signatures can only resolve heterogeneity of log formats. Our initial analysis of different log files revealed that heterogeneity may exist in the parameter portion of a log entry.

Differences in the implementation of logging mechanisms may cause heterogeneity in the representation and the amount of information logged for each parameter field. Such heterogeneity may be observed even if the logging mechanisms are logging events from the same application type (e.g. different implementations of web servers) or system device. As an example, we look at log messages produced by two different kernel logging mechanisms, Linux Audit Framework (LAF) [archLinux 2012] and Sysdig [Draios Inc 2016], shown in Figure 2.1.

Both Sysdig and LAF log system calls invoked and the parameters used, in the Linux system kernel. Parameters are logged in their raw format into the log file in LAF. In contrast, Sysdig translates captured parameters into human readable format before logging into the log file. This heterogeneity in the representation of information being logged can be observed by comparing parameters highlighted by the corresponding boxes between Figure 2.1a and 2.1b. Such heterogeneity complicates automatic extraction of relevant information from log entries as a program would also need to know when a parameter needs to be translated.

Different logging mechanisms may also produce parameters with dif-

2.1 Extracting Information from Logs

```
type=SYSCALL msg=audit(1431919800.905:5766096): arch=c000003e syscall=43 success=yes
exit=5 a0=3 a1=7ff8b0a8150 a2=7ff8b0a814c a3=0 items=0 ppid=1 pid=2050 auid=4294967295
uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=(none) ses=4294967295 comm="sshd"
exe="/usr/sbin/sshd"
subj=system_u:system_r:sshd_t:s0-s0:c0.c1023 key=(null)

type=SOCKADDR msg=audit(1431919800.905:5766096): saddr=0200D3C8C0A802170000000000000000
```

(a) Entry for *accept* system call in LAF

```
679507;2015-09-18 20:13:51.478914528;pid=1074(sshd);ppid=1;uid=0(root);sys=accept>;
param:

679508;2015-09-18 20:13:51.478938100;pid=1074(sshd);ppid=1;uid=0(root);sys=accept<;
param:fd=5(<4t>192.168.110.50:46814->192.168.110.52:ssh)
```

(b) Entry for *accept* system call in Sysdig

Figure 2.1: Comparing log formats of different kernel logging tools

ferent amounts of information. For example, the dotted red boxes in Figure 2.1a and 2.1b highlight the parameter field that describes the network address associated with a network socket. Converting the network address parameter logged by LAF would show the values of a single Internet Protocol (IP) address and the port number. However, Sysdig records the IP address and port number used by both the local and remote hosts as observed in Figure 2.1b. Due to such inconsistency in the amount of information logged, a level of domain knowledge is required for extracting the right information when processing log files from heterogeneous sources.

Ghoshal and Plale [2013] proposed a rule-based engine for parsing and deriving provenance relations from log files. Since the rules are user-defined, knowledge of the log formats can be incorporated into the rules. However, rule-based engines require users to be familiar with the rule formulation guidelines specific to the engine and how the engine works. This increases the complexity of extracting information from log files.

Although extracting relevant information from log files is the initial phase in our data provenance reconstruction workflow, proposing a general approach that works across different types of log files (including unknown log files) is out of the scope of this research. Instead, we im-

plemented customised parsers for different types of log file. Doing so allowed us to incorporate the extraction and modelling of information into provenance relations in a single implementation, without having to deal with the complexity of rule formulation. We discuss the extraction of information from log files in more detail in Chapter 3.

2.2 Modelling Provenance Relations

Once information relevant to provenance has been extracted from log files, the next step is to model the extracted information into provenance relations. In this section, we review existing provenance models and discuss the gaps in these proposed models.

Both theoretical and practical models had been proposed for modelling provenance. Theoretical models such as those proposed by Green et al. [2007], Luttenberger and Schlund [2014], Souilah et al. [2009] and Cheney et al. [2008] focus on formalising the properties and forms provenance can have. On the other hand, practical models such as Open Provenance Model (OPM) [Moreau et al. 2011], PROV-DM [Moreau and Missier 2013], D-PROV [Missier et al. 2013], Time-aware Provenance model (TAP) [Zhou et al. 2011] and many others focus on modelling information gathered off different systems (e.g. workflow systems, computer systems, distributed systems) into provenance.

Many proposed practical models operate at a flat granularity, treating information observed by different tools to be of the same level of detail. In their discussion on how provenance graphs can vary from one another, Coe et al. [2014] showed how graphs generated from different provenance collectors based on existing provenance models can differ in granularity. Provenance collectors that monitor the execution of a workflow at the application space, such as Vistrail [Bavoil et al. 2005] and Taverna [Hull et al. 2006], capture activities such as reading of a file and running of a script as separate individual events. However, provenance collectors monitoring shared system components (e.g. the kernel or file system), such as Progger [Ko and Will 2014] and SPADE [Gehani and

2.2 Modelling Provenance Relations

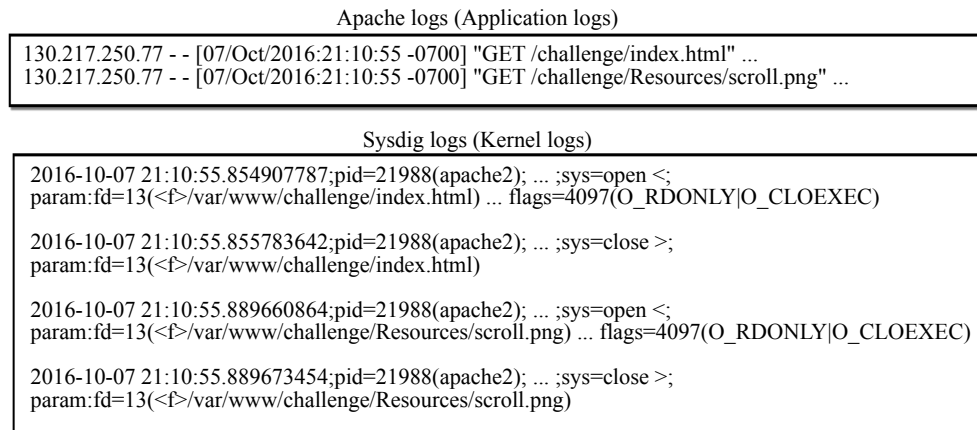


Figure 2.2: Comparing messages between application and kernel logs

Tariq 2012], may observe the same file read as a series of read events. Since most existing provenance models do not consider the granularity layer on which they operate, the events would be modelled equivalently. As a result, the graphs would differ in shape and size even though they are describing the same activities. This is an issue when attempting to compare provenance graphs generated from different granularity layers of a system. Although transformers or similar tools may be used to reduce the impact of granularity during provenance query, it may not produce the right results for all cases. For example, a web server may access the same file back to back due to requests from different remote clients. In such situations, provenance collectors operating at the system layer may capture the two accesses as a series of sequential accesses to the same file. As such, the transformer may not be able to differentiate the series of accesses and collapse the accesses observed by the system provenance collectors into a single access operation.

The difference in granularity described by Coe et al. [2014] can also be observed in log files. Figure 2.2 shows how activities can be captured differently in the application and kernel logs. Requests for two files made to the application, Apache, are logged as two events in the application log. However, each file request is logged as a pair of open and close system call events in the kernel log. If a flat granularity provenance model is used to model both log files, the provenance graph modelled from the application log file will differ from the graph modelled from

the kernel log. This creates an issue when attempting to resolve the two graphs (i.e. determine if the two graphs are the same or looking for duplicated relations).

One possible solution for resolving the differences caused by granularity in log files is to incorporate the concept of multi-granularity into provenance models. This would allow events observed at different granularities to be modelled as separate provenance relations. Association between provenance relations of different granularities can then be expressed through mappings between the different granularity layers in the model. Such a multi-layered provenance model and mapping would allow provenance graphs modelled from different log files to be mapped to and compared at the same granularity. Any comparison at the same granularity would ideally be free of the differences in shape and size of the graph (assuming they are looking at the same activity) caused by the difference in granularity described by Coe et al. [2014].

The need to incorporate granularity when modelling provenance was identified as early as 2007 by Barga and Digiampietri [2007]. From the literature reviewed, two types of abstraction have been identified for achieving multi-granularity in provenance. We term the two forms of abstraction **contextual abstraction** and **structural abstraction**. A quick overview of the abstraction types and the approaches to implement them is shown in Figure 2.3. The two forms of abstraction used are discussed in detail in the following sections.

2.2.1 Contextual Abstraction

Contextual abstraction reduces the amount of contextual information shown in the provenance graph as the granularity approaches coarse-grained, influencing the understanding of elements in the graph (e.g. edges and nodes) in the process. Contextual abstraction can best be understood through the view of Barga and Digiampietri [2007] on differences between granularity layers in a provenance graph. A coarse-grained provenance graph captures the structure of a workflow but not details surrounding each element in the graph. As the granularity be-

2.2 Modelling Provenance Relations

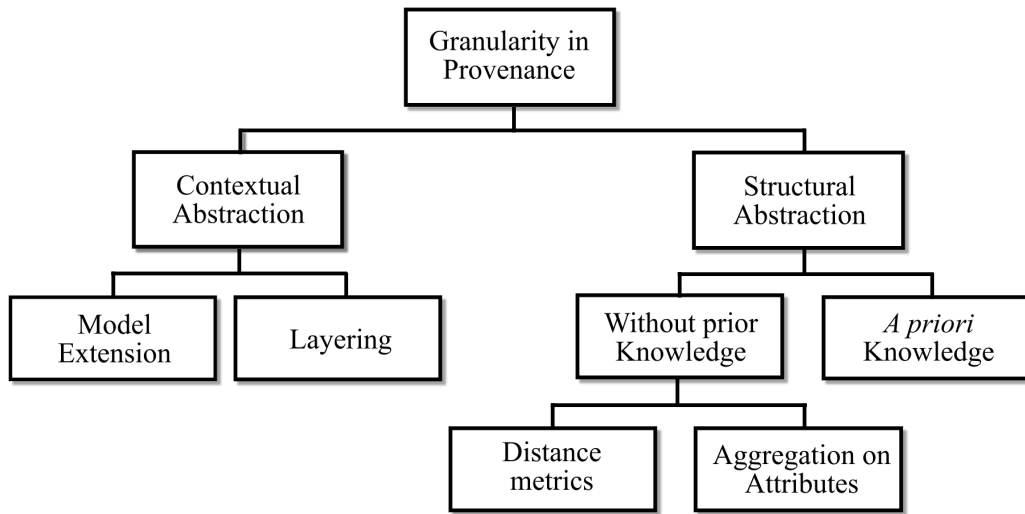


Figure 2.3: Summary of the different types of abstraction in provenance and approaches used to implement them

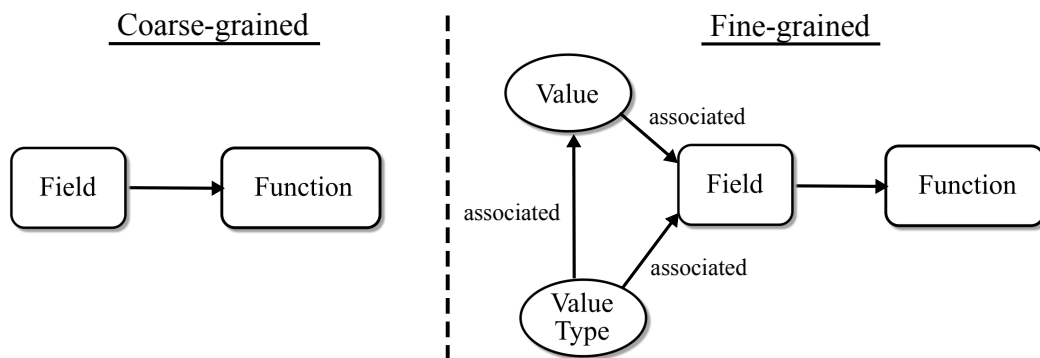


Figure 2.4: Difference between granularity layers caused by “contextual abstraction”

comes more fine-grained, information that describes each element in more detail is added to the graph, thereby giving a more contextual understanding of each element. For example, at a coarse-grained level, an input parameter to a function could be captured as a single node that denotes an input *Field*. However, as the granularity increases, more elements can be associated to the *Field* node to capture more contextual information, such as the value and data type of the *Field*. Such contrast in the level of detail between provenance graphs of different granularity is portrayed in Figure 2.4.

Contextual abstraction in provenance enables users to query prove-

nance with customisable precision, based on their needs. In their discussion on the level of detail that should be captured in provenance graphs, Daniel et al. [2015] linked the requirement on the level of detail to the specificity of the query. The authors argued that by capturing more detail surrounding each element of the provenance graph, queries over a range of specificity, ranging from “*retrieve the list of input for a workflow*” to “*retrieve the list of input of a specific data type for a workflow*” can be executed.

Thus far, contextual abstraction in provenance models have been implemented using two approaches. Barga and Digiampietri [2007] proposed introducing layering into their workflow provenance model. Each layer in the model is responsible for modelling a specific set of information, such as information obtainable from runtime. These sets of information can originate from the separate phases of executing a workflow. For example, L0 (Layer 0) would contain only abstract descriptions of the services used in the workflow. These descriptions can be obtained during the workflow composition stage. In contrast, L3 (Layer 3) would capture details, such as the value and value type, of data observed during runtime execution.

Another approach to implementing contextual abstraction is through extending existing models with specialised classes for capturing finer-grained detail. The extended classes are then appended to the coarse-grained provenance graph as associations, so as to allow elements to be described in more detail. For example, Daniel et al. [2015] proposed PROV-Wf, an extension of the PROV-DM model. Concepts in PROV-Wf that are based on PROV-DM are used to model workflow provenance (e.g. structure of workflow). However, the PROV-DM concepts are not sufficiently fine-grained to allow modelling of contextual information such as value and type of the data. Hence, the authors extended the PROV-DM model with classes designed to allow domain-specific data and information pertaining to the execution of the workflow to be modelled. Likewise, Probst and Hansen [2013] extended the ExASyM model, a model for modelling the spatial and organisational aspects of an organisation proposed by Probst and Hansen [2008], to allow the modelling of security policies

2.2 Modelling Provenance Relations

and activities surrounding data, information which is more fine-grained compared to the structure of a organisation. In doing so, the resulting model, acKlaim, allows analysis concerning data, such as provenance of data, data policy violations and data security to be carried out.

However, even with changes in the structure, interpretation of the interconnectivity (i.e. edges that do not denote associations) between the elements remains constant throughout the layers. With reference to Figure 2.4, one can still deduce that *Field* is still a direct input to *Function*, regardless of the granularity. This is the main difference between contextual and structural abstraction.

2.2.2 Structural Abstraction

Structural abstraction refers to simplification made to the structure of the provenance graph, thereby influencing the interpretation of relationships between elements. Such relationships may represent different semantics, such as flow of information between entities or sequences of execution. For example, in Figure 2.5, one would not know exactly how many processes the data passed through before reaching the recipient process by analysing only the abstracted graph (i.e. coarse-grained).

Structural abstraction in provenance is mostly used to aid analysis of large provenance graphs and to group and annotate segments of a graph. For instance, in visualisation tools proposed by Borkin et al. [2013] and Macko and Seltzer [2011], techniques such as clustering and hierarchical grouping have been employed to summarise a group of nodes, reducing the number of elements users have to analyse. Abreu et al. [2016] argued that by applying structural abstraction to provenance graphs, segments of the graphs can be grouped and annotated with higher-level semantic descriptions. Such descriptions would allow users or algorithms to operate using semantic-rich information to understand and process provenance graphs. For example, a series of interconnected nodes that show the execution sequence of a function can be abstracted and annotated with the function's name. This allows users to relate the execution of the function as part of the provenance.

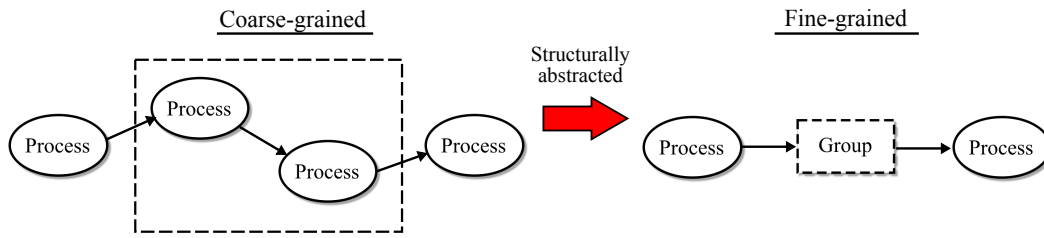


Figure 2.5: Difference between granularity layers caused by structural abstraction

Structural abstraction for summarising a group of nodes or for segmenting a provenance graph can be achieved directly through the use of grouping techniques. These techniques can further be divided in two based on whether a priori knowledge of elements in the provenance graph was used.

Without using a *a priori* knowledge, Macko et al. [2013] proposed the use of metrics such as *closeness*, *betweenness* and *eigenvector* to identify nodes that are inter-related, clustering them together to form local clusters. These local clusters can be used to relate segments of a provenance graph to known activities.

Borkin et al. [2013] applied a time-based hierarchical grouping technique to a provenance graph to create groups of nodes, based on the distance between nodes in the time domain. The creation of groups through measuring temporal distance can then be applied recursively on the groups, until the graph is aggregated to a suitable size for visualisation. Such a visualisation can be said to be hierarchically layered, allowing users to “zoom” into different segments by expanding the aggregated nodes into their original form.

Moore and Gehani [2013] proposed Simple Event Logic (SEL), a domain-specific language for implementing filters that can be used to aggregate streams of provenance events (i.e. nodes, edges and attribute events) generated by provenance collection tools. Differing from previous work, SEL aggregates sequential edges and nodes having the same attributes. For example, when a provenance collector generates a series of provenance events showing a process reading the same file, these events are all buffered and aggregated as they possess the same process ID, file ID

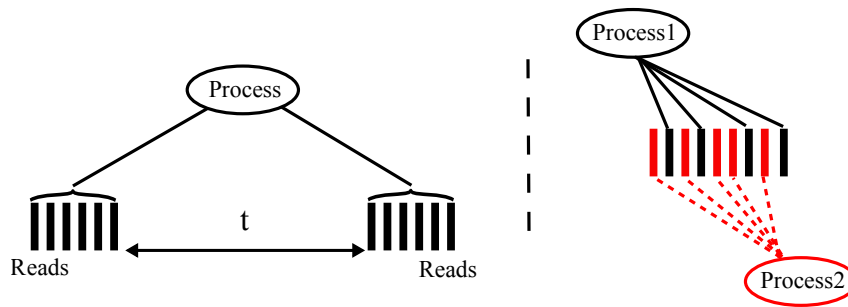


Figure 2.6: Counter use-cases for structural abstractions without using *a priori* knowledge

and activity type. However, once sequential reads are broken by other activities such as a write, the aggregated elements are output and the aggregation is reset. Such a measure is necessary as the authors do not assume that they have an unlimited amount of memory to buffer events.

By not relying on *a priori* knowledge, the discussed approaches can be applied generically to various types of provenance graph. However, they may produce semantically incorrect groupings. Figure 2.6 shows two use-cases where the discussed approaches would produce incorrect groupings.

In the left use-case, variation in the distance between two sets of sequential reads may result in approaches that use distance metrics to erroneously aggregate the nodes into two semantically different nodes. Likewise, the right use-case shows how erroneous aggregation of nodes can happen for approaches that use order of events for determining group membership. Due to events from one process interleaving with events from a different process, structural abstraction using event order would fail to aggregate the events.

Macko and Seltzer [2011] and Buneman et al. [2012] both assumed access to *a priori* knowledge such as the control flow and function call tree of the application the provenance is describing. Patterns that describe the sequence of nodes to expect or relationships between the nodes can be extracted from the *a priori* knowledge. These patterns are then used to guide the abstraction, such that the abstracted graph may reflect the structure seen in the *a priori* knowledge (e.g. functions in the function call tree), as illustrated in Figure 2.7.

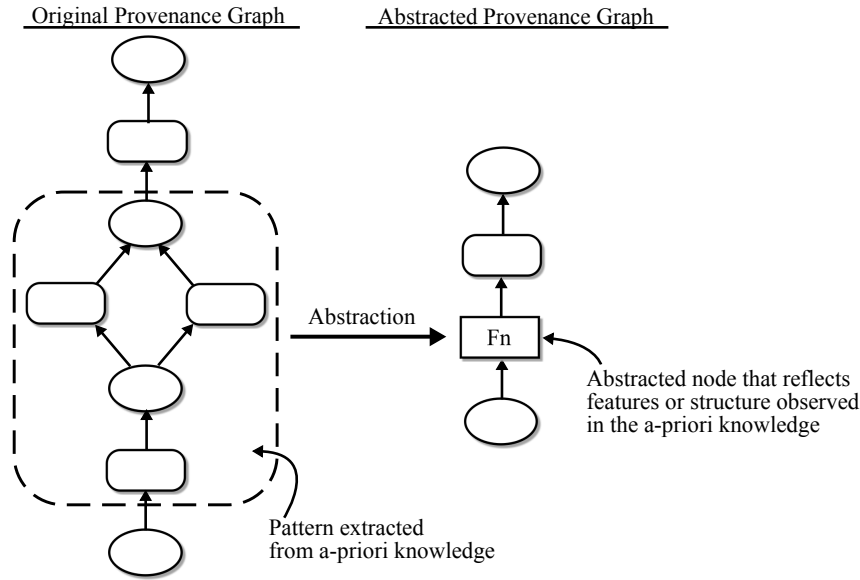


Figure 2.7: Structural abstraction using *a priori* knowledge such as the function call tree

Such abstraction can be applied recursively to the provenance graph, thereby inducing different levels of granularity into the resulting provenance graph. Lee et al. [2013] proposed an approach to split an application’s system trace into units by first deriving the application’s hierarchical loop structure through runtime monitoring. Using the derived loop structure, the application is instrumented to emit specific events denoting the start and end of loops in the system call trace. While this work is done from a non-provenance perspective, provenance models modelling the resulting system call trace can utilise the loop-start and loop-end events to abstract segments of the provenance graph.

Biton et al. [2008] assumed the workflow specifications (e.g. structure, components of the workflow) and the set of relevant elements¹ is known. This contrasts with the previously discussed models where the relevant elements are derived from the call tree. Structural abstraction is achieved by collapsing non-relevant elements together with each relevant element into composite modules. Each composite module is confined to having at most one relevant element, but can have an unbounded number of non-relevant elements. Through such a constraint, the au-

¹In the discussions, the authors describe “relevant elements” as “relevant modules”.

2.2 Modelling Provenance Relations

thors seek to maintain the data flow between relevant elements in each abstracted graph.

The main advantage of using *a priori* knowledge is that semantically meaningful patterns that map lower granularity provenance relations to those of higher granularity can be extracted from the knowledge, resulting in more accurate abstractions. For example, from an application's call tree Buneman et al. [2012] were able to extract the set of directly connected nodes that can be mapped to a higher granularity provenance relation. The extracted set can be viewed as the pattern that maps to the higher granularity provenance relation. Counter use-cases, such as those shown in Figure 2.6, can be resolved using the patterns extracted from *a priori* knowledge as the nodes to expect are defined by the patterns. However, access to a *a priori* knowledge cannot always be assumed.

Arguably, being able to structural abstract provenance graphs modelled from log files is more critical, as compared to identifying their contextual abstraction, when comparing or resolving graphs modelled from different granularity. For example, an application reading from a file can be logged as a single *read* event in the application log file. However, the same read can manifest as a series of *read* operations in the kernel log files. Modelling events from both log files would result in two structurally non-equivalent provenance graphs even though they are describing the same file read event.

Structurally abstracting provenance graphs modelled from log files using models that do not assume *a priori* knowledge can potentially produce semantically incorrect abstractions like those illustrated in Figure 2.6. On the other hand, models based on *a priori* knowledge of elements in the provenance graph can only abstract graphs describing those elements. For example, models based on workflow specifications for *workflow A* will not work for other workflows. For reconstructing provenance from log files, obtaining *a priori* knowledge of applications running on a system is infeasible due to the number of possible applications existing on a system. Obtaining the required *a priori* knowledge of each application is also subject to the availability of information concerning the application, such as its source code or binary. Such information cannot

be assumed if only access to log files is made available.

Another solution for structurally abstracting provenance graphs modelled from log files is by defining patterns that describe known generic operations or behaviours of applications (e.g. read, write). Abreu et al. [2016] proposed a conceptual workflow that would potentially allow such patterns to be extracted without relying on *a priori* knowledge. A provenance graph is segmented into different segments. Semantically meaningful patterns are then extracted from each segment using analytic techniques such as feature extraction. However, as the authors have pointed out, discovering segmentation strategies for provenance graphs is an open research issue in provenance research. In Chapter 4, we present our proposed multi-layered provenance model and discuss how a set of patterns that map provenance relations between different granularity layers is derived and implemented.

2.3 Provenance Reconstruction

Provenance reconstruction can be broadly seen as reconstructing the sequence of events that explain the resulting state of an object or incident. In this section, research that relates to this view in the areas of provenance, digital forensics and workflow planning are reviewed and discussed.

2.3.1 Reconstruction in Provenance

A call for solutions for reconstructing provenance was issued as a recent challenge in the provenance challenge series [?]. The objective is to address the impractical assumption that provenance can always be collected or collected completely. The challenge is symbolic of the problem being recognised as an important issue in provenance research by the community as the challenge series has been a platform for addressing key challenges. Past issues addressed include the need for a standard model for provenance across different fields and provenance interop-

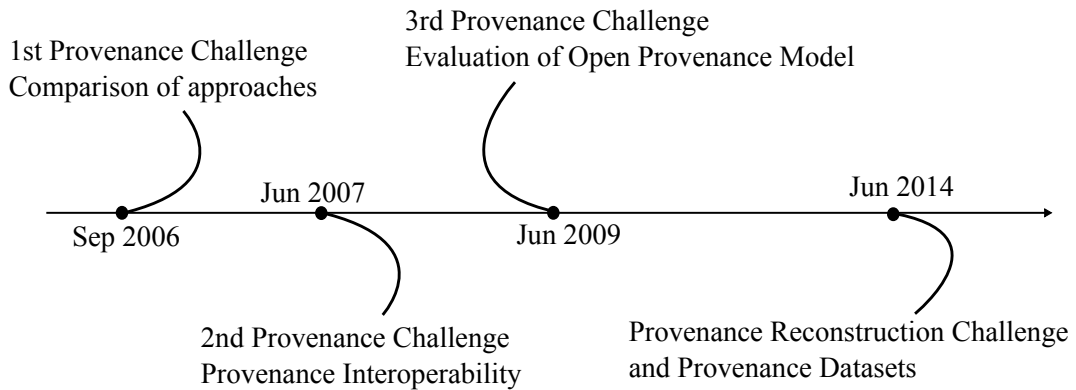


Figure 2.8: Timeline and focus of the series of provenance challenges [Moreau et al. 2010; Groth 2014]

erability; issues that affect the application of provenance. Figure 2.8 illustrates the series of issues addressed. In this subsection, we categorise and discuss research on reconstructing provenance according to the methodologies the approaches take.

Inference-based

Inference-based approaches reconstruct provenance by inferring the missing segments or the complete provenance from existing provenance knowledge, such as the provenance of other similar objects. Huq et al. [2011] proposed using coarse-grained workflow provenance, collected during setup, from workflows working with streaming data to reconstruct fine-grained data provenance. The coarse-grained provenance which captures parameters, input sources and other information regarding the processing elements in the workflow is used to reconstruct a processing window. The window is then used to infer which data tuples in the input data stream are the contributing input data tuples that resulted in the queried output tuple. While the proposed solution allows inference of the contributing data tuples, it assumes that the set of transformations the data undergoes (e.g. the workflow template) before reaching its final state is known.

Both Govindan et al. [2011] and Zhao et al. [2011] discussed how missing parts of a provenance graph can be reconstructed by inferring the

Chapter 2 Literature Review

missing segments from the provenance of objects similar or related to the object described by the partial provenance graph. Govindan et al. [2011] considers the scenario where data provenance may be incomplete due to attacks from malicious users or nefarious nodes in the computer network. Two scenarios where provenance could be lost were addressed: complete loss of provenance information for a network node (i.e. missing nodes in the provenance graph) and partial loss of provenance information on a network node (i.e. incomplete information on a node in the graph).

To reconstruct partial provenance loss, the missing information was inferred from past complete provenance of the node. In the event where the missing information could not be inferred, the provenance for that node is deemed lost and discarded; to reconstruct the provenance graph with missing nodes, a list of possible paths the data could have taken through the network is derived by computing the reachability set or inferred from past behaviour of the network. The most likely path is then selected based on total length of the graph or by the order of common sequences. Zhao et al. [2011] reconstruct partial loss of provenance for data items used in reservoir engineering using an approach similar to how Govindan et al. [2011] reconstruct partial provenance loss. Their approach first searches for data items which are semantically similar to the data item with incomplete provenance. The missing segments are then inferred from the complete data provenance of those semantically similar data items. The underlying assumption is that semantically similar data items are processed in a similar manner.

These approaches have demonstrated that missing segments or the complete provenance graph can be inferred from the provenance of related objects. However, they rely on access to existing provenance knowledge. Our research assumes no provenance collectors were being deployed. Hence, it is assumed that no such knowledge is available.

Planning

Assuming a library containing all possible transformations is available, Groth et al. formulates reconstructing data provenance that shows the

2.3 Provenance Reconstruction

transformation process of a piece of data as a planning problem [Groth et al. 2012]. A prototype based on A* search and heuristic functions based on edit distance is implemented. The prototype searches for possible sequences of transformations that could explain how the data reaches its output state from its initial state. However, the authors highlighted the search space for possible sequences being unbounded as one of the challenges that needs to be resolved. Another challenge of the proposed solution is that the search may yield incomplete results if there are transformations not defined in the library.

Formulating provenance reconstruction as a planning problem has two requirements. First, like in workflow planning, the set of possible transformations or provenance relations from which the provenance can be reconstructed is required. For our research, the set of provenance relations can be modelled from the log files. Second, the end goal needs to be clearly defined and known. However, discerning which objects are relevant to the data provenance from a dataset on which we assume zero knowledge is difficult. As such, formulating the problem of reconstructing data provenance from log files as a planning problem would also face the challenge of an unbounded search space.

Code Analysis

Code analysis approaches reconstruct the provenance of a workflow by analysing the source code or binary of a given piece of software. Huq et al. [2013] reconstructs the workflow provenance of a script by first parsing the program code in the script and constructing an abstract syntax tree from the code grammar used. Based on the syntax tree, the authors' implementation prompts the user for operation-specific information, such as whether an operation is reading or writing persistent data. Corresponding elements in the syntax tree are then annotated with the user's responses. Finally, the tree is converted into a workflow provenance graph.

Viewing provenance from a phylogenetic tree perspective, Dumitras and Neamtii [2011] tackle the problem of malware profiling by attempting to reconstruct the phylogenetic tree for families of malware. Through

Chapter 2 Literature Review

analysing the source or binary, malware are classified into families based on similarities in how they operate. By ordering the different versions of malware, the resulting phylogenetic tree can show the evolution of malware within each family.

Since the functionality of an application can be inferred from its source or binary (i.e. execution steps or sequence of transformations of the application), data provenance that shows how an application transforms a piece of data can be reconstructed. However, in situations where the data is propagated or used by multiple applications, the reconstructed provenance would resemble a set of disjointed graphs. This is because the relationships inferred from the sources would only capture activities within each application's scope. Inter-application communication would fall outside of such scope. Take for example the case where the output of one application is fed directly to the stream input of another application. Relationships inferred from the latter's source would not show the origins of the input data is from the first application since the input source will be treated generically as a stream object.

Content-based

Content-based approaches reconstruct the provenance of a file by comparing the difference in content between files. This is based on the assumption that files which are related (i.e. different revisions of a file) would have similarities in their content. The general approach first reconstructs the relevant entities of the graph by correlating files based on similarity of their content. Using temporal information such as time of creation, correlated files can then be arranged into a time-ordered coarse-grained provenance that shows how a file evolve through different revisions over time. Measuring content similarity may be done based on the actual content or through the semantic properties of the content.

Magliacane [2012], Deolalikar and Laffitte [2009] and Aierken et al. [2014] measured content similarity using distance measures such as cosine similarity and longest common subsequence. Interestingly, in evaluating a prototype developed by Magliacane and Groth [2012] against a

set of clinical documents, the authors reported issues with deriving the appropriate temporal order due to differences in the file metadata. This is a major issue for content-based approaches that rely on timestamps for inferring file revision order as different file systems may update the file metadata in a different manner as discussed by Knutson [2016] and in ForensicsWiki [2015].

Nies et al. [2012] considered the case where revisions of a file may differ in content length. Instead of computing the content similarity distance of two news articles, semantic properties associated with each news article, such as *Named Entities*, are extracted. The coarse-grained provenance is then generated by correlating news articles based on similarities between their semantic properties. Fine-grained provenance is then inferred by identifying and modelling each difference between the content of two correlated news articles. Modelling of content difference is done using relations defined in the PROV-DM model [Moreau and Missier 2013]. The proposed approach was then extended by adding new parameters to how similarity between documents is measured and applied on the 2014 provenance reconstruction challenge dataset by Nies et al. [2016].

In a separate work, Nies et al. [2015] adapted their approach for reconstructing provenance between news articles to social media messages. Fine-grained provenance is reconstructed using knowledge such as how users are interconnected on social media, message IDs, timestamps and other meta-data. The reconstructed fine-grained provenance is able to show how a message or news is diffused from the original source within the connected network of users (e.g. network of friends and friends-of-friends). However, it does not cover messages that were copied or revised manually (e.g. not shared or retweeted through the social media software). To reconstruct coarse-grained provenance that captures the missing relationships, tracked copies of messages in the fine-grained provenance are first removed from the set of all messages. A similarity matrix for all messages is then built using a feature model and semantic similarity function (e.g. TF-IDF and cosine similarity). The messages are then clustered and messages within each cluster are ordered based

on timestamps to produce the coarse-grained provenance. Finally, by integrating both forms of provenance, the graph that shows both message diffusion between users who are friends and possible relationships between messages not captured by the social media software is reconstructed.

Although content-based approaches have demonstrated the possibility of reconstructing provenance, they are reliant on being able to access and compute the content of files. In a computer system, given the large number of files that can exist, it is computationally expensive to correlate every file. Data communicated between processes using volatile memory are also transient. This implies that acquiring the data required to establish communication between processes may not be possible. Finally, while content comparison allows the inference of how data is being transformed, the question of which processes or applications are responsible for the transformations remains. In contrast, studies by Ghoshal and Plale [2013] and Ling [2013] showed information relating to events happening and involved objects are being captured in log files. This information leads to reconstructing provenance that shows the activities (e.g. read, write) and entities involved in the transformation to be promising.

2.3.2 Digital Forensics

Part of a digital forensic investigation involves having to reconstruct the sequence of events happening at a digital crime scene² such that it can be used to reason about or infer how a crime was committed. Reconstruction is usually based on digitised evidence such as computer logs, saved states of a system or even disk images. This is directly relevant to our research on reconstructing data provenance from log files gathered from the system. In this section, we identify two schools of thought surrounding research in reconstructing the sequence of events in digital forensics: timeline and event reconstruction.

²In digital forensics, the crime scene usually refers to the computer system under investigation.

Timeline Reconstruction

Timeline reconstruction aims to temporally order events such that further reasoning (usually manual) can be applied on the timeline to infer the cause of the observed incident (e.g. digital crime or anomalies). The general approach is to first extract events from the digital evidence gathered from the system. Events are then ordered based on their timestamp into a linear time sequence that reflects the order in which the events took place. Approaches mainly differ in how events are being extracted and processed.

Chabot et al. [2014], Hargreaves and Patterson [2012] and Buchholz and Falk [2005] utilised different types of parser to extract timestamped events, along with other relevant information from collected evidence. Extracted events are processed into event nodes that encapsulate information relevant to each event (e.g. start and end time, type, source of extraction) before inserting it into the timeline.

Khan et al. [2007] addresses the issue of where extracting events³ in the past becomes harder as the evidence becomes older (e.g. repeated execution of an application would result in modification to the metadata of files the application accessed). To overcome this issue, the authors proposed the use of neural networks for automatic classification and extraction of the execution time frame and files manipulated for a set of known applications. Training of the neural networks is done using file system activities that describe the behaviour of each application in the set. Experiments conducted using the implementation showed that a highly accurate timeline of the applications can be reconstructed despite multiple applications and multiple file accesses happening at the same time. However, the reconstruction only works for applications the neural network is trained for. Identifying new applications would require the neural networks to be retrained.

While a timeline captures the sequence between events, its linear properties are not intuitive for capturing relationships between events, which is key to understanding the 'hows' and causes of events. This is espe-

³In this case, the authors are looking at traces of execution of an application.

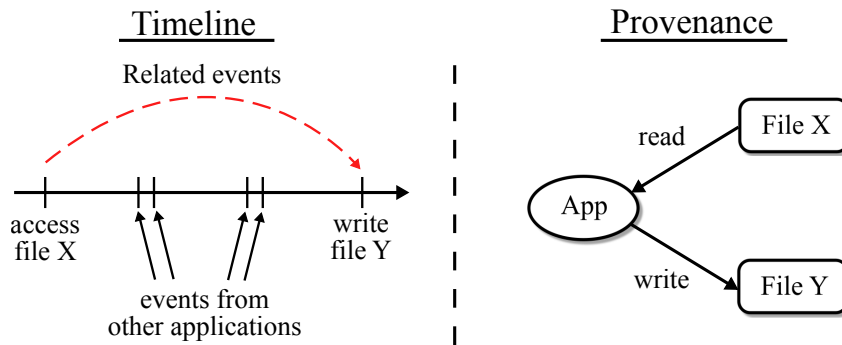


Figure 2.9: Case where linearity of timeline is not intuitive for deducing relationship between events

cially the case if two related events are far apart on the timeline or that there are many other events taking place between the two events (e.g. noise), as illustrated in Figure 2.9. As a result, approaches for timeline reconstruction are insufficient for reconstructing provenance.

Event Reconstruction

Deviating from timeline reconstruction, event reconstruction approaches reconstruct each event⁴ as a separate entity. Such an entity would contain information that describes the event, extracted from different evidence. For example, FACE, an event reconstruction implementation by Case et al. [2008], reconstructs an entity that describes a process as an event. Such an entity would contain information on the files the process accessed, the sockets it opened and other contextual information. From a provenance perspective, each entity is equivalent to a node in the graph or a small segment of the graph.

Once events are reconstructed, reasoning can be applied such that the sequence of events that led to an incident can be reconstructed (e.g. *cause* and *effect*). Carrier and Spafford [2004b] defined a conceptual

⁴In the reviewed literature, the definition of the unit term 'event' varies between two definitions: 1)Event is equivalent to an entity. 2)Event as in a singular activity executed by an entity. It is not our goal to provide a unified understanding of 'event' in this review. As such, we use 'event' to interchangeably refer to both definitions, based on the context of the discussed work. Having said that, this variation of definition of term has little impact on the overall discussion of reconstructing the sequence of events.

framework that consists of the following five phases to achieve this end goal.

1. Evidence examination—Identify and extract objects likely to be related to the investigation, its relevant information and characteristics from the gathered evidence.
2. Role classification—Classify objects as the *cause* or *effect* of an event, based on the information and characteristics associated with the object.
3. Event construction and testing—Different objects and events are correlated, such that each event has a *cause* and an *effect*. The aim is to allow reasoning on how a *cause* object could have brought about changes that resulted in the *effect* object, through the known event.
4. Event sequencing—Events that are reconstructed in the previous phase are sequenced together, forming a chain of events. The end result may be a list of possible chains of events or even series of small event chains that needs to be sequenced further.
5. Hypothesis testing—Each chain of events is tested with the discovered evidence or known facts about the crime to see if the chain of events can explain the crime.

The majority of work on event reconstruction focuses on phases one to three. Tools such as ECF by Chen et al. [2003], FACE by Case et al. [2008] and FIRESTORM by Ahmad and Ruighaver [2002] place emphasis on providing a platform that collects and associates information relevant to an event from different sources. Phase four, the most important phase, where the sequence of events is reconstructed, is however delegated to the user. This is achieved through tools, such as FACE by Case et al. [2008], that allow users to query or visualise the reconstructed events for manual correlation and reasoning. However, as pointed out by Schatz et al. [2004], such tools would not scale as the number and complexity of events to be investigated increases. An increase in the number and

Chapter 2 Literature Review

complexity of events can be due to several causes, such as increased workload on the system under investigation, investigation of an incident that happens across long periods of time or inclusion of multiple sources of evidence.

To overcome the increase in volume and complexity of events, rule-based correlation engines such as SEC by Vaarandi [2002] and FORE by Schatz et al. [2004] were proposed. Events extracted from evidence are formulated as knowledge. User-defined rules that describe known patterns or anomalies in the form of *antecedents* (e.g. cause) and *consequences* (e.g. effect) are then fed to the correlation engine and executed over the knowledge base. The engine can return either the sequence of events that matches the defined rules or return the effects of the matched rules. Rule-based correlation engines provide the flexibility for users to update rules periodically, thereby increasing the coverage of incident types that can be detected. However, the specificity of the rules affect the results returned. If the rule is overly strict, the engine may miss other relevant events. In contrast, if the rule is too general, the engine may return a large number of false positive results. In addition, knowledge of the patterns or anomalies is required for the formulation of rules. This makes it difficult to detect an unknown sequence of events.

Another approach to event reconstruction is through profiling the digital fingerprints of specific applications performing different actions. Profiling is achieved by monitoring changes in the timestamps in the metadata of files (e.g. access, modify, create timestamps) in the file system, as discussed by Kalber et al. [2013] and James et al. [2011]. By matching profiles showing sequences of file activities expected from applications to the file metadata of the current state of the system, analysts can reconstruct which applications are running at the instant of time of the snapshot. However, the disadvantage is that the profiles can only identify when the last time the application executed as the file metadata would have been overwritten by the latest file activity. Profiling digital fingerprints of applications also suffers from an inability to detect unknown applications running on the system

Phase four of the conceptual framework defined by Carrier and Spaf-

ford [2004b] is more relevant to provenance reconstruction as the key challenge is establishing the relationship between entities. However, we note approaches using rule-based engines require knowledge on structure of the provenance graph or expected behaviour of the entities for rule formulation. We do not assume such knowledge in this research.

2.3.3 Workflow Planning

Our review exercise on techniques that reconstruct sequences of events led to work surrounding workflow planning. It is not our intention to provide an extensive review of the work in this area, but rather to gather inspiration on how relationships between entities can be reconstructed.

In his article on the application and implementation of classical planning in game development, Vassos [2012] defined the planning problem as:

“Given the initial state of the world (world model), a set of actions that describe how the world changes in terms of preconditions and effects and a goal condition, find the sequence of actions (plan) such that when applied one after the other, they would transform the current state description into one that satisfies the goal condition.”

The Stanford Research Institute Problem Solver (STRIPS) by Fikes and Nilsson [1971] is an implementation of the classical planning technique. In STRIPS, each action consists of a set of preconditions that must be met in the world model before the effects of the action can be applied onto the world model and a set of effects in the form of a *delete* and *add* list. The *delete* list contains the set of conditions that are to be removed from the world model and the *add* list contains the set of conditions that are to be added to the world model. In general, STRIPS works as follows:

1. A theorem prover is used to prove whether the world model, M_0 satisfies the goal conditions, G . If the prover is able to find a valid proof, the planning is deemed complete and any applied actions will be output as the plan (if the planner is able to prove M_0 , then the

problem is solved trivially). If not, STRIPS treats the incomplete proof as the difference between the state of the world model and the goal conditions.

2. STRIPS searches for actions with effects that can reduce the difference between the state of the world model and the goal conditions. The preconditions of such actions then become subgoals.
3. STRIPS uses the theorem prover to check if there are instances of the selected action that can be used to prove the subgoal. If not, Step 2 is applied onto the subgoal. If it can, the action is applied to transform M_0 to a new world model, M_1 . From here, the planner goes back to Step 1 using M_1 as the new world model instead.

Although different implementations of classical planning may adopt different searching algorithms and heuristics, these implementations generally are exposed to the issue of a large search space as the number of valid actions increases in each state of the world model. In addition, classical planning also suffers from an unbounded length of the plan and does not guarantee an optimal solution. We note that these issues have been raised by Groth et al. [2012], where they investigate formulating reconstructing data provenance as a planning problem.

While there are other strategies proposed for workflow planning, such as temporal planning [Chen et al. 2006; Cushing et al. 2007] and Hierarchical Task Network planning (HTN) [Nau et al. 1999; Georgievski and Aiello 2014], the problems and assumptions that these strategies are designed for renders them irrelevant to our scenario. For example, the assumption for HTN is that the abstract structure of the plan is known. The plan can then be broken down to different subtasks, such that the sequence of actions required to achieve each task can be discovered separately. However, we do not assume that the structure of the provenance graph is known. Likewise, temporal planning was designed to solve problems where time constraints surrounding the concurrent execution of different actions are considered. However, we do not assume such constraints in our reconstruction scenario.

2.4 Evaluating Provenance Reconstruction

In the process of reviewing literature on provenance reconstruction in Section 2.3, we noted the inadequate discussion on the evaluation of results. Details such as how are the results measured and the metrics and dimensions used in the evaluation are lacking. Discussions such as those by Huq et al. [2011], Dumitras and Neamtiu [2011], Groth et al. [2012], Zhao et al. [2011] and Nies et al. [2015] either did not discuss or only briefly mentioned the results of their evaluation. The methodology that led to the authors arriving at the stated results was not documented. This makes it difficult to replicate or compare other approaches with the discussed work. However, some of the authors also highlighted the lack of a suitable dataset for evaluation as the main reason for the absence of evaluation.

Govindan et al. [2011] evaluated the performance of their algorithm for reconstructing missing segments of a provenance graph under different loss ratios⁵ using the Receiver Operating Characteristics (ROC) curve. The algorithm was repeated a hundred times using a random starting point for each loss ratio. The rate in which the missing provenance segments are correctly reconstructed is then plotted against the rate of incorrect reconstruction results to form the ROC curve. By plotting the ROC curve for each loss ratio on the same chart, the performance of the algorithm under different loss ratios can be deduced quickly from the chart. Instead of a ROC curve, Zhao et al. [2011] directly plotted the *precision* of their algorithm against different loss ratios. However, from the discussion, it is not clear how *precision* was calculated or what it means. The authors only briefly state that it was calculated by comparing the reconstructed provenance with the ground truth. In both discussions, the proposed algorithms were evaluated based on whether the missing segments were correctly reconstructed. The metrics used (e.g. ROC curve, precision-to-loss ratio) do not provide insights to the quality of the reconstructed provenance, such as the amount of redundant information (i.e. noise) in the output.

⁵Loss ratio refers to the ratio between the size of the lossy provenance graph and the provenance graph without loss (ground truth).

Most content-based approaches, such as those by Aierken et al. [2014], Nies et al. [2012] and Magliacane and Groth [2012], assume that the entire provenance graph is not available (i.e. 100% loss). As a result, the approaches are not evaluated against varying loss ratios. Instead, evaluation is centred on understanding how well the reconstructed provenance represents the ground truth using metrics such as *precision* and *recall*. Unfortunately, how *precision* and *recall* are computed and the significance of those metrics were not discussed in the experiments.

Knowing the approach to how the values for *precision* and *recall* is derived is important as there can be different methods for calculating those metrics. These methods can potentially result in different values. Manning et al. [2009] described *precision* as “the fraction of retrieved documents that are relevant” and can be computed as follow:

$$\text{Precision} = \frac{tp}{(tp + fp)}$$

where tp is the number of retrieved items that are relevant (true positive) and fp is the number of retrieved items that are not relevant (false positive) and can be computed by subtracting fp from the total length of the result. However, in their research on proposing an automatic machine translation evaluation methodology for computational linguistics, Papineni et al. [2002] highlighted two different approaches to measuring tp . Assuming matching unigrams, traditional approaches to calculating precision would equate tp to be the $\max(\text{Count}_{\text{result}}, \text{Count}_{\text{truth}})$, where Count is the number of matches in the respective sentences between the result and the ground truth. In some cases, the traditional precision calculation would yield high precision but questionable translations, such as the case shown in Figure 2.10a. The authors argued that this approach is different to how humans would distinguish the performance of a translation.

Instead, the authors proposed to take the minimum count between the result and the ground truth when computing tp . The authors remarked that counting the minimum is closer to how humans distinguish a good translation from a bad one as compared to the traditional approach to

2.4 Evaluating Provenance Reconstruction

Ground Truth: The fox jumps over the wall Translation Result: The fox the fox the fox wall $tp = 7$, $precision = 7/7$

(a) Traditional approach takes the maximum match count when computing true positives

Ground Truth: <u>The fox</u> jumps over <u>the wall</u> Translation Result: <u>The fox</u> <u>the fox</u> the fox <u>wall</u> $tp = 4$, $precision = 4/7$
--

(b) Considering the minimum match count when computing true positives

Figure 2.10: Difference between traditional approach to computing true positives and approach proposed by Papineni et al. [2002]

computing tp . Figure 2.10b illustrates the outcome of the proposed modified approach. While the discussion surrounds computing $precision$, it applies to $recall$ too as tp is also a key variable for deriving $recall$, as shown in the formula below:

$$Recall = \frac{tp}{(tp + fn)}$$

where fn (false negative) is the number of items falsely deemed non-relevant. Although subtle, these two different approaches yield different evaluation results. As such, it is important that the approach to how the metrics are computed be documented in any discussion on evaluation.

Cheah and Plale [2014] suggested that quality of a provenance graph can be deduced by measuring the correctness and completeness of the graph. The authors defined correctness to be associated with the degree in which the graph is free of errors and inconsistencies and completeness to the ratio of loss or over-completeness of a graph in comparison to the expected provenance graph. A framework that evaluates the correctness and completeness of a provenance graph is proposed. Correctness

is evaluated by searching for conflicting annotations⁶ and timestamp inconsistencies between connected nodes⁷. Completeness is evaluated by comparing the provenance graph with knowledge of the graph, such as a workflow template, to determine if there are missing elements. The errors found were then averaged over the expected nodes and edges into a single scoring metric that is used to denote the quality of the graph.

While the dimensions used by Cheah and Plale [2014] can also be applied to evaluating a reconstructed provenance graph, consolidating the outcome to a single scoring metric is insufficient for highlighting the aspect the reconstruction failed poorly in. Also, the authors' proposed framework only factored missing segments into their quality metric. We argue that noise (e.g. extra provenance relations in the graph) is also a factor that should be included in the evaluation as it reduces quality and hinders comprehending the provenance graph.

In Chapter 6, we address the lack of evaluation methodology for provenance reconstruction research. We argue that addressing this gap is critical for future research as an open and clear evaluation methodology is key for comparison and performance evaluation of proposed solutions.

2.5 Dataset for Provenance Reconstruction

An important research gap highlighted by some of the work reviewed in Section 2.4 is the lack of a suitable dataset for evaluation. Magliacane and Groth [2013] discussed their efforts identifying such a dataset for their research on reconstructing file provenance given a set of files whose relationship is unknown. Their survey was based on the following use cases:

1. Detecting plagiarism, text and multimedia content reuse.

⁶In the context of their work, the authors referred to annotations as the contextual information associated with each node in the graph, in the form of name-value pairs.

⁷Since in a provenance graph, each edge is equivalent to a dependency relationship between the two connected nodes, timestamp inconsistencies can be checked by observing the causal order of the two nodes.

2. Connecting publications with related data such as other research data and blog posts.
3. Tracking the evolution of scientific knowledge and discourse through publications and informal communications between scientists.

As a result, the datasets surveyed were mostly corpora of text documents and multi-media datasets. Since the scope of this research is on computer system log files, the findings reported by the authors were not applicable. Having said that, the authors broadly discussed some guidelines for selecting a dataset for evaluating provenance reconstruction. Based on these guidelines, we defined a list of requirements for identifying datasets that can be used for our research. To keep the discussion concise, the requirements for the datasets and our work on surveying publicly available datasets is discussed in Chapter 3.

2.6 Summary

In this chapter, work related to the requirements identified in Section 1.6 are reviewed. Figure 2.11 illustrates the categories of work reviewed.

Techniques used in log analysis, such as frequent item mining and text analysis, can be used to extract the message signature and parameters portion of log entries in a log file. However, our review showed that these techniques do not handle the heterogeneity found in parameters caused by logging mechanisms. Extracting the correct information from the parameter portion of a log entry is critical to modelling log entries into provenance relations. Having said that, proposing a general solution for resolving heterogeneity in both the message signature and parameter portions of a log entry is not within the scope of our research. Instead, we combine the extraction of information from different log files together with the modelling of such information and implemented them as customised parsers. We elaborate upon our approach in Chapter 4.

A provenance model is required to model information extracted from log files into provenance relations, such that they can be used for recon-

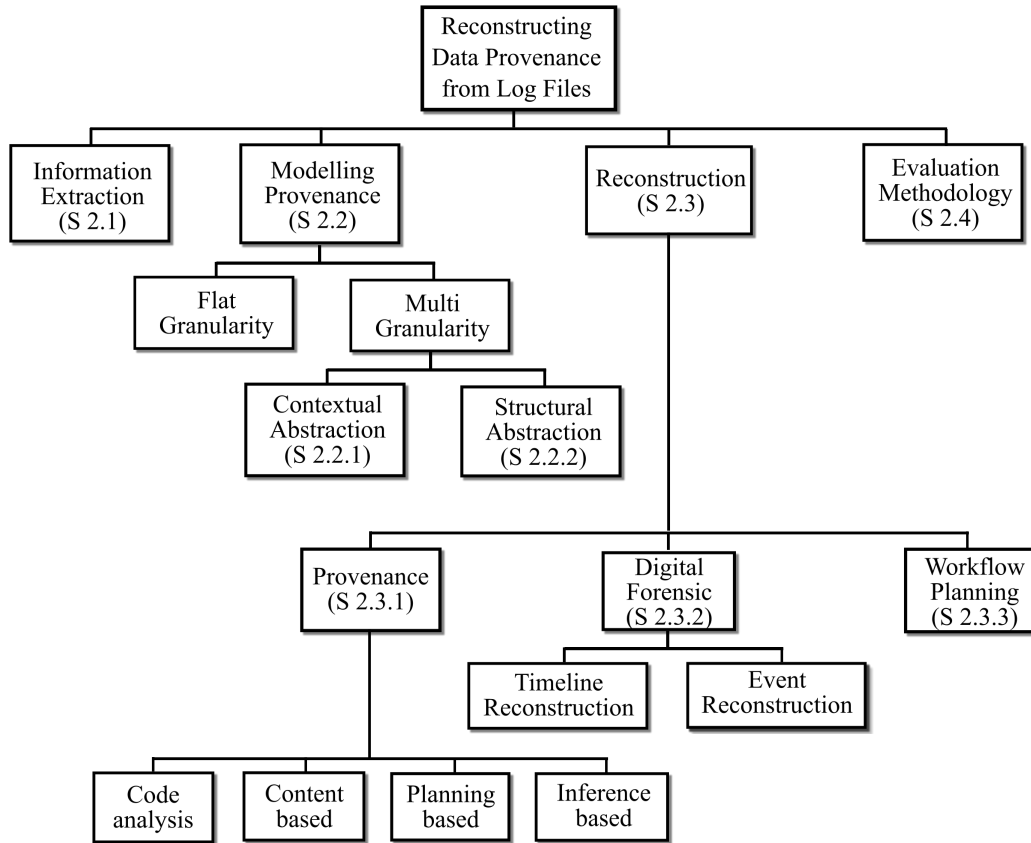


Figure 2.11: Structure illustrating the categories of work reviewed

structuring the data provenance. Existing provenance models can be categorised into those that operate on flat granularity and models where a multi-granularity view is induced using abstraction. However, modelling log files generated from different granularity layers using flat granularity models would result in disparate provenance graphs. We argue that differences between log files, induced by granularity, resemble structural differences in the graph. Hence, multi-granularity provenance models based on structural abstraction are potential candidates for modelling log files generated from different granularities into provenance relations. However, existing models either assume *a priori* knowledge on elements of the provenance or are sensitive to noise or interference in the data. Our proposed provenance model and approach to mapping provenance graphs between different granularity layers is discussed in Chapter 4.

In Section 2.3, research relating to reconstructing the sequence of

events in the areas of provenance, digital forensics and workflow planning are reviewed. Work on reconstructing provenance either references existing provenance or access information such as the application source code or content of data files for deducing the missing provenance. However, this research does not assume provenance is available or collected. Approaches relying on access to application source code or data files may also incur high computational cost due to the number of applications and files that can exist on a computer system. Many timeline and event reconstruction techniques in digital forensics do not focus on reconstructing the relationships among events and entities. While some have proposed using rule-based engines for inferring relationships, users are expected to possess knowledge of either the provenance graph or behaviour of the entities in order to formulate correct rules. However, this research assumes no knowledge on the structure of the provenance graph. In Chapter 5, we present and discuss our proposed algorithm for reconstructing data provenance from the set of modelled provenance relations. The proposed algorithm leverages the causality of events and assumes only basic domain knowledge on how operating systems function.

Although there are published work on provenance reconstruction, discussion on how the approaches are evaluated have been brief. One of the attributing reasons was the lack of suitable datasets for evaluation purposes. We address this gap in Chapter 3 by constructing and making public, a set of datasets for provenance reconstruction research. Amongst work that discusses evaluating their proposed approaches, none have detailed or referenced the methodologies used in the evaluation. In most cases, only the metrics used and the results are presented. Although the metrics used were known metrics such as *precision* and *recall*, we argue that it is still important to detail or reference the approaches used for deriving the values. This is because there can be a different interpretation of the metrics and approaches for computing the values. In Chapter 6, we detail our methodology for evaluating reconstructed provenance and demonstrate its usage by evaluating our solution using the proposed methodology.

Chapter 3

Constructing a Dataset for Provenance Reconstruction Research

A dataset that allows provenance to be reconstructed while possessing the ground truth for evaluation is essential to addressing the thesis question—how complete the reconstructed data provenance is compared the known derivation history of the data. Hence, to systematically identify suitable datasets from a pool of publicly available datasets, a set of requirements is defined. However, our survey revealed a lack of suitable datasets, prompting the Cyber Security Lab to record the New Zealand Cyber Security Challenge (NZCSC'15) dataset. This chapter details the process and setup used to create the dataset. The discussion wraps up with a description of a set of use cases that we derived for evaluation purposes from the NZCSC'15 dataset.

3.1 Requirements for Dataset

Magliacane and Groth [2013] broadly stated the following requirements for a dataset to be considered suitable for evaluating provenance reconstruction research:

- The dataset needs to contain the information that the provenance graph can be reconstructed from.

- The dataset needs to have a gold standard provenance graph which the reconstructed provenance can be compared with.

However, the authors did not elaborate further on the kind of information required for provenance to be reconstructed. Based on the requirements stated by Magliacane and Groth [2013], the following list of requirements are proposed to aid the selection of dataset:

Requirement 1 (R1). *Entities*—the dataset should contain information, independent from the ground truth, that can lead to the identification or inference of at least a partial set of entities involved.

Requirement 2 (R2). *Relations between entities*—the dataset should contain information, independent from the ground truth, that would possibly allow at least partial set of the relationships between entities to be inferred or established.

Requirement 3 (R3). *Ground truth*—the ground truth is a separate portion of the dataset that sheds light on the sequence of events happening during the generation of the dataset. Depending on how the ground truth is collected, it can either be in provenance form (which will be equivalent to the gold standard provenance stated by Magliacane and Groth [2013]) or data from which the gold standard provenance can be derived.

Here, we draw a distinction between the *base data* and the *ground truth data*. The *base data* is the portion of the dataset used for reconstruction. The *ground truth data* is a separate portion of the dataset used mainly for verifying the reconstructed output. The *ground truth* can be generated through manual annotation or captured from a perspective separate from the generation of the *base data*. It should be noted that the *ground truth* is used purely for evaluation purposes and not in the reconstruction.

On top of those listed above, we further extend the requirements for our desired evaluation dataset to include capturing events from multiple perspectives in a computer system. This is in line with our scope of studying data provenance reconstructed from log files generated at different

granularities of a computer system, stated in Section 1.5. We break down the multiple perspective requirement into the following types of events:

Requirement 4 (R4). Multiple Perspective

- *(R4.1) **System events**—events collected from the system layer. These events provide information on process interactions with other elements within a computer system such as files, processes or remote systems. Example of system events are system call traces and network events.*
- *(R4.2) **Application events**—events collected by logging mechanisms operating at the application layer. Application events detail the operational details, state and activities of applications monitored and are usually consolidated into application log files.*

With these requirements in mind, we first look at some publicly available research datasets and evaluate their suitability.

3.2 Evaluating Public Datasets

Following the scope defined in Section 1.5, the survey focuses on datasets describing events happening within a computer system. Table 3.1 summarises the datasets surveyed with respect to the list of requirements presented in Section 3.1.

3.2.1 Provenance Datasets

ProvBench [ProvBench 2012] is the product of an effort to construct a repository of provenance datasets for interoperability testing, validating proposed approaches and benchmarking of solutions by the provenance community. To-date, the repository contains a handful of datasets collected using various provenance collection tools. As a result, most datasets in the repository are in the form of complete provenance graphs modelled using provenance models such as PROV [Moreau and Groth

Table 3.1: Overview of discussed datasets

Dataset	Entities and Relationship (R1 and R2)	Ground Truth (R3)	Multi-Perspectives (R4)	
			System Events (R4.1)	Application Events (R4.2)
ProvBench [ProvBench 2012]	✓			workflow engines
Netresec Dataset [Netresec 2010]	✓		network data	
SNAP Dataset [Leskovec and Krevl 2014]	✓		network data	different application data
ISCX Dataset [Shiravi et al. 2012]	✓	labelled attack data	network data	
ADFA Dataset [Creech and Hu 2013]		labelled attack data	system call traces	
DARPA Dataset [Lincoln Laboratory 1998]	✓	✓	system call traces & network data	

2013; Moreau and Missier 2013]. Since the provenance graphs are already captured, the use of the ProvBench datasets contradicts the objective of reconstructing data provenance.

Since the ProvBench datasets are never intended for provenance reconstruction research, they are not published with ground truth data. As such, the ProvBench datasets do not satisfy requirement R3. Hence, these datasets cannot be used for evaluating the success of a reconstruction.

3.2.2 Network and System Logs

While there are many publicly available network and system datasets, most of them only provide a single perspective (i.e. system events). In the following, we review some of the common datasets used for system security research:

- **Netresec Dataset Collection**—Maintained by Netresec [Netresec 2010], the collection consists of network traffic datasets collected from various cyber security competitions, deployed honeypots and sandboxes for malware analysis. These datasets describe various forms of network based attack and remote intrusion. The objective was to provide data for intrusion detection research. As a result, most of the datasets do not capture events from the application perspective, hence, fail to satisfy requirement *R4.2*. Most of the datasets also do not satisfy requirement *R3* as they do not have ground truth data in the releases.
- **Stanford Large Network Dataset Collection (SNAP Dataset)**—the Stanford SNAP group looks at analysing large social and information networks. Amongst the rich set of social and communication network datasets hosted on the group’s website [Leskovec and Krevl 2014], there are some on computer network communications. These datasets range from monitored email exchanges to peer-to-peer networks. However, like the ProvBench datasets, the datasets hosted here are mostly modelled graph data. Hence they are not relevant for provenance reconstruction research. Having said that, it is interesting to note that the datasets capture a variety of data from different applications such as email clients, Twitter and other social media applications.
- **UNB ISCX Intrusion Detection Evaluation Dataset**—the dataset was generated by the Information Security Centre of Excellence (ISCX) at the University of New Brunswick (UNB) in 2012 [Shiravi et al. 2012]. Agents were created to simulate real world services such as web, mail and file services and were deployed in a testbed

environment. Various kinds of multi-stage attacks against those agents were then carried out and the network traffic was captured as the dataset. By simulating attacks and services in a segregated network, the authors were able to generate a labelled dataset that was free from real-time background traffic (i.e. noise). The labels serve as a form of data from which the ground truth can be derived. However, the dataset only describes events happening on the network. It does not capture application events, hence does not satisfy requirement *R4*.

- **ADFA Intrusion Detection Dataset**—the Australian Defence Force Academy (ADFA) Intrusion Detection Dataset [Creech and Hu 2013] was intended to replace the outdated KDD99 dataset [UIC 1999] released during the Knowledge Discovery and Data Mining (KDD) conference in 1999 for system intrusion detection research. The ADFA dataset consists of system call traces captured off systems that were subjected to simulated malicious attacks. The main advantages of the ADFA dataset are the dataset’s relevancy to modern computer systems (the dataset was generated on recent versions of Linux and Windows operating systems) and that the dataset was labelled, separating the malicious and normal system call traces. Having said that, the arguments associated with the system calls invoked were not captured in the dataset (i.e. the system call traces only contain information on system call types executed by the process). This made it not possible to derive the relationships between elements and as a result, unable to fulfil the entity and relationship requirement (*R1* and *R2*). On top of that, the dataset contains only events from the system perspective and hence does not meet requirement *R4.2*.
- **DARPA Intrusion Detection Dataset**—the Defense Advanced Research Projects Agency (DARPA) Intrusion Detection Dataset is one of the most comprehensive system research datasets, capturing system events such as system call traces and network traffic. It was generated by subjecting agents that simulate real systems to scripted

3.2 Evaluating Public Datasets

attacks. System call traces, along with their arguments, were collected from those systems on a daily basis. The entire simulation was conducted in an isolated environment. A document that describes the attacks carried out, a brief description of each type of scripted attack and their start times was published together with the dataset. The documentation acts as an approximated ground truth data which can be used to verify the results. Unfortunately, the dataset focuses on collecting only system events and neglects events generated by the applications running on the system. Hence, the DARPA dataset does not fulfil requirement *R4.2*. Another downside of the dataset is that it is generated on the niche Solaris operating system. As a result, the dataset is considered to be outdated compared to modern systems.

The ground truth column in Table 3.1 shows even with recent released datasets, ground truth data is usually not included in the releases. Some datasets use labels to annotate and differentiate abnormal events from the normal ones. However, labels do not capture the semantics of the actual incidents (e.g. actual flow of the attack, how it happened or whether two different attacks are related) as the labels only denote whether the respective events were relevant or not. Since a key aspect of provenance is the relationship between entities, it is important that the reconstructed relationships can be evaluated too using the ground truth. As such, the use of labels as the sole ground truth is insufficient for evaluation.

Due to the lack of ground truth data, most of the surveyed datasets could not be used for our research. The documentation that details attacks carried out in the DARPA dataset can be used to derive an approximated ground truth. Based on the description of the attacks and their start times, it is possible to determine which events in the system logs are the relevant attack events. Relationships between entities can then be inferred manually from the relevant events. However, since there is no indication of how long each attack lasted, the ground truth can only be approximated. As stated above, DARPA dataset also does not include application events, making it not possible to evaluate provenance reconstruction using application log files.

Since none of the surveyed datasets can fulfil the listed requirements, we set out to collect and construct our own evaluation dataset from the New Zealand Cyber Security Challenge 2015 (NZCSC'15). The setup for the data collection and the methodology used for the dataset construction is discussed in the following sections.

3.3 New Zealand Cyber Security Challenge 2015

Before describing the NZCSC'15, we would like to clarify the author's contribution with regards to the work done towards constructing the dataset. Although the design and execution of the challenge is critical to the construction of the dataset, it is a group effort together with other members from the University of Waikato Cyber Security Lab. Hence, the design and implementation of the challenge discussed in this section are not part of the contributions claimed by the student for the work done towards constructing the dataset. Instead, the author's contribution surrounds the design and methodology used for capturing and constructing the dataset, specifically, the round 2 portion of the dataset. This section aims to provide the background knowledge for understanding the dataset discussed in Section 3.5.

The NZCSC'15 was designed to raise awareness of cyber security, particularly on web application vulnerabilities and the potential types of attack vectors a vulnerable system is susceptible to. The challenge was structured into two rounds, with round 1 conducted in a capture-the-flag style and round 2 as an attack and defence scenario.

In round 1, participants were required to identify and exploit existing vulnerabilities on a set of hosted web services in order to retrieve a "flag" from the respective systems. Depending on the challenge, the flag could be embedded in source code or within the text of an HTML document. Vulnerabilities introduced were mostly implementation and configuration based, such as lack of user input sanitisation and erroneous permission configuration. This translates to very little file interaction on the system¹.

¹To recap, as stated in Section 1.5, the scope of provenance views data at a file level.

3.3 New Zealand Cyber Security Challenge 2015

As a result, the data gathered from this round was not used and hence, will not be discussed further in this chapter.

Round 2 was designed to test participants' skills on system and web application security. Participants were designated as the defenders and assigned to different "blue" teams. Each blue team was required to maintain the availability of a set of services representative of those found commonly in a corporate environment. The set consisted of a file sharing service, a web server and an email server. Vulnerabilities introduced into the servers were intended to emulate a poorly secured proprietary environment (e.g. a company providing online services). These vulnerabilities include poorly implemented web applications (e.g. unsanitised parameters and exposed management interfaces), outdated applications with vulnerabilities published in the Common Vulnerabilities and Exposure database [MITRE 2017] (e.g. shellshock and php vulnerabilities), weak security and account configurations [van der Stock et al. 2015]. Blue teams had to patch and harden the security of each service so as to prevent the services and their systems from being compromised.

The participants were pitted against volunteers from industry sponsors, who were penetration testers by trade from various security companies, and volunteers from our own research lab. Together, these volunteers form the "red" teams. The red teams' task was to disrupt the services maintained by the blue teams.

Each blue team was scored based on the availability of their services by a scoring machine. Availability was determined using a script that periodically checked if the respective services of each team could be reached from the scoring machine. Points were then awarded to the respective team based on the services reached successfully. Figure 3.1 shows the schematic for the setup used in Round 2.

All system were hosted on Virtual Machine (VM)s running Linux Ubuntu 15.04 as the operating system. The entire virtual environment was hosted using OpenStack [The OpenStack Foundation 2010], a private cloud virtual infrastructure management suite. The OpenStack version used was the *Havana* release. Each blue team was assigned three VMs, each run-

Hence, emphasis is placed on file and system interactions.

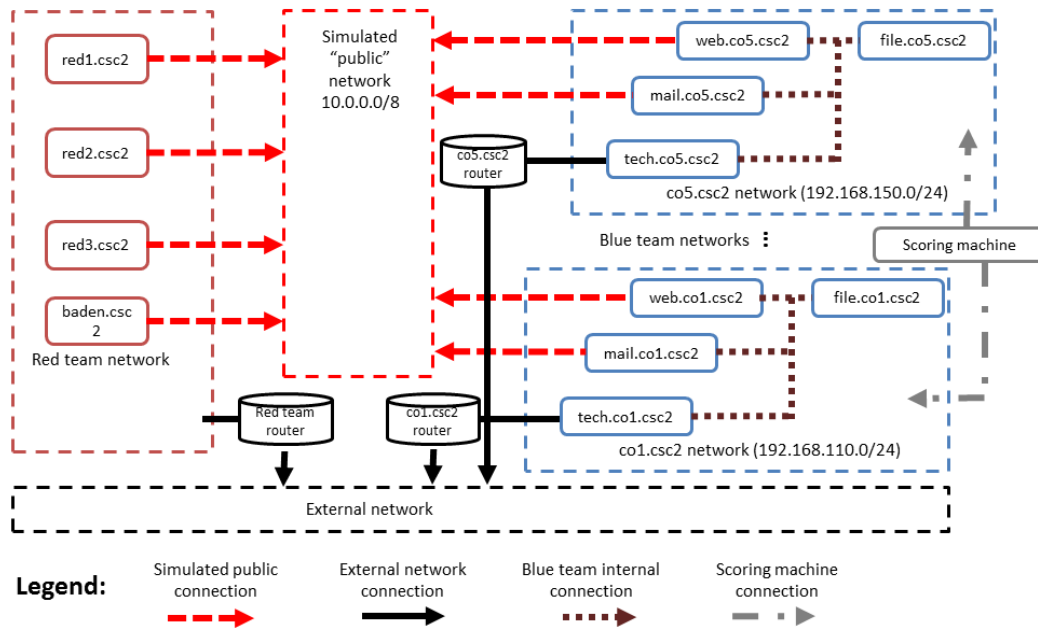


Figure 3.1: Schematics of the setup used for Round 2 of NZCSC'15

ning a service; a *web server*, a *file server* and an *email server*. VMs for each team were placed in a virtual network allocated to each team so as to prevent interference between the blue teams. Each blue team was also allocated a *Technician* VM for tunnelling into their allocated virtual network from the physical network. Participants would then access the VMs hosting the services through the tunnel.

Blue team networks are shown as the blue boxes in Figure 3.1. The red team systems were placed in a separate red team virtual network. This was to simulate attackers from outside of a proprietary network (e.g. the Internet). Each red team VM was pre-installed with Metasploit [Rapid7 2016], a penetration testing suite, as the basic tool. A list of attacks observed to have been carried out by the red teams on the blue teams² is attached as reference under Appendix A.

²This is done by going through the video recordings captured on both the blue and red teams. Details on the video recording is discussed in Sections 3.4 and 3.5.

3.4 Setup for Data Collection

As the dataset to be constructed is intended for evaluation, events in the dataset should be generated in a realistic manner, as opposed to a simulated dataset where events that happened are planned. Evaluating against such a realistic dataset would allow us to truly understand whether the proposed reconstruction solution works.

The NZCSC'15 provided a platform for capturing data that are close to real-world data. Although the vulnerabilities introduced in the challenge were specifically selected, there was no control on the attack vectors used (e.g. types of exploit used) or the breath of the attacks (e.g. activities executed after a successful exploitation). This introduces a sense of uncertainty on the impact and effect of each attack, even if two attacks are exploited using the same vulnerability. Potential limitations of the dataset will be discussed in Section 3.7.

To construct the *base data*, both system and network events were collected from both blue and red team VMs. From the system perspective, events were collected from both the system and application layers. At the system layer, kernel logging mechanisms Sysdig [Sysdig 2014] and LAF [archLinux 2012] were used for collecting system call traces from the kernels. Both the system calls invoked and their associated arguments were logged. This was done to fulfil requirements *R1* and *R2*, which deals with being able to infer the entities and relationships involved.

The motivation behind using different tools for logging events was so that interoperability between different data formats could be tested (i.e. that a proposed solution is able to digest different log formats). On the application layer, where possible, application logs produced by the various services were collected at the end of the competition. From the network perspective, Daemonlogger [Roesch 2006] was used to monitor and log both incoming and outgoing network traffic on each VM. By collecting events from the system, network and applications running, requirement *R4* is satisfied. Table 3.2 lists the logging mechanisms used and the types of log file captured.

The main challenge with constructing a realistic dataset that satisfies

Table 3.2: Summary of tools used and logs collected on each VM type

VM Type	System logs	Application logs	Network logs	Screen capture
Blue-Web server	Sysdig	Apache logs	Daemonlogger	-
Blue-File server	LAF	Samba logs	Daemonlogger	-
Blue-Technician VM	-	-	-	VLC
Red VM	Sysdig	-	Daemonlogger	-

the requirements listed in Section 3.1 is generating the *ground truth data* without affecting the *base data*. In simulated datasets such as the DARPA dataset, events are planned (e.g. simulated attacks made on the systems). As such, the authors of the datasets are aware of details concerning those events. Ground truth can then be produced by documenting those details. However, with constructing a non-simulated dataset, authors should have no influence or role over events that happened. Hence, alternative means, independent of the those used for capturing the *base data*, is required to generate the ground truth.

For the data collection exercise, *ground truth data* is generated using VideoLan Client (VLC), screen recording software. Screen activities of participants from both teams are captured as video recordings from each team’s physical terminals using the software. Since no data logging was done on the physical terminals, such a setup ensured that the video recording did not interfere with the data collection. Also, since control of the VMs was done through the physical terminals, recording the activities from the physical terminals ensured that all activities for each team were captured. Figure 3.2 illustrates a simple overview showing the separation of the data logging and the video recording.

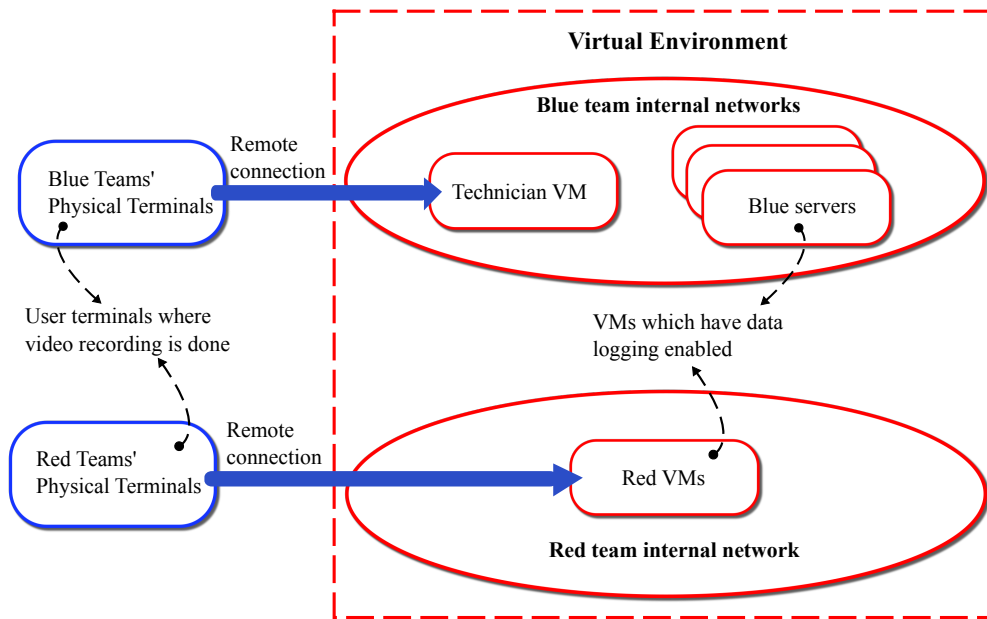


Figure 3.2: Environment separation between data logging and video capture

3.5 NZCSC'15 Dataset

Overview of the NZCSC'15 dataset

In total, data was gathered from five blue teams and three red teams (two industry experts and the backdoor VM). Permission to record screen activities was obtained from all members of the teams for ethics approval. Unfortunately, one of the blue team's recording was found to be corrupted after the competition. Hence only recordings for four blue teams and the two red team experts were retrieved successfully. We also did not manage to obtain a set of usable log files from the file server application due to a misconfiguration in the logging mechanism. A breakdown of the data collected is as follows:

Blue Team:

- Network traffic log – 5 sets
- System call trace log on web server – 5 sets (Sysdig)
- System call trace log on file server – 5 sets (LAF)
- Apache web server log – 5 sets

Red Team:

- Network traffic log – 3 sets
- System call trace log – 3 sets (Sysdig)

Video Recording:

- Blue team – 4 sets (Blue 2-5)
- Red team – 2 sets (Red-1 and Red-2)

In the following subsections, we discuss the format and process used for transcribing the video recordings into ground truth data.

Format of transcript

To enable the video recordings to be machine-processable, each recording was transcribed into text form. Each observed event in the video was transcribed as an entry in the resulting transcript. An example of an entry in the transcript is shown in Figure 3.3. Each entry in the transcript is broken down into seven data fields. This is to allow for a systematic approach to transcribing the video and to help preserve the semantics of the events observed in the video. The data fields used are as follow:

1. Time - the *time* field records the observed system timestamp when the event first took place on the system. Because the timestamp observed was obtained by monitoring the desktop clock, we were only able to capture the timestamp to minute granularity.
2. Action - the *action* field describes the observed event. Values of this field are structured to first contain a short description of the command or action observed, followed by the VM the event was carried

Time	Actions	Local Command	Send	Receive	Comments	Intent
20:21	ssh to file.co2.csc2 from tech.co2.csc2 as tech user on terminal	ssh tech@file.co2.csc2	request to authenticate password of tech user to tech.co2.csc2	credentials of tech user from tech.co2.csc2	Authenticate successfully and established connection with file.co2.csc2	

Figure 3.3: An example of an entry in the transcript

out on (i.e. location), the user account that executed the action and finally the means through which the action was executed (e.g. command line, web browser). To simplify the transcription process, location is not transcribed if the event happened on the VM to which the transcript belongs. Similarly, means of execution is not recorded if the action was executed through the command line interface.

3. Local Command - the *local command* field records the observed command and arguments used by the user in carrying out the observed event. This field is only valid for events happening on the command line interface.
4. Send - the *send* field provides a general description of the data sent to the remote host on the network due to the observed event. It details the nature of the data sent and to which VM the data is being sent.
5. Receive - similar to *send*, the *receive* field details the nature of the data received at the local host (the VM which the transcript is for) on the network and the VM from which the data was sent.
6. Comments - recording the semantics of the observed events into fixed fields is not an easy task due to the wide variation of visual feedback possible and domain knowledge required to understand the observed events. Hence, the *comments* field is a free-text field that records either a description of the feedback from the system or comments that will provide insights into what is happening in the observed event.
7. Intent - in some situations, it is not easy to understand what a user is trying to achieve through just one event. As such, the *intent* field is used for annotating what the participant is attempting to achieve over one or more observed events. The information provided in the *intent* field is derived through one or a combination of two methods. The first method was through observing what happened over the series of events. The second method was by looking at the Google search queries carried out on the browser by the participant when

attempting to perform tasks related to the observed event. As such, the information provided in the *intent* field is a best effort attempt. Having said that, we believe that annotating the perceived intent of the participant can facilitate reasoning about and understanding of the ground truth data. This field is left empty if the comments or event description is clear in expressing the intention of the participants.

The next subsection describes the process used for breaking down each event observed in the video recording into the defined fields.

Video transcribing process

Events observed in the videos can be generated either by the user or the system. An event is considered user generated if it is observed that the event is the result of a direct action by a participant (i.e. a participant executed a command or action). The time of execution, the command used and the data sent are noted for each observed user generated event. This information, along with a brief description of the action or command executed are transcribed into the transcript using the format described above. The system is then observed for feedback returned to the user in relation to the command or action executed. Such feedback may come as a message prompt returned by the system or as a visual effect caused by the participants' action, such as the successful loading of a web page. Feedback observed is transcribed into either the *comments* or *receive* fields or both. In the case where a goal is achieved through a series of events, the *intent* field for the respective entries will be annotated with the same goal.

On the other hand, an event is considered system generated if the observed event happened suddenly or automatically without the participant's involvement. An example of a system generated event could be a sudden disconnection of the participant's remote terminal to one of the servers, observed in video recording. In such cases, the time at which the event is observed and a description of what had happened are noted

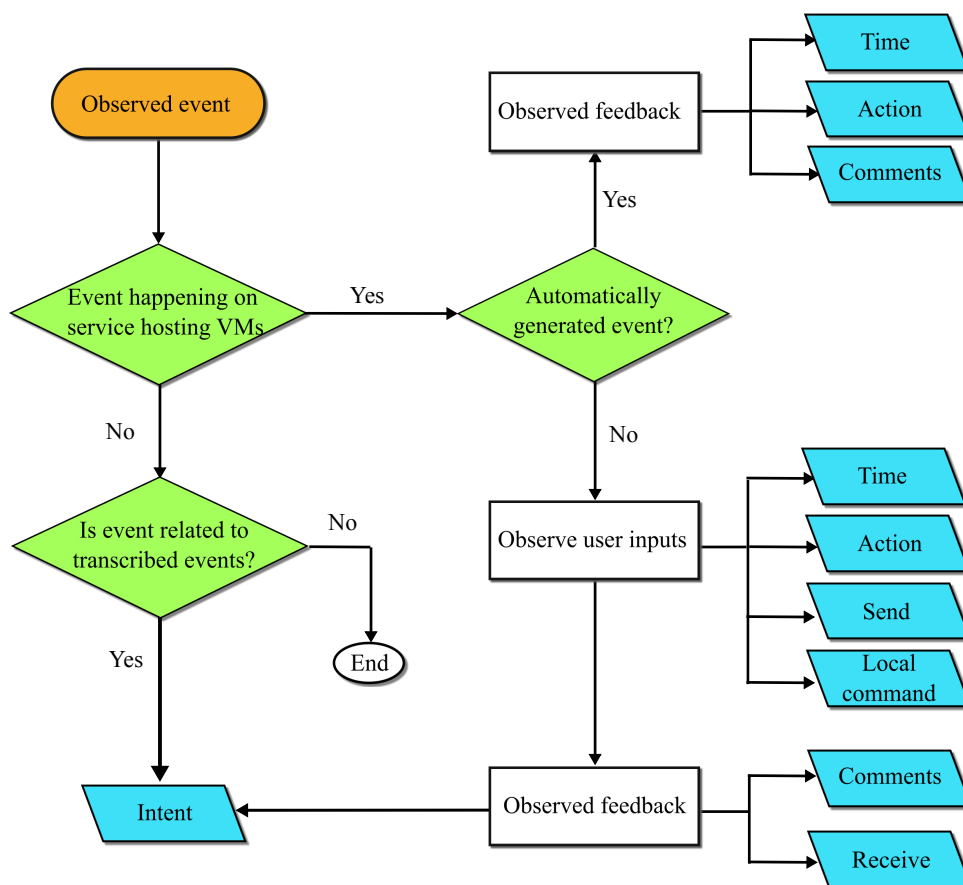


Figure 3.4: Flowchart depicting the video transcribing process for system and user generated events

in the respective fields. If required, further details are noted in the *comments* field. It should be noted that a system generated event could be caused by user generated events from other teams (i.e. attacks from red teams causing immediate visual feedback on blue team terminals). In those cases, the responsible event is transcribed as a user-generated event in the other team’s transcript. As such, it is possible to correlate two transcripts and deduce the cause and effect of actions or commands executed by other teams. However, we do not annotate such correlation into the ground truth so as to avoid complicating the ground truth data. The transcription process is illustrated as a flowchart in Figure 3.4.

Two examples of the resulting video transcripts are shown in Figure 3.5. Figure 3.5a shows the result of transcribing an observed standalone event. Information in the *action* field is decomposed using boxes and

Chapter 3 Constructing a Dataset for Provenance Reconstruction Research

Time	Actions	Local Command	Send	Receive	Comments	Intent
20:21	ssh to file.co2.csc2 ¹ from tech.co2.csc2 ² as tech user ³ terminal ⁴	ssh tech@file.co2.csc2	request to authenticate password of tech user to tech.co2.csc2	credentials of tech user from tech.co2.csc2	Authenticate successfully and established connection with file.co2.csc2	

(a) A single transcribed event. The action field is highlighted with different colours and annotated with numbers to illustrate formatting used

Time	Actions	Local Command	Send	Receive	Comments	Intent
21:56	edit crontab as root	sudo crontab -e			Crontab opened using /bin/nano.	Participant removes all cronjobs for root user.
21:56	close editor editing crontab	ctrl+x			Modified crontab to remove all cronjobs. Modifications saved to crontab.	
21:56	edit crontab as root	sudo crontab -e			Crontab opened using /bin/nano.	
21:56	close editor editing crontab	ctrl+x			No modifications made to crontab.	
21:56	list cronjobs for root user as root	sudo crontab -u root -l			no cronjobs listed.	

(b) Intent field used to annotate relation and intention behind executing a series of observed events

Figure 3.5: Examples of the video transcript of the ground truth

numbering to demonstrate the format described in the field descriptions above. The intention behind having a format for the fields is to allow the transcript to be processed in an automatic and structured manner. Figure 3.5b shows how the *intent* field is used to semantically group a series of events together. Although the *intent* field is in free text form, it can be used to conveniently group and label events into event groups.

After transcribing the video recordings, the NZCSC’15 dataset would have event log files and the transcripts describing the participants’ activities. These two sets of data represent the base data and the ground truth data, discussed in Section 3.1, respectively. The event log files will be used for data provenance reconstruction, while the transcripts will be used for deriving use cases for evaluation. Next, we describe the use cases derived.

3.6 Use Cases from the NZCSC'15 Dataset

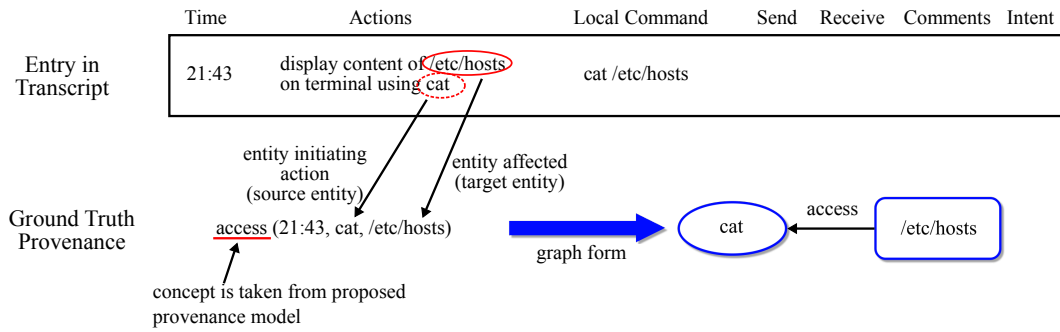


Figure 3.6: Converting event in transcript to ground truth provenance

3.6 Use Cases from the NZCSC'15 Dataset

Using the transcript, a sample set of use cases was derived for the purpose of evaluating the proposed reconstruction solution. For each use case, a piece of data (e.g. file) is first selected. Events in the transcripts that are relevant to the file (e.g. events that modify, access or propagate data in the file) are then identified. Information such as timestamp, the actions and involved entities are then extracted and expressed as a provenance graph using concepts described in our proposed provenance model, discussed later in Chapter 4. Figure 3.6 shows a simple example of an event in the transcript being converted to ground truth provenance. From the *actions* field of the entry in the transcript, one can deduce that the application *cat* is used to read data from the file */etc/hosts*. Using concepts in the proposed provenance model, the entry is translated to a tuple format that shows the relationship between the two entities. The format of the tuple is discussed along with concepts in the model in Chapter 4. The tuple can also be converted into graph form for visualisation purposes, as shown in the figure. The resulting provenance graph for each use case is the ground truth provenance with which the reconstructed data provenance can be compared for evaluation.

The derived use cases and selection of files are designed to test the reconstruction solution on the following capabilities:

1. ability to retrieve all relevant entities and relationships
2. reconstruct the sequence that depicts how the data is derived from

its origin

3. reconstruct the sequence that shows how the data evolve from a given state

In the following, a brief description of the scenario and the simplified ground truth provenance³ are presented for the use cases derived.

Use-case 1—Backing up Critical Files

Scenario - Participants from the blue team, *web-co2*, were attempting to backup a set of critical files which they were supposed to protect from the red team. First, the blue team checked the content of the file */home/tech/PrivateFiles/PrivateFile1* by reading it using the application *less* on the terminal. After ensuring that the file was correct, they copied it from the file server to the technician VM using the *scp* application. A checksum was then generated for the file using the *md5sum* application.

Based on Pechanec [2007]’s explanation of how *scp* utilises the underlying secure shell (*ssh*) daemon, *sshd*, for sending data to remote hosts, we added the passing of data from the *scp* to a *sshd* process into the provenance graph. The resulting high level provenance graph is shown in Figure 3.7. The dotted arrow represents host-to-host network communication.

This simple use case covers not only day to day user interactions with files but also communication between processes. Process-to-process com-

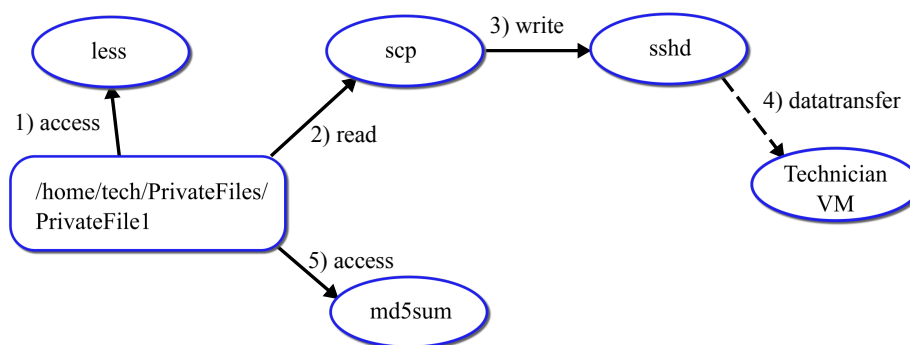


Figure 3.7: Use-case 1—Backing up */home/tech/PrivateFiles/PrivateFile1* to remote host, Technician VM

³In some cases, certain events were repeatedly executed over time. For simplicity, these repeated sequences are not shown in the figures.

3.6 Use Cases from the NZCSC'15 Dataset

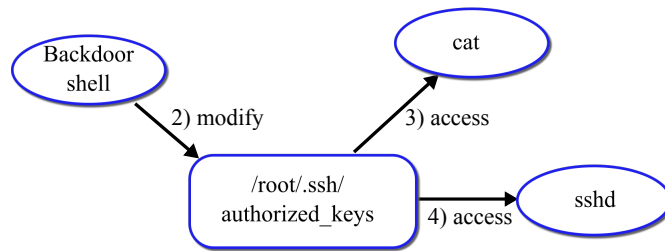


Figure 3.8: Use-case 2—Modification of `/root/.ssh/authorized_keys` file in order to gain remote shell access through ssh

munication is an important behaviour in the operating system, used mainly to move data from files to other locations such as other files or hosts. By selecting the file `/home/tech/PrivateFiles/PrivateFile1` as the data whose provenance is to be reconstructed, this use case tests whether the propagation of data across different processes and over the network can be reconstructed.

Use-case 2—Malicious File Modification

Scenario - One of the first things the red teams did upon successfully compromising the blue teams' VMs was to enable password-less remote shell access for themselves. To achieve that, the red team injected their own ssh-key into the `/root/.ssh/authorized_keys` file on the blue team's VM. Modification of the file was done via a malicious backdoor shell installed previously. The red team then checked to see if the ssh-key had been successfully injected by reading and displaying the content of the file using the `cat` application. Every time the red team logs in remotely to the VM using ssh, the file `authorized_keys` will be automatically accessed by the ssh daemon (i.e. `sshd`).

The high-level provenance graph for use-case 2 is shown in Figure 3.8. Use-case 2 is an example of how hidden malicious software can alter the system, eventually leading to the system being compromised. It tests if the modification of `/root/.ssh/authorized_keys` and events showing different applications consuming the modified data over time can be reconstructed.

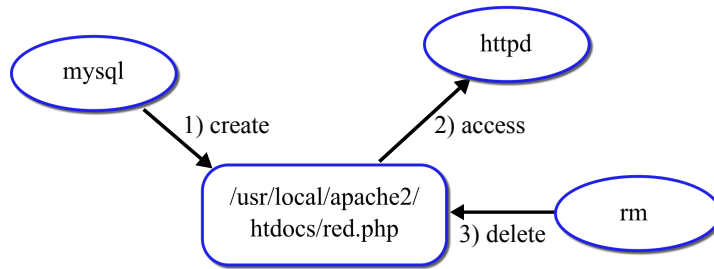


Figure 3.9: Use-case 3—Injecting Malicious Payload via Database

Use-case 3—Malicious Payload

Scenario - Use-case 3 captures the red team injecting malicious shell code into one of the blue team’s VMs via the MySQL database, *mysqld*. Access to the blue team’s web server (*httpd*) is gained by leveraging the malicious shell code and the method depicted in use-case 2. Finally, the shell code is deleted so as to cover the attackers’ tracks.

Figure 3.9 shows the high-level provenance graph of use-case 3. Differing from the previous use cases, use-case 3 captures the entire lifecycle of the file *red.php*; from its creation till its deletion. Creation and deletion of a file are important relationships in data provenance as these relations mark the beginning and end of a file’s life cycle. As such, the proposed reconstruction solutions should not only be able to reconstruct the interactions but also the creation and deletion of files.

Use-case 4—File Versions

Scenario - In this use case, the red team first injected a malicious shell code, *red0.php* into one of the blue team’s VM web folders */usr/local/apache2/htdocs/*. To ensure availability of the shell code on the compromised VM, the red team made backups of the shell code in various directories. This was achieved by copying the file to other directories using the *cp* application. Two different backups were created, the first one from the original file to another web server directory and the other to the system temporary file folder.

Figure 3.10 shows the ground truth provenance for file, *../test/red0.php*.

3.6 Use Cases from the NZCSC'15 Dataset

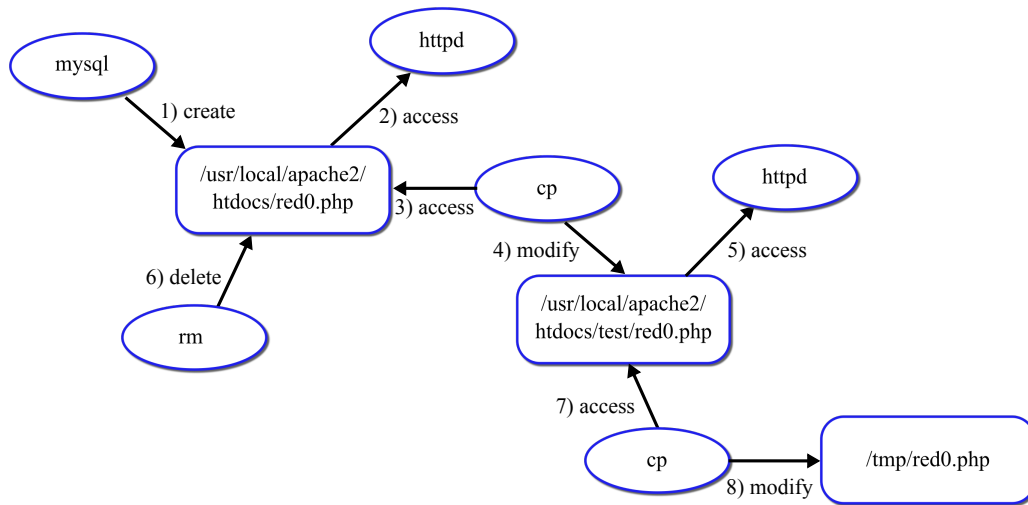


Figure 3.10: Use-case 4—Data provenance showing the derivation of the file */usr/local/apache2/htdocs/red0.php*

The file is selected intentionally to test whether the relationships that show how the file is derived from *../htdocs/red0.php* and how it evolves to */tmp/red0.php* can be reconstructed.

These four use cases are examples of the type of use cases that can be extracted from the ground truth for testing provenance reconstruction solutions. These use cases demonstrate some scenarios of file tampering and file evolution during the competition⁴. However, the use cases do not exhaustively represent the different variety of attacks that can be observed in the real world, such as those encountered in a digital forensics investigation. This is due to the limitation in the scope of the dataset, which will be discussed in Section 3.7.

Having said that, the use cases are sufficient for representing the basic operations that can be carried out on a file (e.g. create, read, write, delete) and the communication of data between entities. By focusing on reconstructing these aspects, we can verify if the proposed reconstruction solution can produce data provenance that allow questions such as, “how was the file created?” and “where did the data in this file originate from?” to be addressed. Such a reconstructed output is relevant to

⁴Other use cases that can be derived from the ground truth can be inferred from the list of attacks, listed in Appendix A, carried out in the challenge.

Table 3.3: Overview of the set of use cases

	File create	File read	File write	File delete	Derivation across entities
Use-case 1		✓	✓		✓
Use-case 2		✓	✓		
Use-case 3	✓	✓		✓	
Use-case 4	✓	✓	✓	✓	✓

the data provenance we defined in Section 1.5 of Chapter 1⁵. Table 3.3 provides an overview of the aspects captured in each use case.

3.7 Limitations of the NZCSC'15 Dataset

Although the NZCSC'15 dataset was captured from a live platform, we do not assume that the dataset is complete and perfect. In this section, we discuss limitations of the dataset that stems from how it was captured and constructed.

Enclosed Virtual Environment

Due to constraints regarding the venue of the competition, it was decided that the competition be hosted in a closed environment (e.g. a virtual environment separated from external networks). This decision has an effect on the amount of non-malicious activities captured and represented in the dataset.

Although non-malicious activities are still being generated throughout the competition (e.g. participants testing and checking their system functionality; events generated by scoring machine; execution of background services), the amount of non-malicious activities captured is still limited compared to a real-world environment. This may affect the fuzziness of

⁵We defined data provenance as, “the information that depicts the evolution process of a piece of data, including the entities and activities involved in the process”

3.7 Limitations of the NZCSC'15 Dataset

the dataset required for evaluating security research such as intrusion detection.

Another limitation that resulted from the decision to host the competition in a virtual environment is the tools that can be used for collecting the ground truth. Participants interacted (e.g. keyboard typing and mouse clicks) with the virtual environment from the physical environment where installing monitoring software such as key loggers was not possible. As a result, more precise ground truth could not be captured and we had to rely on screen recording for generating the ground truth. We intend to address this limitation through revisiting the design of the architecture used for the competition in future efforts in dataset collection.

Focus of the Challenge

The objective of the NZCSC'15 challenge was to educate the participants on the impact of poorly implemented and secured web applications. Hence, vulnerabilities introduced into the challenge were targeted at web applications and system perimeter defense⁶. The variety of attacks (e.g. types of payload and exploits used) executed and captured were also constrained by the duration of the competition. As a result, the data captured may not be rich enough to represent the wide range of types of exploits and payloads seen in the real-world. These two factors may constraint the types of use cases that can be generated for research purposes.

Having said, based on our observation on the attacks carried out, events captured in the dataset does cover the different phases of a system intrusion (e.g. reconnaissance, vulnerability scanning, vulnerability exploitation, malware insertion and covering up) [Wai 2002; Lois 2015]. Tampering of user files and system configurations are also included in the types of payload observed to be executed. These factors allow for the generation of use cases suited for research that emphasis on root cause

⁶For more information on the type of vulnerabilities introduced, reader can refer to the attack labels listed in the Table in Appendix A.

analysis, such as system or data security using a provenance approach.

Structure of the Ground Truth

One of the initial goals set for the NZCSC'15 dataset was to cater to research work in the areas of provenance and security. However, the format for the ground truth required for the different types of research work may differ. For example, provenance research often require the ground truth to be in the form of graphs, usually modelled using defined provenance models [Magliacane and Groth 2013; Nies et al. 2015]. In contrast, research that explores the use of machine learning approaches, such as classifiers [Nari and Ghorbani 2013] and neural networks [Khan et al. 2007] for detecting malicious and abnormal events relies on labelled ground truth.

Hence, instead of providing the ground truth in multiple formats, we opted to represent the ground truth in a structured document. This is such that it can be automatically processed into the desired ground truth format for experimentation use. However, this adds another processing step of converting the transcript into a format that can be readily consumed.

3.8 Summary

A list of requirements for datasets that can be used for evaluating provenance reconstruction was proposed and discussed in the beginning of this chapter. Using the list, we showed how publicly available datasets used commonly for system security and other system related research are not suited for evaluation of our research. As such, the NZCSC'15 dataset was constructed.

The dataset was constructed through a data collection exercise done in conjunction with the NZCSC'15 competition. Events happening on the VMs used were collected as the *base data* for reconstruction. Video recordings of screen activities of participants were transcribed into text

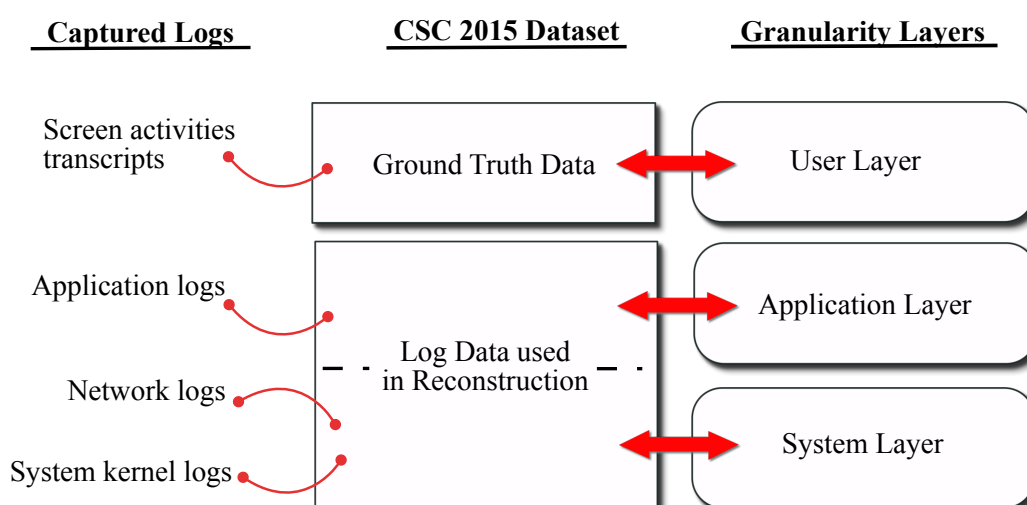


Figure 3.11: Mapping between the log files and transcript in NZCSC’15 dataset to granularity level of computer system

form to act as the *ground truth data* from which use cases and their respective ground truth provenance can be derived for evaluation purposes. The use cases, shown in Section 3.6, demonstrate the types of use cases that can possibly be derived from the transcripts for experimentation. The data gathered in the NZCSC’15 dataset is summarised in Figure 3.11. Limitations caused by how the data was collected and constructed were also discussed in Section 3.7.

The NZCSC’15 dataset contributes towards the effort by the Cyber Security Lab on constructing a comprehensive repository of datasets for research use. At the point of writing this thesis, the NZCSC’15 dataset is being used internally by other researchers of the Cyber Security Lab. Requests for the dataset (e.g. for examination purposes) can be made to the lab at info@crow.org.nz. Together with the NZCSC’16 and NZCSC’17 datasets, this dataset will be published for public consumption by 2018.

The next step of the reconstruction workflow, shown in Section 1.7, is to extract and model information from the log files, such that it can be used for reconstructing the data provenance. In the next chapter, the approach used for the extraction and the model proposed for modelling of provenance relationships is discussed.

Chapter 4

Modelling Log Events as Provenance Relations

This chapter discusses how information is extracted from log files and modelled into provenance relations, to be used for reconstructing the provenance. The approach that is adopted for information extraction is discussed. To address the need to model log events from a multi-granularity perspective, the Data Flow Provenance Model (DFPM), a multi-layered provenance model for modelling log events into provenance relations, is proposed. A set of patterns that defines how provenance relations can be mapped between different layers in the model is then presented. Together with the defined patterns, the proposed model allows data provenance, reconstructed from fine-grained log files, to be compared with the ground truth provenance, derived from events observed at the user layer.

4.1 Modelling Log Events to Provenance Relations

One of the challenges of extracting information from log files is dealing with heterogeneity in the logs (e.g. in log format and parameters). However, as pointed out in Section 2.1, proposing a generic approach for extracting information from different types of log file is not the focus of this thesis. Instead, an adaptor-based architecture is adopted for processing different types of log file, as illustrated in Figure 4.1. This architecture

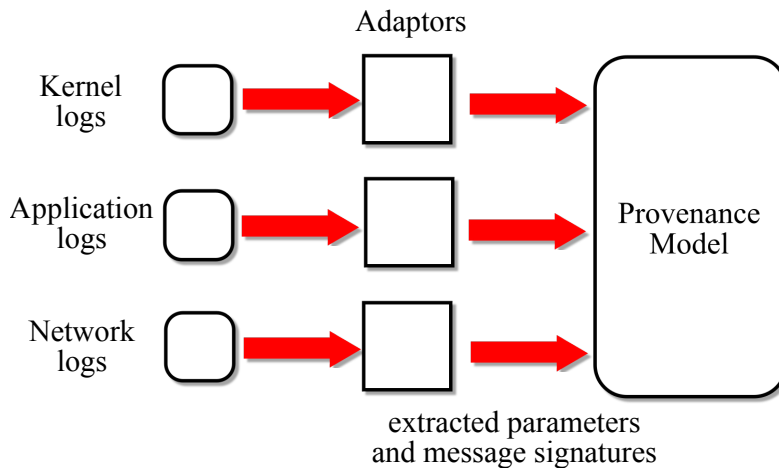


Figure 4.1: Illustration of the adopted adaptor-based architecture for processing log files

has been used in state-of-the-art analytic tools such as ArcSight [Bambenek 2007] and Splunk [Splunk Inc 2005]. For each type of log file, an adaptor is designed for extracting parameters and message signatures using regular expression and string parsing techniques. Contextual information such as timestamps and results returned from the events are also extracted for correlation purposes.

Knowledge such as the structural format of the arguments and the types of encoding used to encode the arguments are applied in the respective adaptors for processing the extracted parameters. By incorporating such knowledge into the respective adaptors, the heterogeneity in parameters caused by variations in the implementations of logging mechanisms can be managed.

Information extracted from each log entry is modelled into pair-wise provenance relations using the model described later (Section 4.3). Each pair-wise provenance relation shows how data is propagated or the relationship between two entities. These provenance relations form the basic building blocks that will be used by the reconstruction algorithm, discussed later in Chapter 5, for reconstructing the data provenance.

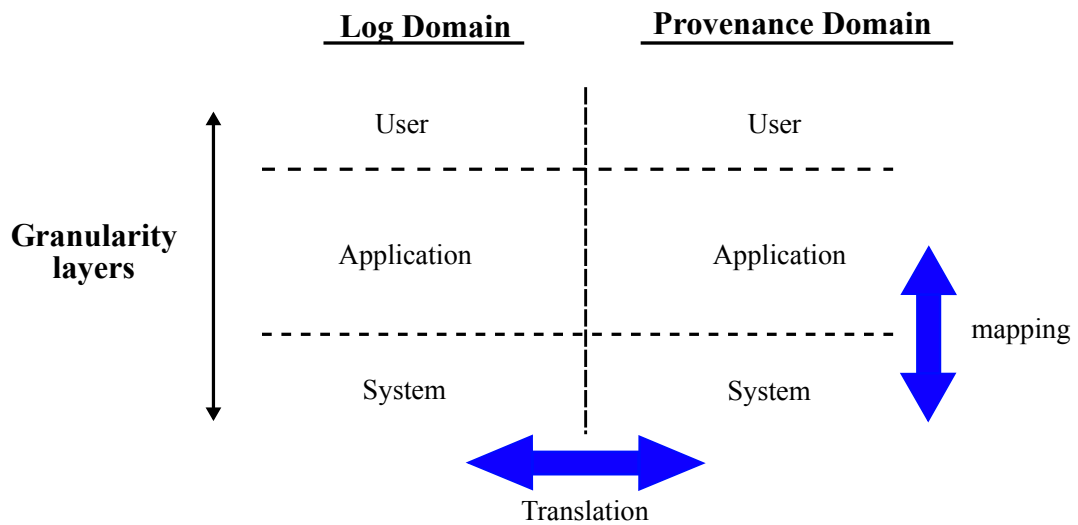


Figure 4.2: Illustration of granularity layers used in the log and provenance domain

4.2 Assumptions Made

Before presenting the proposed model, the problem and assumptions used are first discussed. In Section 2.2, the issue of modelling log entries generated at different granularity layers into provenance was highlighted. Due to differences in the level of detail, modelling log entries of different granularity using a flat granularity provenance model would result in disparities in the resulting graphs. To resolve this, a multi-layered provenance model is required. Hence, the first assumption is made:

Assumption 1. *The provenance domain can be divided into the same set of granularity layers used to describe the log domain.*

The granularity layers used were discussed in Section 1.5 and are illustrated here in Figure 4.2. Based on Assumption 1, the following two problems are identified:

1. How can relationships be modelled from the log domain into the provenance domain?
2. How can provenance relations be mapped between granularity layers in the provenance domain?

In the context of modelling, the term *translation* is used to describe the modelling of information (e.g. entities and relationships) between the two domains. Likewise, the term *mapping* is used to describe the abstraction of provenance relations between granularity layers.

With respect to the first problem identified, learning the translation between the log and provenance domain would require extensively labelled datasets from both domains. While log files can be scraped from running computer systems, obtaining the corresponding provenance datasets is challenging. For example, provenance datasets discussed in Chapter 3, such as the ProvBench datasets [ProvBench 2012], do not have the corresponding log files packaged in the releases. Existing provenance collectors, such as SPADE [Gehani and Tariq 2012], assume the translation using their understanding of the events being monitored (e.g. domain knowledge). Based on this, the second assumption made follows:

Assumption 2. *Knowledge about translating relationships from the log domain to the respective layers in the provenance domain can be inferred from the nature of the activities described by log events (e.g. reading and writing of data).*

The rest of this chapter focuses on addressing the second problem identified. Concepts defined in the model proposed in Section 4.3 allow relationships to be represented in the provenance domain. The patterns defined in Section 4.4 address how provenance relations can be mapped to other granularity layers.

4.3 Data Flow Provenance Model

Data Flow Provenance Model (DFPM) is proposed for addressing the need for a multi-layered provenance model for modelling log events into provenance relations. It is based on the Open Provenance Model (OPM) [Moreau et al. 2011], a model proposed for modelling provenance. The model has been used in the past for modelling provenance in computer systems [Gehani and Tariq 2012; Groth and Moreau 2011]. While OPM models relationships between entities from an ancestry perspective (e.g.

derivedFrom, wasGeneratedBy), DFPM models how data flows between entities. This results in a difference in the vocabulary used, even though the format and fields used by both models are similar. Having said that, the main difference between the two models is the treatment of the granularity of provenance. OPM treats all provenance relations to be of the same granularity, while in DFPM provenance relations are modelled into different granularity layers.

The motivation for modelling data flow between entities is in relation to our objective to reconstruct data provenance that depicts the evolution of a piece of data. A process can create multiple files, but only writes the relevant data to one of those files. Based on our definition of data provenance in Chapter 1, the reconstructed data provenance should capture only the file with the relevant data. Hence by modelling the data flow, the reconstruction can directly identify which file or entities have interacted with the data.

Concepts defined in DFPM are also heavily influenced by the treatment of data in this thesis. To recap the research scope defined in Section 1.5, a piece of data is viewed at a file level. Although by no means exhaustive, the concepts are intended for modelling the basic interactions (e.g. create, read, write, delete) [Silberschatz et al. 2012] and the types of entity that can interact with files in the context of a computer system.

DFPM consists of two classes - *Entity* and *Activity*. *Entity* associates to beings or objects that exist either in the real world or digitally. Examples of entity can range from user accounts to processes or remote hosts in a computer network. On the other hand, *Activity* models the propagation of data and the management of channels required for data to flow between entities.

4.3.1 Entity

The *Entity* class encapsulates two concepts - *Agents* and *Objects*. Collectively, these two concepts are termed as “entities”. They are defined as follows:

- **Agents**—Agents refer to entities that are capable of initiating or

being held responsible for an activity observed with another entity. Examples of agents are processes, users or even network hosts.

- **Objects**—Objects refer to entities that are understood to be incapable of initiating any form of activity on their own. Objects are often viewed as data sources or data sinks. Objects can range from file descriptors used by the kernel for data communication to files within a file system.

Each entity can be expressed into a provenance record¹ that captures information regarding the entity. The format used for expressing each entity is as follow:

Entity_type(*ID*, [*owner*], [*context1*;*context2*;...])

Entity_type is used to record whether this entity is of *Agent* or *Object* type. *ID* captures the label used by the respective logging mechanisms for uniquely identifying the entity². *Owner* is used to capture information regarding the user who created or owned this entity. Lastly, *context* captures any contextual information that can provide insights into the identity of the entity. Such information can either be used to distinguish two entities with the same *ID*³ or for correlating between entity instances across granular layers⁴. Multiple pieces of contextual information in the *context* field are separated using semi-colons.

Classification of entities is based on the expected behaviour of the entity. However, in situations such as a malicious attack, it is possible to have entities with behaviours overlapping that of an agent and object.

¹A provenance record is an entry in a provenance log. Its relation to a provenance log is akin to a log entry's relation to a log file.

²Note that processing of events are done independently at log file level. As such, labels for the same logical entity may differ across different logging mechanisms.

³In cases such as kernel logs from two different systems, entities with the same *ID* may exist in both logs. However, both entities are essentially different as they originate from different systems. As such, contextual information such as the identity of the host can be used to differentiate them.

⁴A process can be identified by its binary name at the application layer or by its process ID at the system layer. If the respective binary name or process ID are also captured at the system or application layer respectively, they can be represented in the *context* field.

Table 4.1: Composition of concepts for layers in DFPM in BNF form

<code><prov_graph></code>	<code>::=</code>	<code><user_layer> <app_layer> <system_layer></code>
<code><user_layer></code>	<code>::=</code>	<code><user_a><user_layer> <application_a><user_layer> <system_a><user_layer> <user_a> <application_a> <system_a></code>
<code><app_layer></code>	<code>::=</code>	<code><application_a><app_layer> <system_a><app_layer> <application_a> <system_a></code>
<code><system_layer></code>	<code>::=</code>	<code><system_a><system_layer> <system_a></code>
<code><user_a></code>	<code>::=</code>	<code>merge copy</code>
<code><application_a></code>	<code>::=</code>	<code>createfile access modify_rw modify_w datatransfer</code>
<code><system_a></code>	<code>::=</code>	<code>create generate delete open connect close read write transfer</code>

For example, a malicious PDF document can automatically download or execute background programs in the system when opened by a user. In this case, the PDF document is classified as an object, as a PDF document is known to be a file. However, when analysing the provenance, it may appear that the object is initiating activities. In such cases, the definitions listed here can be formulated into a rule for detecting abnormalities within the provenance graph.

4.3.2 Activity

Concepts defined in the Activity class (otherwise known as “activities”) are derived based on the operations used by operating systems when handling files and processes, as discussed by Silberschatz et al. [2005]. The defined concepts aim to express the data flow relationship and the construction of data channels required for data flow between entities.

DFPM is divided into user, application and system layer; three hierarchically ordered layers with the system layer being the finest granular layer. Each layer is composed of a set of layer specific activities and of the

activities defined in the layer beneath it. This hierarchical composition of activities is described using Backus-Naur Form (BNF)⁵[McCracken and Reilly 2003; Backus et al. 1960] in Table 4.1. The reason for layers to adopt activities defined in the layer below is to allow events that are semantically and structurally equivalent (i.e. events that cannot be decomposed into a group of activities in the layer below) to be modelled in the layer the event was observed at. For example, a simple file open observed at the application layer is equivalent to a file open in the system layer. Hence, by having an *open* activity defined in the application layer, granularity of the modelled events from the log file can be kept consistent.

The format used for modelling each activity into a provenance record is as follows:

```
activity_type(time, source_entity, target_entity, [context1; context2; ...])
```

Activity_type denotes the activity this record is describing and is based on concepts defined in the following subsections. *Time* records the accompanying timestamp observed in the log event and as such denotes when the activity occurred. The *source_entity* denotes the entity responsible for initiating this activity while the *target_entity* denotes the recipient entity of this activity. Lastly, the *context* field captures contextual information that provides further information that is specific to this activity.

In the following, we list only the layer-specific activities for each layer. Activities inherited from the lower layers use the same definitions, hence, are not repeated.

System Layer

The system layer is composed of concepts that model events concerning data flow and the setting up of data channels observed beneath the

⁵BNF is a context-free grammar introduced by John W. Backus and Peter Naur and is traditionally used for describing structure in a language

abstraction layer of operating systems. These events include those observed in the kernel or file system. Since the system layer is the finest granularity, the following concepts can be viewed as basic activities in DFPM.

- **Create**—models an *agent* creating a new *object*. A channel that allows data to be communicated between the *agent* and the *object* is also assumed to be established as part of the process of creating the new *object*. Either the owner of the *agent* or the *agent* itself (if the identity of the owner of the *agent* is unknown) would assume the ownership of the new *object*.
- **Generate**—models the generation of a new *entity* by an existing *entity*. Similar to *create*, the owner of the parent *agent* will assume ownership of the new *agent* immediately. However, it does not assume a data channel is automatically created between the two entities.
- **Delete**—models the attempt made by an *agent* to remove or destroy an existing *object*.
- **Open**—models an *agent* establishing a new data channel with an existing *object*.
- **Connect**—models the establishing of a new data channel between two existing *agents*. The target *agent* can be either a local (e.g. process) or remote (e.g. network host) *agent*.
- **Close**—models the termination of an opened data channel from the initiating *agent* to either an existing *object* or another *agent*.
- **Read**—models the initiating *agent* obtaining data from an existing *object* or *agent* with which it has an opened data channel.
- **Write**—models the initiating *agent* outputting data to an existing *object* or *agent* with which it has an open data channel.

- **Transfer**—models the transfer of data across the network between the initiating *agent* and another remote *agent*. Like *read* and *write*, a data channel must be established beforehand between the two *agents*.

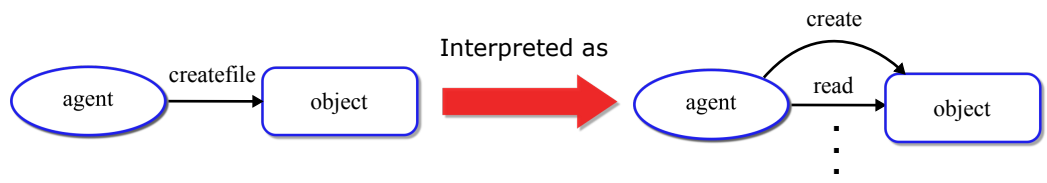
Application Layer

Concepts defined in this layer model events concerning data flow observed in applications the application layer (e.g. application log files). Such data flow can take place between other systems or even files.

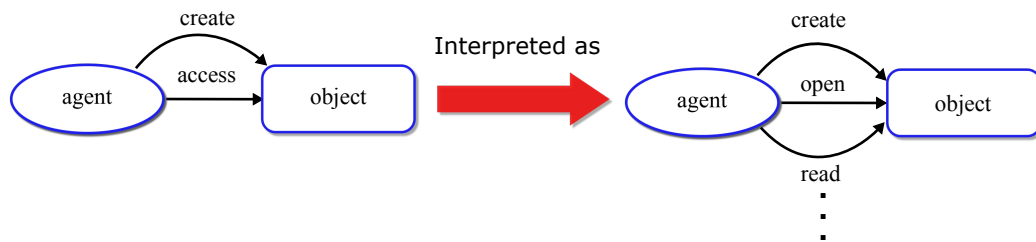
- **Createfile**—models the initiating *agent* creating a new *object*. The *agent* may subsequently perform a combination of read and write activities using the data channel created along with the *object*.
- **Access**—models the initiating *agent* performing read-only operations on an existing *object*. Such activities may retrieve partially or the entire data content of the *object*. *Access* also models the execution of the content of the *object* (e.g. execution of a script by a process).
- **Modify**—models the initiating *agent* modifying the content of an existing *object*. Unlike *access*, at least one change is made to the *object*. *Modify* is also represented by two instances, *modify_w* and *modify_rw* to distinguish between instances of blindly writing to an *object* and reading and writing to an *object*.
- **Datatransfer**—models the exchange of data between the initiating *agent* with a remote *agent* across the network. The exchange may be bi-directional (i.e. multiple *transfers* that includes sending and receiving data between both *agents*).

The key difference of *createfile* from *access* and *modify* is the creation of a new *object*. This is illustrated in Figure 4.3. Without *createfile*, a *create* activity would need to be added before *access* or *modify* to

4.3 Data Flow Provenance Model



(a) Using *createfile* to represent *object* creation and its associated *read-write* events



(b) Separating *object* creation using *create*

Figure 4.3: Difference between having *createfile* and using the *create* activity to represent file creation at the application layer

model the creation of the new object first. However, as shown later in Section 4.3.3, *create* and *open* can be used for separating system layer activities into groups. As such, prepending a *create* before the *access* or *modify* may give rise to the misinterpretation that the *read-write* events are independent of the *create* (i.e. there is an *open* in-between the *create* and the *read-write* events).

User Layer

Lastly, user layer concepts model high level entity relationships that are not captured in the application and system layers and events observed from a user's perspective.

- **Merge**—models the information flow between two existing *objects*. *Merge* denotes the convergence of either partial or full content of the source *object* into the target *object*.
- **Copy**—models the derivation relationship from an existing source *object* to a newly created target *object*. It signifies that the target *object* would contain either partial or full content of the source *object*'s content.

Chapter 4 Modelling Log Events as Provenance Relations

```
192.168.130.50 -- [18/Sep/2015:21:31:42 +1200] "GET /phpmyadmin/js/functions.js? ...  
192.168.130.50 -- [18/Sep/2015:21:31:42 +1200] "GET /phpmyadmin/js/functions.js? ...  
192.168.130.50 -- [18/Sep/2015:21:31:42 +1200] "GET /phpmyadmin/js/jquery/jquery-ui-1.8.16.custom.js? ...
```

(a) Example of events from Apache Web Server logs

```
read(21:23:42, process:apache, file:/phpmyadmin/js/functions.js)  
read(21:23:42, process:apache, file:/phpmyadmin/js/functions.js)  
read(21:23:42, process:apache, file:/phpmyadmin/js/functions.js)  
read(21:23:42, process:apache, file:/phpmyadmin/js/jquery/jquery-ui-1.8.16.custom.js)
```

(b) Modelled System layer relations may appear to be sequential reads from the same process without *open* and *close*

```
| open(21:23:42, process:apache, file:/phpmyadmin/js/functions.js) |  
| read(21:23:42, process:apache, file:/phpmyadmin/js/functions.js) |  
| read(21:23:42, process:apache, file:/phpmyadmin/js/functions.js) |  
| close(21:23:42, process:apache, file:/phpmyadmin/js/functions.js) |  
| open(21:23:42, process:apache, file:/phpmyadmin/js/functions.js) |  
| read(21:23:42, process:apache, file:/phpmyadmin/js/functions.js) |  
| close(21:23:42, process:apache, file:/phpmyadmin/js/functions.js) |  
open(21:23:42, process:apache, file:/phpmyadmin/js/jquery/jquery-ui-1.8.16.custom.js)  
read(21:23:42, process:apache, file:/phpmyadmin/js/jquery/jquery-ui-1.8.16.custom.js)  
close(21:23:42, process:apache, file:/phpmyadmin/js/jquery/jquery-ui-1.8.16.custom.js)
```

(c) *Open* and *close* activities can be used to clearly separate logically non-sequential reads

Figure 4.4: Example showing how *open* and *close* can be used to group relations into logical groups

4.3.3 Modelling Data Channels

The purpose of modelling the management of data channels is to allow series of *reads* and *writes* to be separated into logical groups of activities that are independent (i.e. not a series of reads/writes that result from the coarser-grained action). An example adapted from the Apache Web Server logs, shown in Figure 4.4, is used to explain the separation of activities.

Figure 4.4a illustrates three log events showing an Apache server accessing two different files. While the three events from the application log hinted at three groups of system layer activities surrounding reads to the two involved files, the exact number of *read* activities in each group

is unknown. Without *open* and *close*, the group membership of the *read* activities cannot be determined accurately, as illustrated in Figure 4.4b. However, with the *open* and *close* activities, relevant *read* activities can be enclosed. Series of data flow activities can then be segmented into groups that reflect events observed in the less granular layers with high confidence, as shown in Figure 4.4c. As discussed in Section 2.2, an issue with existing structural abstraction provenance models is the reliance on a-priori knowledge of elements in the provenance graph for deriving an accurate abstraction. However, it is observed that the same user action achieved through different applications can result in slightly different sets of activities in the finer-granular layers. For example, creating a new file using the *vim* document editor would result in a set of system events that differ from using the *nano* document editor. This difference stems from how applications can be implemented differently even for tasks of the same nature. In the following subsection, the approach used to discover a generic set of patterns for abstracting concepts defined in the model is discussed.

4.4 Aggregating Activities

4.4.1 Discovering Patterns

Instead of learning patterns specific to a set of applications, as discussed by Macko and Seltzer [2011] and Buneman et al. [2012], the following study focuses on discovering patterns that can be used independently of applications to map provenance relations. The study assumes that each mapping that maps a group of fine-grained activities to a coarse-grained activity (e.g. set of system layer activities to an application layer activity) has a consistent structure and hence can be derived from the patterns observed.

Based on the activities defined in the user and application layer of DFPM, sets of scenarios for accomplishing each of those activities using different applications were designed. For example, the use of different

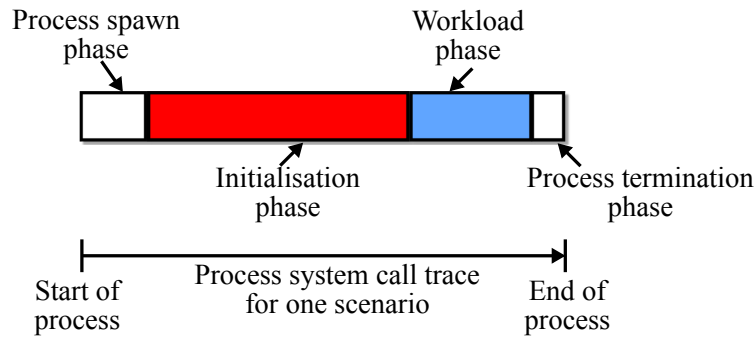


Figure 4.5: Segmentation of system call trace for each scenario

text editors and approaches for creating a file constitute the set of scenarios for the *createfile* activity. System events generated at the kernel were captured for each scenario using Progger, a tool for logging system call events [Ko and Will 2014], and translated using activities defined in the system layer⁶. The scenarios used for each activity defined in the model are listed in Appendix B for reference.

For each scenario, the modelled system call trace is manually segmented into four phases, as shown in Figure 4.5. The *process spawn phase* consists of events executed by the kernel when spawning a new process. The main objective is to establish the input and output data channels to various devices, such as the standard input and output terminals, for the process. As such, events executed mainly concern the opening, connecting and closing of pipes. This phase is almost identical for all processes. The *initialisation phase* is specific to each application type. In this phase, library functions and other functionality required by the application are loaded from system and application-specific library files. These files can be identified by looking at their full system paths. They are usually prefixed with known shared system directories such as *"/etc"* and *"/usr/lib"*. All files accessed in this phase are on a read-only basis (e.g. no *write* relations). The *workload phase* is where the application executes events for accomplishing the task set out in the scenario. The beginning of this phase is identified by the first event executed on an entity used in the scenario (e.g. opening of a file or connecting to known network host). The *workload phase* would terminate either when the

⁶The system layer activities are used as the log files are all captured at the system kernel.

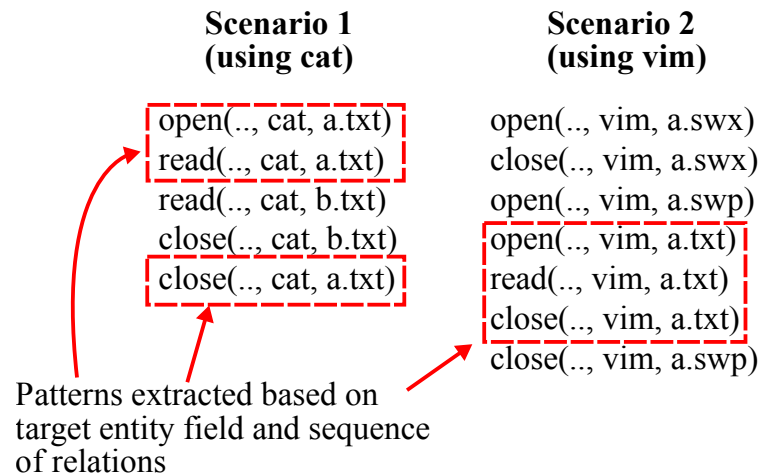


Figure 4.6: Illustration of the extraction of patterns from modelled provenance relations in the workload phase

process’s system call trace ends or when the *process termination phase* begins. The *process termination phase* is the final phase of a process’s system call trace. Its appearance is subject to how the process is being terminated; hence, it may not appear in all system call traces. It can be identified by the closing of the data channels established during the *process spawn phase*, right before the system call trace ends.

Each scenario is executed several times and the *workload phase* is extracted from each modelled system call trace. The extracted segments from scenarios within the same set (e.g. all scenarios designed for *create-file*) are then compared manually. The comparison attempts to identify common sequence of events with the same target entities. This comparison is illustrated in Figure 4.6. Common sequence of events observed across all segments are then extracted as the patterns for abstracting provenance relations to the respective model activity. We describe the identified patterns in Tables 4.2 and 4.3 using BNF. Each set of production rules describes the abstraction of activities from one layer to its immediate higher granularity layer (i.e. activities in application layer to user layer; activities in system layer to application layer). Although in practice, activities may be abstracted across multiple layers (i.e. system to user layer), we assume abstraction across multiple layers is done in multiple parses.

Since the production rules in Tables 4.2 and 4.3 are either in the form

Table 4.2: Abstracting system layer activities to application layer

<createfile_c>	::=	create<CF>
<CF>	::=	write<CF> read<CF> close
<access_c>	::=	open<AR>
<AR>	::=	read<AR> close
<modify_w_c>	::=	open<MW>
<MW>	::=	write<MWD>
<MWD>	::=	write<MWD> close
<modify_rw_c>	::=	open<MA>
<MA>	::=	read<MA> write<MD>
<MD>	::=	write<MD> read<MD> close
<datatransfer_c>	::=	open<DT> connect<CT>
<DT>	::=	connect<CT>
<CT>	::=	transfer<TR>
<TR>	::=	transfer<TR> close

Table 4.3: Abstracting application layer activities to user layer

<merge_c>	::=	access<MER> modify_rw<MER>
<MER>	::=	modify_w modify_rw
<copy_c>	::=	access<CPR> modify_rw<CPR>
<CPR>	::=	createfile

of $A \rightarrow aB$ or $A \rightarrow a$, where A and B are non-terminals and a is a terminal, we can say the grammar defined is regular [Linz 2012]. Using the approach discussed by Linz [2012], where non-terminals are treated as states and terminals as transitions, we elaborate on the patterns using Finite State Automata (FSA). The FSA is defined as follows:

$$FSA = (\Sigma, Q, \delta, q_0, F)$$

where:

- Σ is defined as the set of input alphabets represented by the concepts defined in the Activity class.

- Q denotes the finite set of states,
 $\{S, MER, CPR, CF, AR, MA, MD, MW, MWD, DT, CT, TR, TER\}$.
- δ denotes the transition function $\Sigma \times Q \rightarrow Q$ and is represented by the right hand side expressions in Tables 4.3 and 4.2⁷.
- q_0 denotes the start state, S .
- F denotes the final state, TER .

The FSA⁸ for identifying each layer specific activity are as follow:

Createfile

Createfile uses *create* and *close* to enclose relevant *read* and *write* relations that are executed using the data channel that resulted when the initiating *agent* created the *object*. While *createfile* can have zero or more *read* or *write* relations, the group must have one *create* and *close*.

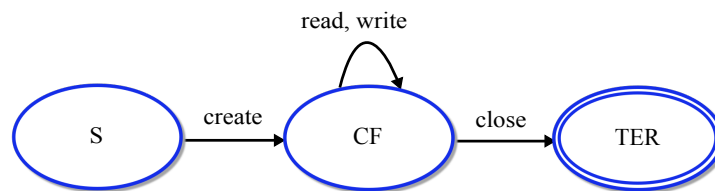


Figure 4.7: FSA for 'createfile' activity

Access

The pattern identifying *Access* is very much similar to *createfile*. Except instead of *create*, *open* is used to denote the start of the group pattern for *Access*. *Objects* can be accessed for various reasons (e.g. a process reading data from the file or executing the file, which may be a piece of code) and in some cases may not involve any *read* operations. As such, *access* can have zero or more *read* relations.

⁷Each activity-state pair is treated as a state transition rule. In this case, the '|' symbol used in the BNF can be treated as a rule separator.

⁸In the FSAs illustrated, we adopt the shorthand notation of using a comma in transition labels to simplify multiple transitions that have the same initial and destination states but with different labels into a single transition, as shown in [Hopcroft et al. 1979; Anderson 2006].

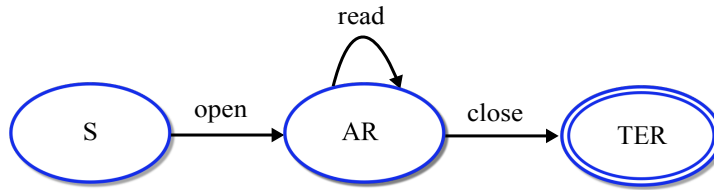


Figure 4.8: FSA for 'access' activity

Modify_w and Modify_rw

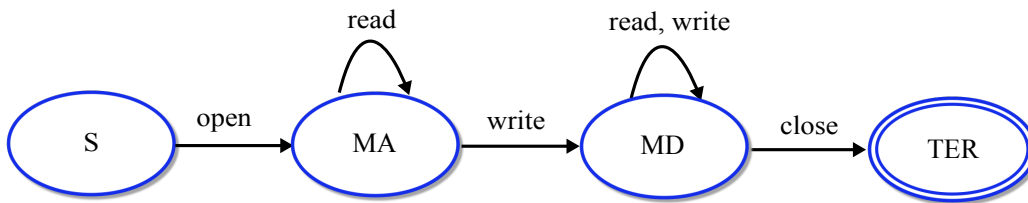


Figure 4.9: FSA for 'modify_rw' activity

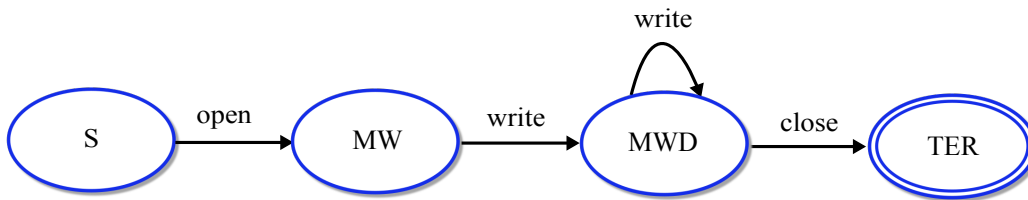


Figure 4.10: FSA for 'modify_w' activity

Modify models edits made to an *object* and hence, should contain one or more *write* relations. On the other hand, *read* relations are not essential but should not be excluded from the patterns as it is possible for *agents* to both *read* and *write* data to an *object* using the same data channel. The distinction between a process blindly writing to a file and a process reading and writing to a file is made through *modify_w* and *modify_rw* respectively.

Datatransfer

In the description of system layer activities, *connect* is explained to denote creation of a data channel between two *agents*. Hence, *connect*

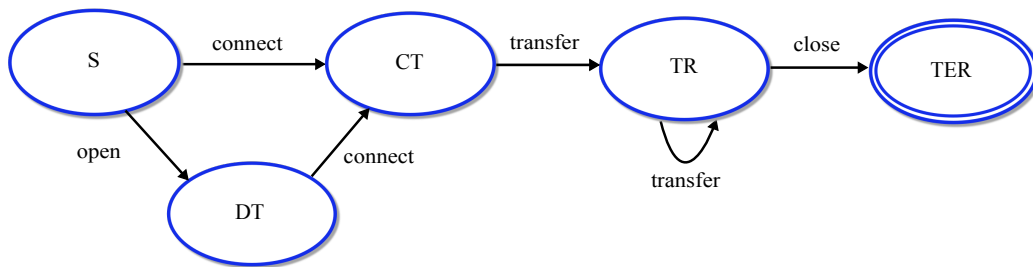


Figure 4.11: FSA for 'datatransfer' activity

can be used to mark the beginning of the set of relations that describe the communication of data between two *agents*. However, due to the way that modern operating systems work when communicating with remote systems, a network communication channel has to be associated to a socket. As a result, in cases where a socket for the network communication is newly initiated, an *open* is used to denote the creation of the socket. Like *modify*, we only identify a group as *datatransfer* if the group consists of one or more *transfer* relations, denoting that data has actually been communicated between the two *agents*.

Copy



Figure 4.12: FSA for 'copy' activity

Copy models not only the derivation of the content in the target *object* but also the creation of a new *object*. As such, a *createfile* activity should come after the initiating *agent* has obtained a full or partial copy of the data from the source *object*. Although it is possible to first create the *object* before obtaining a copy of the data from the source *object*, we simplify such cases and express them using *merge* and precede the *merge* with a *createfile* relation. It should also be noted that the pattern here precludes blind writes made to a file as the *agent* will have to obtain data from the first *object* first.

Merge



Figure 4.13: FSA for ‘merge’ activity


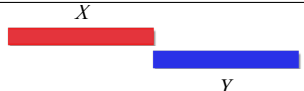
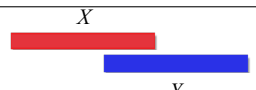
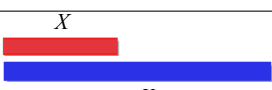
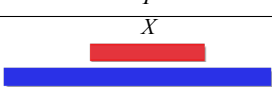
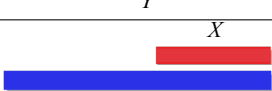
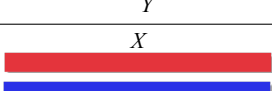
Unlike *copy*, *merge* models the derivation of data between two existing *objects*. Therefore either a *modify_rw* or *modify_w* is expected in the place of a *createfile*, indicating an edit done to the target *object*. Like *copy*, the *agent* is expected to obtain data from the first *object* first, hence a *modify_rw* is expected before the modification of the second *object*.

Unlike application layer activities, there are no activities that can be used to denote the start and end of a pattern. Hence an alternative approach to determining the relevancy of relations when automatically abstracting relations for user layer activities is required. One possible alternative is through temporal reasoning.

It is assumed that events observed close together in time are more likely to be related to each other than if they are observed a long time apart (e.g. sequence of functions called automatically to accomplish a task). Based on this assumption, the likelihood of relevancy between activities can be determined by modelling the time intervals at which the activities were observed using Allen’s interval algebra [Allen 1983]. Allen defined the thirteen relationships possible between two individual intervals shown in Table 4.4.

Pairs of application layer activities that satisfy any of the relations in the set, {*meets*, *overlaps*, *starts*, *during*, *finishes*, *is equal to*} can be grouped together automatically. The automata can then be applied to each group to determine whether the activities can be rolled up.

Table 4.4: The 13 relations defined in Allen’s interval algebra

Relation	Symbol	Illustration
X before Y	$X < Y$ $X > Y$	
X meets Y	$X m Y$ $X mi Y$	
X overlaps Y	$X o Y$ $X oi Y$	
X starts Y	$X s Y$ $X si Y$	
X during Y	$X d Y$ $X di Y$	
X finishes Y	$X f Y$ $X fi Y$	
X is equal to Y	$X = Y$	

4.4.2 Parsing

The main objective of describing the patterns identified using BNF grammar and FSA is to facilitate the understanding of how activities can be abstracted. Proposing an efficient approach for parsing the modelled provenance relations for abstraction is beyond the scope of this research.

Having said so, the parsing and detection of patterns can be achieved using Complex Event Processing (CEP) systems, such as Beepbeep [Hallé and Varvaressos 2014; Laboratoire d’informatique Formelle 2015] and Flink [Rohrman 2016] or through parsers generated using tools that supports various parsing methods (e.g. LALR(1) [Grune and Jacobs 1990]), such as HIME [Iwouters et al. 2014] and PLY [Beazley 2001]. The grammar presented in Tables 4.2 and 4.3 can either be used directly as input to parser generators, such as those listed above, or reformulated as rules for event detection systems.

In this section, we briefly describe some of our experiences in pars-

ing the stream of provenance relations. Ideally, the discussion here can be useful when considering which systems can be used for parsing and detecting patterns.

Interleaving Events

Due to concurrency in operating systems, it is common for events generated by different processes to appear in the log file in an interleaving manner. Further complicating the situation, a process can also interact with multiple objects in an interleaving manner (e.g. a process reading concurrently from two different files). The interleaving of events may affect the detection of sequential patterns. In a recent comprehensive survey on CEP systems, Hallè [2017] noted that few systems support features that allow event streams to be *sliced*, such that different computations (i.e. different patterns) can be executed over the different slices.

A simple work-around for this issue is to pre-process the stream of events, such that events are segmented based on their attributes, into segments of inter-related events. For example, events resulting from a process reading from two files concurrently can be segmented into two segments of events, one for each file. Each segment can then be parsed and computed by the event processing system. Segmentation of provenance relations modelled from log files using DFPM is discussed in details in Sections 5.2.1 and 5.5 of Chapter 5.

Arbitrary Pattern Length

In their survey on CEP systems, Hallè [2017] and Cugola and Margara [2012] noted the language used by some CEP systems only permit primitive sequential patterns, such as “A follows B”, to be expressed. More complex notions, such as negations and expressing unbounded number of sequential events (e.g. “zero or more” or “one or more”) cannot be expressed directly⁹.

⁹It is interesting to note that while some CEP systems, such as RAPIDE [Luckham et al. 1998; Luckham and Frasca 1998] allow repetitions to be expressed, they do

The ability to express patterns with arbitrary length of sequential events is important as it is difficult to predict the amount of data transfer relations (i.e. *read*, *write*, *transfer*) to expect for each pattern. For example, the number of *read* system calls a process invokes when reading data from a file is dependent on the size of data to be read. Hence, the number of *read* can only be known during runtime. Variations may also exist in the patterns shown in Tables 4.2 and 4.3. For example, applications may reuse an existing socket or open a new socket for establishing a connection with another host or process. As a result, the *open* socket event is observed only in some cases. This creates variations in the pattern that abstracts to the *datatransfer* activity.

4.4.3 Verifying the Patterns

The patterns identified are implemented as a FSA for parsing and aggregating the provenance relations. The system takes a finite list of provenance relations (e.g. the reconstructed data provenance) as input and assumes that the provenance relations are already segmented according to their attributes¹⁰. Relations in each segment is parsed and evaluated against the FSA. If a segment matches a pattern (i.e. reaches the *TER* state), the system will output the activity corresponding to the matched pattern. In contrast, if a segment does not match any patterns, the system outputs the original segment. The system completes execution once there are no more segments available for parsing.

To verify the patterns, the implemented FSA system was applied to the system log files captured in the NZCSC'15. Since the patterns focus on mapping data flow relations, the count for *read*, *write* and *transfer* relations before and after applying the FSA are shown in Figure 4.14.

The results showed that the discovered patterns were indeed able to map majority of the data flow relations to coarser-grained activities (e.g.

not allow variations (e.g. “zero or more”) in pattern definitions. Having said that, this limitation can be resolved by explicitly formulating each possible variation of the pattern as a separate rule.

¹⁰The approach used for the segmentation will be discussed in Sections 5.2.1 of Chapter 5.

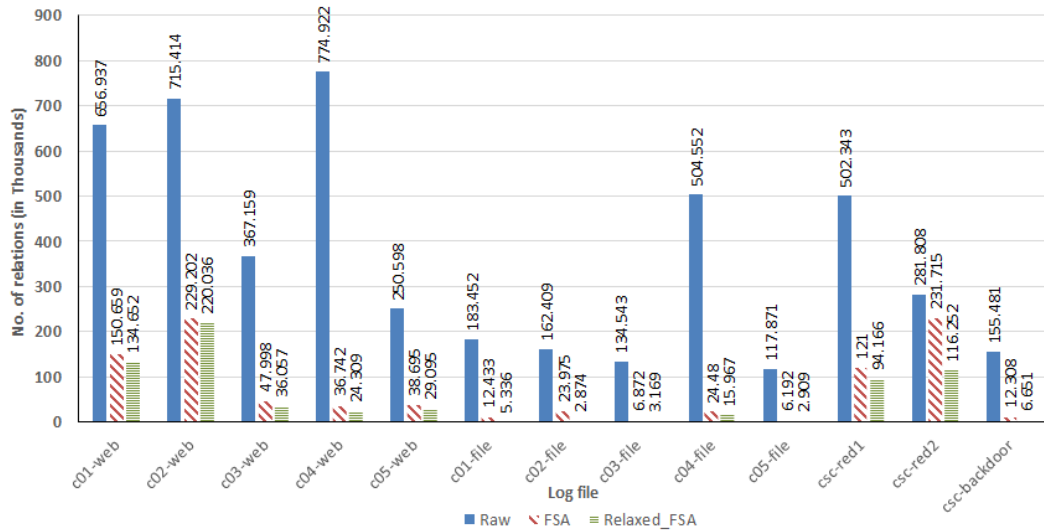
application layer activities). Investigation into the *read* and *write* relations that are not mapped successfully showed those relations were interacting with the system input and output terminals. Channels to the system input and output terminals usually do not have *open* relations associated with them as they are created at system initialisation and remain open throughout the operating system's uptime. Having said that, segments of *read* and *write* relations to some files were noticed to fail the mapping. Large amount of network transfer relations are also observed to have not been mapped successfully for log files captured from the red team. Reference to the ground truth data (i.e. the transcript and video) showed that mapping for these segments of relations was unsuccessful because the processes were terminated abruptly before the data channels could be closed (e.g. no *close* relation). Since the FSA implemented requires a *close* relation to enclose the patterns, the FSA would determine those segments to have failed the matching conditions. The workaround is to relax the patterns defined, such that matching for a *close* relation is optional. Results in Figure 4.14b showed the relax FSA was able to map majority of the network transfers.

4.5 DFPM and OPM

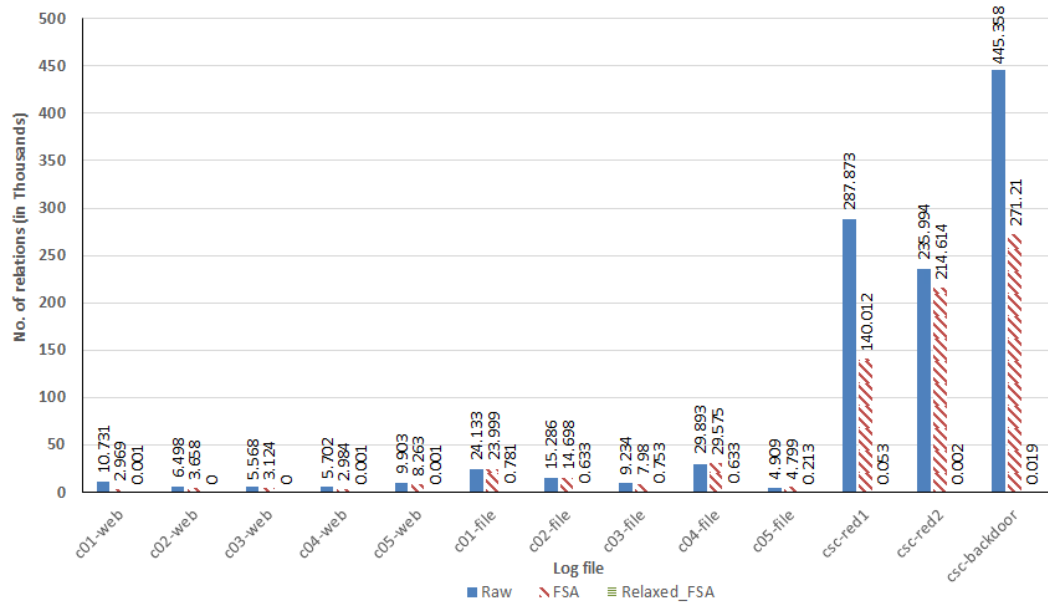
This section highlights the differences between the OPM used for modelling provenance in computer systems Gehani and Tariq [2012] and the proposed DFPM.

Modelling of Entities

The approach taken to model entities is the first difference between the two models. To explain this difference, a snippet of events, taken from a LAF log file and shown in Figure 4.15, is used. The snippet consists of three events. **Event 1** shows the parent process, *pid:26272*, creating a data channel that will be used for data communication. **Event 2** shows the spawning of the child process, *pid:26273*, and the inheritance of the



(a) Read and write relations



(b) Network transfer relations

Figure 4.14: Results for applying the FSA on the NZCSC'15 system log files

Event 1:	type=SYSCALL msg=audit(1431919799.945:49458): arch=c000003e syscall=22 ... pid=26272 auid=0 uid=0 ... comm="scp" exe="/usr/bin/scp" ... type=FD_PAIR msg=audit(1431919799.945:49458): fd0=5 fd1=6
Event 2	type=SYSCALL msg=audit(1431919799.945:49462): arch=c000003e syscall=56 ... pid=26272 auid=0 uid=0 ... comm="scp" exe="/usr/bin/scp" ...
Event 3:	type=SYSCALL msg=audit(1431919803.497:50401): arch=c000003e syscall=1 ... pid=26272 auid=0 uid=0 ... comm="scp" exe="/usr/bin/scp" ...

Figure 4.15: Events snippet from a LAF log file, simplified to show only the relevant fields

data channel set up in Event 1 by the child process. **Event 3** shows the use of the data channel for data communication from the parent to the child process.

In OPM, users are modelled as *Agents*, running processes as *Processes* and files and other passive entities as *Artifacts*. This is illustrated in Figure 4.16, where events in Figure 4.15 are modelled using the OPM model adopted by Gehani and Tariq [2012] in their system provenance monitoring tool, SPADE.

Figure 4.16 shows OPM models not only the relationship between the parent and child process, but also the owner of those processes (i.e. *user:0*). However, this level of detail is achieved at the expense of adding a large number of nodes and edges to the provenance graph.

Although this overhead is manageable when modelling events at the user layer, the same cannot be said for events at the finer granularity layers. The volume of events and entities generated is higher in the finer granularity layers due to the increasing level of detail captured in those layers. As a result, modelling all this information will result in the provenance to be overwhelmed with information. This increases the complexity and difficulty of analysing the provenance.

To reduce the impact caused by the volume of new entities and events generated, DFPM only models entities which have a direct impact on the movement of data. Figure 4.17 illustrates the result of using DFPM to model events shown in Figure 4.15. Entities such as users are modelled implicitly using the *entity* and *context* fields. This difference between

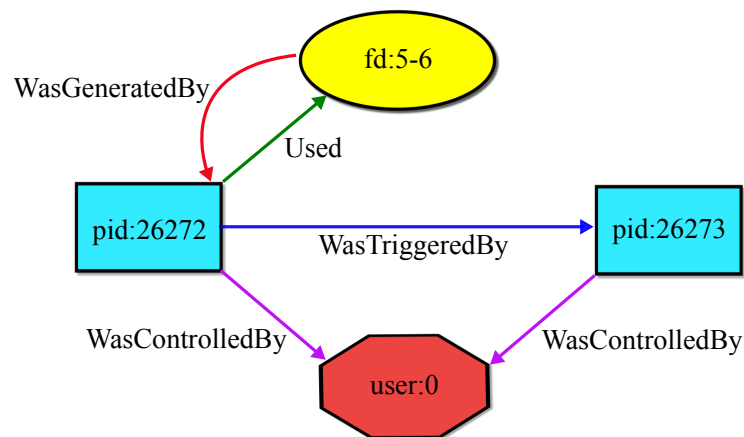
```

Agent(user:0)
Process(pid:26272)
Process(pid:26273)
Artifact(fd:5-6)

WasGeneratedBy(fd:5-6, pid:26272)
WasControlledBy(pid:26272, user:0)
WasTriggeredBy(pid:26273, pid:26272)
WasControlledBy(pid:26273, user:0)
Used(pid:26273, fd:5-6)

```

(a) Corresponding OPM provenance records



(b) Corresponding OPM provenance graph

Figure 4.16: OPM provenance resulting from modelling snippet of events from LAF log file

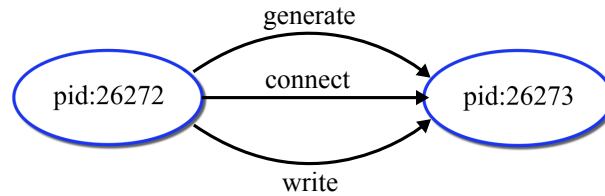
the two modelling approach can be observed by comparing Figures 4.16b and 4.17b.

As shown in Figure 4.17a, ownership association between an user and a process is implicitly captured through the *owner* field in the *agent* record. Although not shown in the graph, ownership and other *role* based association can still be visualised through the use of property graphs, where nodes and edges can be decorated with contextual information. By not explicitly modelling every entity, the resulting provenance is simplified to focus only on activities that shows how data is moved between entities.

```
agent(pid:26273, user:0, process:/usr/bin/ssh;host:192.168.2.23)
agent(pid:26272, user:0, process:/usr/bin/scp;host:192.168.2.23)

generate(1431919799.94505, pid:26272, pid:26273, host:192.168.2.23)
connect(1431919799.94505, pid:26272, pid:26273, type:pipe;readfd:5;writefd:6;host:192.168.2.23)
write(1431919803.49703, pid:26272, pid:26273, fd:6;bytes:25;host:192.168.2.23)
```

(a) Provenance records modelled using DFPM



(b) Provenance graph modelled using DFPM






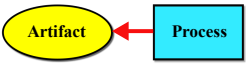
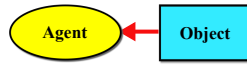
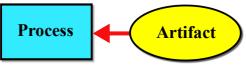
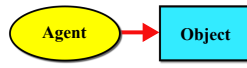
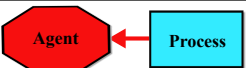
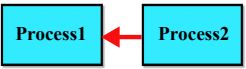
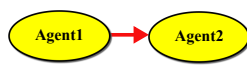
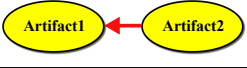
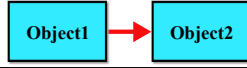
Figure 4.17: Provenance of the snippet of events in Figure 4.15 modelled using DFPM

Concepts and Terminologies

While concepts defined are based on those in OPM, DFPM uses a different set of terminology to show how data is propagated between entities.

The mapping between concepts in the two models are shown in Table 4.5. From the table, one can observe that there are no mappings between some of the concepts between the two models. From OPM, there are no mappings to model association between *Agents* and *Processes* as DFPM does not model ownership of entities. There are also no OPM equivalent relations for $\{close, delete, connect, transfer, datatransfer\}$ due to the difference in focus between the two models. DFPM models data flow and setting up of data channels while OPM emphasises how entities are associated with each other. As such, OPM is not concerned with data channels or when objects cease to exist. We argue that modelling the termination of entities is critical for provenance in computer systems as entities such as files and processes can be reused. Without knowledge of when entities cease to exist, confusion can arise when entities are being reused as activities related to the new entity will be linked to the old instance.

Table 4.5: Mapping concepts between OPM and DFPM

OPM Concepts	OPM Visual Representation	DFPM Concepts	DFPM Visual Representation
<i>Entities</i>			
Agent		Agent	
Process			
Artifact		Object	
<i>Dependencies</i>		<i>Activities</i>	
Used		Open Read Access	
wasGeneratedBy		Create Write Createfile Modify	
-	-	Close Delete	
wasControlledBy		-	-
wasTriggeredBy		Generate	
-	-	Connect Transfer Datatransfer	
wasDerivedFrom		Copy Merge	

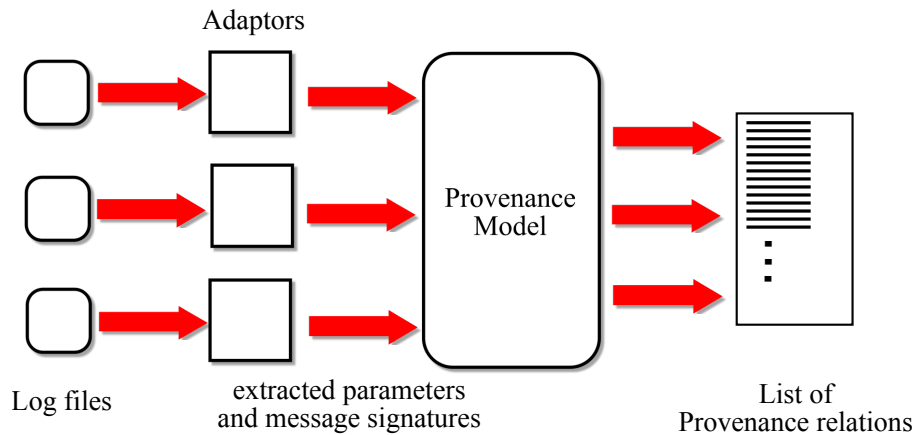


Figure 4.18: Illustration of the extraction and modelling of log events into provenance relations

Multiple Layers of Granularity

The last main difference between DFPM and OPM is the view of the provenance domain as a multi-granularity domain. The motivation of having a multi-layered model for provenance has been discussed in Chapter 2 and as such, will not be discussed again here.

4.6 Summary

DFPM models the information extracted from log events into provenance relations that describe how data flows between different entities. Through the extraction and modelling of information, heterogeneity in the log format and parameters is being normalised. This allows the following phases of the reconstruction workflow to easily consume the information.

Figure 4.18 illustrates the transformation of log events to list of provenance relations that can be used for reconstruction. First, the adaptors extract key parameters and message signatures from the log files. Using the DFPM model, the extracted information is then modelled into provenance relations.

A set of patterns are also derived and implemented as FSA for mapping provenance relations between different granularity layers. These FSA will be used for mapping the reconstructed provenance graph to the

4.6 Summary

granularity of the ground truth provenance, such that the two can be compared without disparities in their structure and semantics. Abstraction of provenance relations is done after the reconstruction and will be discussed further in Chapter 5.

After modelling log events into provenance relations, the next step is to reconstruct the data provenance graph that describes the derivation of a piece of data. Chapter 5 discusses the problem and the proposed algorithm for provenance reconstruction.

Chapter 5

Reconstructing Data Provenance

This chapter discusses how provenance of a piece of data can be reconstructed from the list of provenance relations modelled from the log files using DFPM. In Section 5.1, the problem of data provenance reconstruction is defined. Reconstruction is achieved through a two-step process. In the first step, the data provenance is reconstructed using the algorithm described in Section 5.2. Following which, the reconstructed output is mapped to the granularity layer of the ground truth provenance using the FSA discussed in Section 4.4. This is demonstrated and explained in Section 5.5.

We adopt the following notation for this chapter:

Notations:

- R set containing all pair-wise provenance relations modelled from the dataset
- E set containing all entity instances modelled from the dataset
- e the object whose data provenance is to be reconstructed, e is a member of E
- rg a group of provenance relations from R between two entities, based on the FSA defined in DFPM
- $DP(e)$ the set of provenance relations that constitute the data provenance of e

5.1 The Data Provenance Reconstruction Problem

In most situations, provenance relations in R are not focused on a single entity and thus contain provenance relations that are not relevant to $DP(e)$. This is because e is usually not known during logging, causing events to be logged in an indiscriminate manner. Hence, the problem of reconstructing $DP(e)$ from R can be viewed as finding all likely *source* and *derivative objects* of e and the paths that link each of those objects to e . An object is considered the **source of** e if:

1. The object has an instance that contains at least part of the data in an instance of e .
2. There exists a path from the object instance to the instance of e , such that it shows data being propagated from the object to e .

Likewise, an object is considered the **derivative of** e if:

1. The object has an instance that contains at least part of the data in an instance of e .
2. There exists a path from the instance of e to the object instance, such that it shows data being propagated from e to the object.

The problem of reconstructing $DP(e)$ can be decomposed into the following two sub problems:

1. **Reconstructing the derivative provenance** - Given an instance of e , find the likely *derivative objects* of e and the paths, such that each path describes how e eventually reaches the state of each derivative object.
2. **Reconstructing the source provenance** - Given an instance of e , find the likely *source objects* of e and the paths, such that each path describes how e is derived from each source object.

We define rg to be a structure that describes the flow of data between two entities, $e1$ and $e2$ (i.e. pair-wise provenance relation):

$$rg = (r, t, e1, e2) \tag{5.1}$$

5.1 The Data Provenance Reconstruction Problem

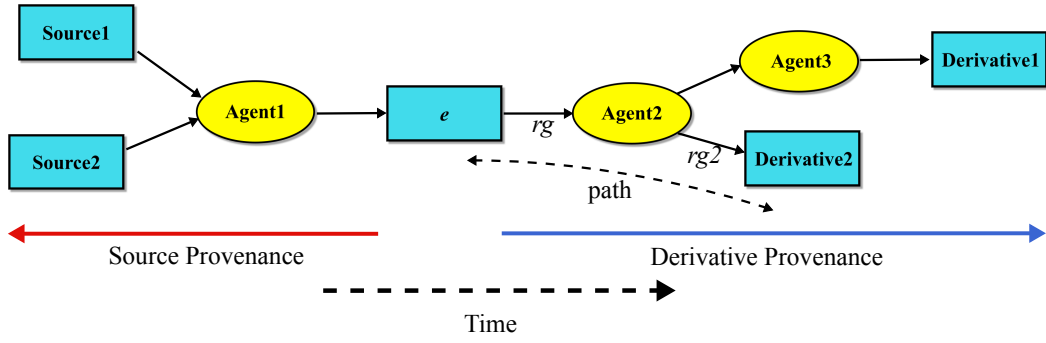


Figure 5.1: Illustration of source and derivative provenance from e

where r is a member of the set of concepts defined in DFPM, shown in Table 4.1 in Chapter 4. r also denotes the direction of the data flow and is illustrated in the last column in Table 4.5. e_1 denotes the *source_entity* and e_2 denotes the *target_entity* between the two entities connected by rg . Both e_1 and e_2 are members of E . t denotes the timestamp at which rg was first observed. Note that rg is a group containing one or more distinct provenance relations from R . The approach used and benefits of grouping is discussed in Section 5.2.1.

A path connecting two objects, o_1 and o_2 , can then be defined as a sequence of rg :

$$\begin{aligned}
 P(o_1, o_2) &= (rg_1, rg_2, \dots, rg_n) \\
 &= ((r_1, t_1, e1_1, e2_1), (r_2, t_2, e1_2, e2_2), \dots, (r_n, t_n, e1_n, e2_n)) \quad (5.2)
 \end{aligned}$$

with the following constraints:

1. $(rg_1, rg_2, \dots, rg_n)$ is ordered such that $t_{n-1} < t_n$
2. rg_1 describes the data flow between an agent and o_1 while rg_n describes the data flow to o_2 ($e2_1 = o_1 \wedge e2_n = o_2$)
3. rg in the sequence are connected to each other through the entities such that either $e1_n = e1_{n+1}$ or $e2_n = e1_{n+1}$

Note that each path does not have branches. Instead, multiple objects passing through the same entity are considered as separate paths. As such, $DP(e)$ can be composed of multiple paths joined together to form a graph. Figure 5.1 illustrates the relationship of the source and derivative provenance with respect to e and the notation defined thus far.

Algorithm 1 Algorithm for Reconstructing Data Provenance of e

- 1: **Input:** e_id ▷ user input that denotes id of e
 - 2: $R, E \leftarrow \text{readModelledData}()$ ▷ modelled relations and entities
 - 3: **Initialise:** $Q \leftarrow \emptyset, DP(e) \leftarrow \emptyset$
 - 4:
 - 5: $L_{sorted}, T_{lookup} \leftarrow \text{groupRelations}(R)$ ▷ **sort**
 - 6:
 - 7: $I, DP(e)$ ▷ **find instance**
 $\leftarrow \text{findInstances}(e_id, E, L_{sorted}, DP(e), T_{lookup})$
 - 8: $Q \leftarrow Q \cup I$
 - 9:
 - 10: $DP(e) \leftarrow \text{BF-traverse}(Q, \text{filter} = \text{findInOutEdges})$ ▷ **reconstruction**
 - 11: $DP(e) \leftarrow \text{checkDuplicates}(DP(e))$
 - 12: **return** $DP(e)$
-

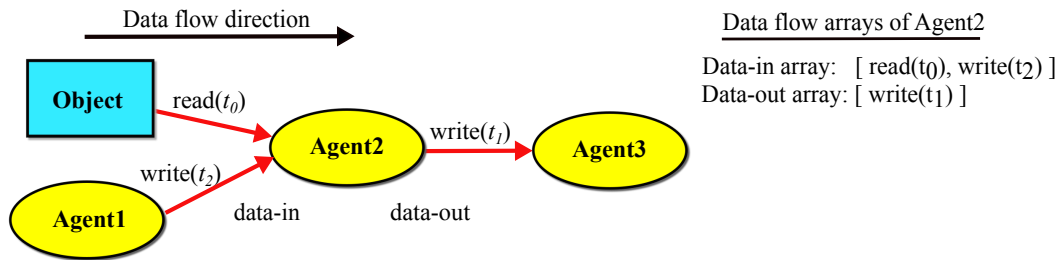


Figure 5.2: Illustration of the content of data-in and data-out arrays from Agent2's perspective

5.2 Algorithm for Reconstructing Data Provenance

Based on the problem stated in Section 5.1, an algorithm for reconstructing both source and derivative provenance for a given instance of e is proposed. The algorithm is based on the principles of the *happens-before* causal relationship defined by Lamport [1978].

The notion of applying causality to log events have been explored in various fields of research in computer science, such as workflow mining [van der Aalst et al. 2004] and intrusion tracking [King and Chen 2003; King et al. 2005]. These approaches generally determine causality between entities by observing the order of events appearing in the logs. While the algorithm proposed here adopts a similar approach of using

5.2 Algorithm for Reconstructing Data Provenance

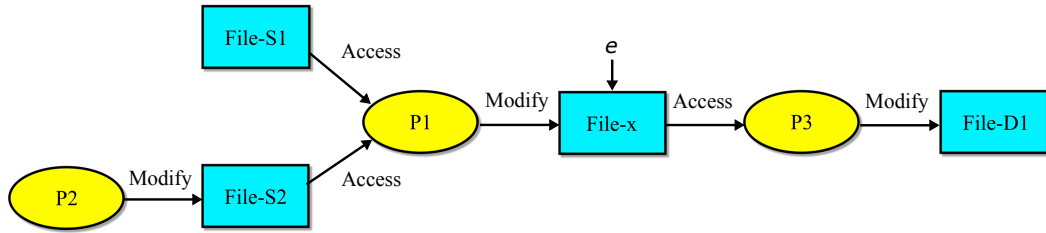


Figure 5.3: Illustration of the hypothetical use case that will be used in the explanation of the reconstruction algorithm

temporal information, the activity involved (e.g. *read*, *write*, *transfer*) is also considered when determining causality between two provenance relations. Overall, the algorithm consists of three phases, namely the sort, find-instance and the reconstruction phase. The pseudocode in Algorithm 1 gives an overview of the algorithm.

The algorithm works on the assumption that **an entity has to obtain the data before it can be propagated to other entities**. Based on this assumption, two arrays, data-in and data-out, are initialised in each entity's instance data structure. The two arrays capture the inflow and outflow of data with respect to the entity, as illustrated in Figure 5.2. Note that assignment of data flow provenance relations differs for *agent* and *object* type entities due to their definitions. The difference is discussed in Section 5.2.4.

The algorithm first locates entities that are directly connected to e , the object of interest, in the find-instance phase and initialises them to a queue, Q . Based on Q , the algorithm traverses through the data graph in a breadth-first manner. The traversal uses the function **findInOutEdges**, which is based on Lamport's *happens-before* principles, to determine which entities (vertex) or relations (edges) to visit. Visited entities or relations are placed on $DP(e)$ and returned as the final reconstructed data provenance graph.

The following hypothetical use case will be used to illustrate how the algorithm works as the different phases of the algorithm is explained. Figure 5.3 shows the data provenance for *File-x*. The provenance shows how *File-x* is being derived from files *File-S1* and *File-S2* (i.e. source objects of *File-x*) through the application, *P1*. *File-D1* is the derivative

object of *File-x*, through the application, *P3*. The provenance also shows that *File-S2* is being influenced by the application, *P2*.

5.2.1 Sort Phase

The sort phase calls the **groupRelations** function for sorting and grouping provenance relations in R into lists of rg . Relations are first sorted based on their *source_entity* field into separate lists of provenance relations and stored in a sorted list, L_{sorted} ¹. Note that R is modelled from log files gathered from across different computer systems. Hence, it is possible to have entities with the same *source_entity* value but originate from different computer systems. Contextual information such as the host ID captured in the *context* field² is used in conjunction with the *source_entity* field to uniquely identify each entity.

After sorting, provenance relations in each list in L_{sorted} are grouped into sets of rg . Grouping is done based on the FSA patterns defined in Section 4.4. Based on the patterns, sequence of provenance relations between the same pair of *source_entity* and *target_entity* can be grouped and mapped to a lesser granular activity. However, in practice, order of the log events observed from the same *source_entity* may show events to different *target_entity* interleaving each other. This can happen when a process accesses multiple files at the same time. Since provenance relations are modelled from log events, the interleaving behaviour will be reflected in the order of the modelled relations. Therefore, to account for interleaving of events, the algorithm also considers the value of the *target_entity* in addition to the *activity_type* when determining group membership of each provenance relation. This is done in conjunction with using the FSA patterns for determining the start and end of a rg .

Figure 5.4 visualises the sorting phase using the use case given in Figure 5.3. Although grouping is done based on the FSA patterns, the algorithm does not replace each group with the coarser-grained activity associated to the matched pattern. This is to preserve the granularity so as

¹We assume that the modelled provenance relations are already in chronological order.

²*Context* field is defined in the format of an activity (see Section 4.3.2).

5.2 Algorithm for Reconstructing Data Provenance

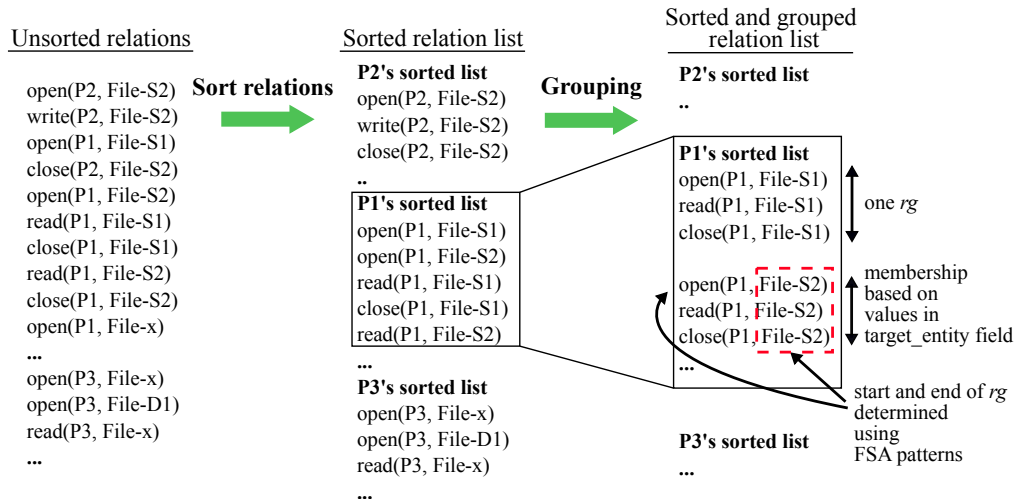


Figure 5.4: From unsorted relations to list of *rg*, illustrated based on use case shown in Figure 5.3

to allow the search for relevant provenance relations to be as precise as possible. For the rest of the discussion on the algorithm, we use “sorted relation list” or L_{sorted} to reference the sorted and grouped relation list, unless otherwise indicated.

The motivation behind grouping provenance relations into *rg* is to facilitate the retrieval of data channel provenance relations associated with the relevant data flow provenance relations. As pointed out in Section 4.3.3, data channel relations are an integral part of segmenting relations into logical groups and the FSA. Hence, by grouping relations, the algorithm can focus on analysing data flow relations when reconstructing the paths. The associated data channel relations can be retrieved together with the group when one of the data flow relations in the group is determined to be relevant to $DP(e)$.

A lookup table is also generated while parsing provenance relations during sorting to facilitate the search for related entities in other phases of the algorithm. The table holds a relational mapping between entities and returns a set of entities that are associated (i.e. have one or more provenance relations that connect the two entities) to a given entity.

The mapping for an *agent* entity is updated whenever the corresponding agent in the *source_entity* field is observed to be associated to a new *target_entity*. Mapping of an *object* entity is updated when the corresponding object in the *target_entity* field is observed to be associated

with a new *source_entity*. The update of mapping is done differently between *agent* and *object* entities due to objects being regarded as passive (e.g. not able to initiate activities).

5.2.2 Find Instance Phase

After the sort phase, the algorithm retrieves e from E and attempts to find all *agents* directly connected to e using the **findInstances** function. The pseudocode for the function is shown in Algorithm 2. We use the form $rg.r$ to represent attributes associated to rg (e.g. the provenance relation associated to rg). $i.dIn$ and $i.dOut$ represents the data-in and data-out arrays associated to the entity instance i .

Retrieval of e is done using e_id , a user supplied string that denotes the ID of the *object* whose data provenance the user wants reconstructed³. Once e is retrieved, the algorithm starts reconstructing the source and derivative provenance of e . Since e is an *object* entity, entities that connect directly to e are expected to be of *agent* type (e.g. *agents* executing activities in relation to e).

A list of IDs of agents associated with e is retrieved using the lookup table constructed in the sorting phase. Based on the list, the instance that represents each agent is retrieved from E . The algorithm then looks for rg that connects each *agent* directly to e by searching for rg with e as its *target_entity* value in the respective *agent*'s sorted relation list. Each qualified rg is assigned to the *agent*'s respective data flow array. Assignment of rg to the arrays is based on the data flow direction described by the provenance relation. Details of the assignment is discussed later in Section 5.2.4. Once all *agents*' sorted relation list are processed, the *agent* instances are returned and inserted into Q for processing in the main function.

Referencing the example use case in Figure 5.3, Figure 5.5 shows the progress of reconstructing at the end of the find instance phase. Instances of *agent* directly connected to *File-x*, *P1* and *P3*, are retrieved

³Note that e will always be an *object* type entity (e.g. file object) due to the scope of our research.

Algorithm 2 Function: **findInstances**

```

1: Inputs:  $e\_id, E, L_{sorted}, DP(e), T_{lookup}$ 
2: Initialise:  $E_{new} \leftarrow \emptyset$ 
3:
4:  $e \leftarrow E[e\_id]$ 
5: if  $e$  is null then terminate algorithm
6:  $S \leftarrow T_{lookup}[e]$ 
7:  $DP(e) \leftarrow DP(e) \cup \{e\}$ 
8:
9: for each  $s \in S$  do
10:    $i \leftarrow E[s]$ 
11:    $RG_{read} \leftarrow \{rg \in L_{sorted}[s] \mid (rg.r = read \wedge rg.e2 = e)\}$ 
12:    $i.dIn \leftarrow i.dIn \cup RG_{read}$ 
13:
14:    $RG_{write} \leftarrow \{rg \in L_{sorted}[s] \mid (rg.r = write \wedge rg.e2 = e)\}$ 
15:    $i.dOut \leftarrow i.dOut \cup RG_{write}$ 
16:
17:    $E_{new} \leftarrow E_{new} \cup \{i\}$ 
18: end for
19: return  $E_{new}, DP(e)$ 

```

from E . rg that directly connects the *agents* to *File-x* are assigned to the respective *agents'* data flow array. Q shows the *agents* that will be processed in the next phase of the reconstruction.

5.2.3 Reconstruction Phase

Source and derivative provenance are reconstructed in an iterative manner by finding directly connected rg that can logically explain the 'next step' of the data flow. This is done in two steps.

First, the algorithm identifies the entity on which to perform the search by checking for the ID of the instance being processed. Based on the ID of the instance, the entity's sorted relation list is retrieved from L_{sorted} . Next, the algorithm tries to find matching rg from the retrieved list for each rg in the instance's data-flow arrays.

Matching is done using Lamport's *happened-before* causal relationship [Lamport 1978]. Lamport defined a *happened-before* b if a comes before

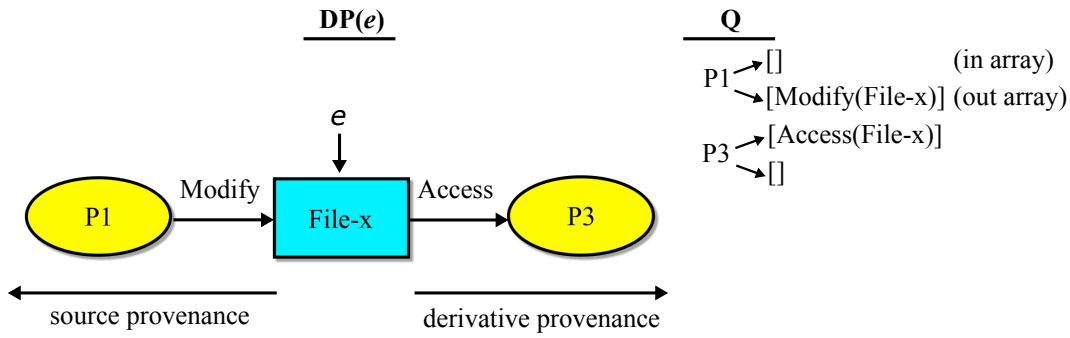


Figure 5.5: State of $DP(e)$ at the end of find instance phase (with reference to the example use case)

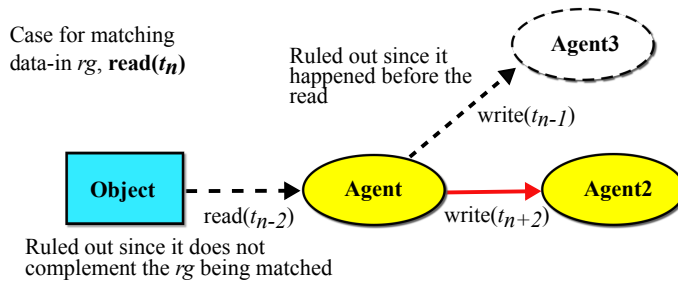


Figure 5.6: Finding a match for a data-in rg on Agent. Dotted arrows represent ruled out provenance relations

b in terms of clock order (in our case, we use timestamp) and that b is the recipient of the message sent by a . A rg is considered to be causally related (i.e. a match) if it meets the following two conditions:

1. The rg in the provenance relation list complements the rg in the data flow array in terms of the direction of data flow. For example, to match a data-in rg , the rg in the provenance relation list should describe data flowing out of the entity (i.e. *write* provenance relation).
2. The timestamp on the pair of rg satisfy the causal order stated by our assumption in the beginning of Section 5.2. That is to say, the timestamp on a data-in rg should be smaller or equal to the timestamp on a data-out rg .

Figure 5.6 illustrates the conditions for the matching. Assuming that the algorithm is matching a *read* at time t_n , the *read* on the object by **Agent** would fail to complement the *read* in the data-in array since both provenance relations describe data flowing into **Agent**. The *write* to

5.2 Algorithm for Reconstructing Data Provenance

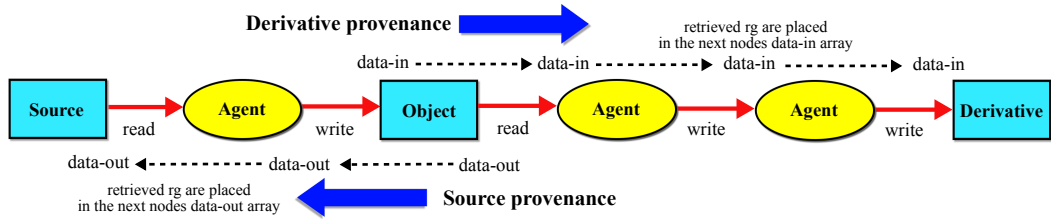


Figure 5.7: Illustrating the concept of data-in and data-out rg at agents along a path

Algorithm 3 Function: **findInOutEdges**

```

1: Inputs:  $i, E, L_{sorted}, T_{lookup}, DP(e)$ 
2: Initialise:  $E_{new}$  ▷ To hold new nodes
3:
4: if  $i$  is an agent then
5:    $E_{new}, i \leftarrow \text{findAgentEdges}(i, E, L_{sorted})$ 
6: else if  $i$  is an object then
7:    $E_{new}, i \leftarrow \text{findObjectEdges}(i, L_{sorted}, T_{lookup}, E)$ 
8: end if
9:  $DP(e) \leftarrow DP(e) \cup E_{new}$ 
10: return  $E_{new}, i$ 

```

Agent3 would fail to meet the conditions too due to the timestamp being smaller than the data-in *read*.

For every instance in Q , part of the derivative provenance is reconstructed by searching for matching data-out rg in the entity's sorted relation list for each rg in the data-in array. Source provenance is reconstructed in a similar fashion, except the algorithm searches for matching data-in rg for each rg in the instance's data-out array. For each rg matched, the algorithm retrieves the instance for each unique entity introduced by the rg from E . The newly matched rg is then assigned to the mirroring data flow array on the new instance the rg is matched to (e.g. if matching rg in data-in, newly matched rg is placed into the new instance's data-in array). This process is visualised in Figure 5.7 and is implemented in the **findInOutEdges** function used in the reconstruction phase. The general structure of **findInOutEdges** is shown in the pseudocode described in Algorithm 3. For clarity, the searching of rg is described separately in Section 5.2.4 (Algorithms 4 and 5).

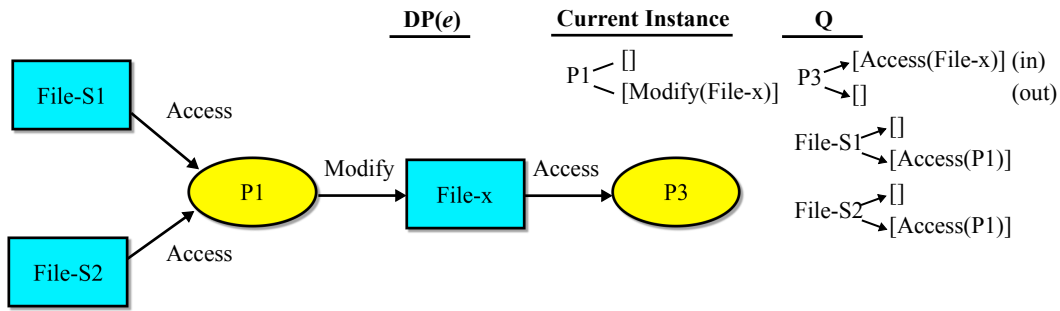


Figure 5.8: Progress of reconstructing the example use case after processing $P1$

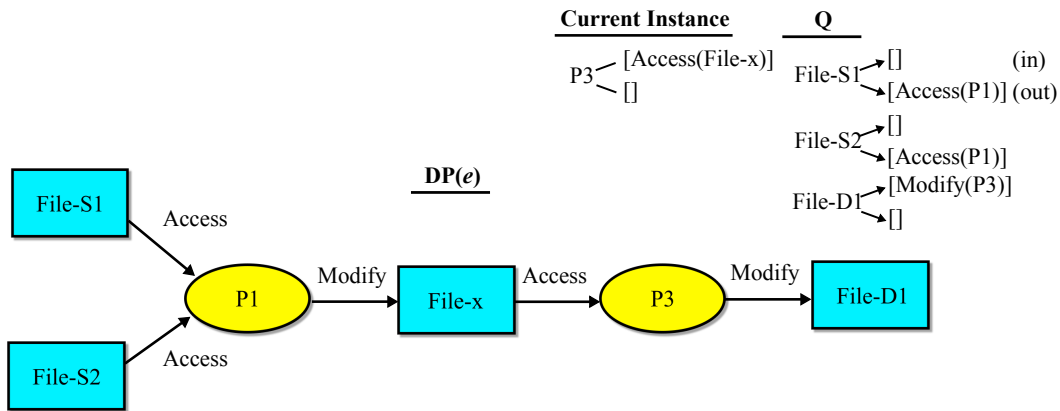


Figure 5.9: Progress of reconstructing the example use case after processing $P3$

The **findInOutEdges** function is called by the traversal for every instance in Q . Newly discovered instances at the end of **findInOutEdges** are placed onto Q . The processed instance is then placed into $DP(e)$. The reconstruction stops once Q is empty. The algorithm then checks for duplicated provenance relations in $DP(e)$ before returning it as the result of the reconstruction.

Figure 5.8 and 5.9 illustrate two iterations of the reconstruction phase when reconstructing the example use case. Calling **findInOutEdges** when processing $P1$ will result in the discovery of $File-S1$ and $File-S2$ due to the rg connecting them to $P1$. In the example, it is assumed that these two rg satisfy the conditions for the matching. Once $P1$ is processed, the newly discovered entities, together with the rg that connects them to $P1$ are added to Q . The state at the end of the second itera-

Algorithm 4 Function: **findAgentEdges**

```

1: Inputs:  $i, E, L_{sorted}$ 
2: Initialise:  $E_{new} \leftarrow \emptyset$ 
3:
4:  $t_{small} \leftarrow \text{findSmallestTime}(i.dIn)$ 
5:  $t_{large} \leftarrow \text{findLargestTime}(i.dOut)$ 
6: for each  $rg \in L_{sorted}[i]$  do
7:    $n \leftarrow E[rg.e2]$ 
8:    $n.dIn \leftarrow n.dIn \cup \{rg \mid rg.r = \text{write} \wedge rg.t \geq t_{small} \wedge rg.e2 \notin DP(e)\}$ 
9:    $n.dOut \leftarrow n.dOut \cup \{rg \mid rg.r = \text{read} \wedge rg.t \leq t_{large} \wedge rg.e2 \notin DP(e)\}$ 
10:
11:    $i.dOut \leftarrow i.dOut \cup (\{rg\} \setminus n.dIn)$ 
12:    $i.dIn \leftarrow i.dIn \cup (\{rg\} \setminus n.dOut)$ 
13:    $E_{new} \leftarrow E_{new} \cup (\{n\} \setminus DP(e))$ 
14: end for
15: return  $E_{new}, i$ 

```

tion is shown in Figure 5.8. Note the rg placed in the data-out array for *File-S1* and *File-S2* as the algorithm reconstructs the source provenance from *P1*. Likewise, the third iteration, shown in Figure 5.9, processes *P3* in a similar manner using **findInOutEdges**. Over time, the algorithm reconstructs the source and derivative provenance one step at a time.

5.2.4 Agents and Objects

Differences in the definitions of *agent* and *object*⁴ resulted in the search for relevant rg for the two entities to be slightly different. Algorithm 4 and 5 contain the pseudocode for the search function for *agents* and *objects* respectively. *Objects* in DFPM are treated as passive entities; they are unable to initiate activities by themselves. This implies that *object* entities will not appear in the *source_entity* field of a provenance relation. Since rg in the sorted relation list are sorted based on the *source_entity* field, the algorithm will need to know which entity's list to look at. This is done by retrieving the list of entities that have interacted with the *object* using the T_{lookup} table.

⁴The two entities are defined in Section 4.3.1.

Algorithm 5 Function: **findObjectEdges**

```

1: Inputs:  $i, L_{sorted}, T_{lookup}, E$ 
2: Initialise:  $E_{new} \leftarrow \emptyset$ 
3:
4:  $t_{small} \leftarrow \text{findSmallestTime}(i.dIn)$ 
5:  $t_{large} \leftarrow \text{findLargestTime}(i.dOut)$ 
6:  $S \leftarrow T_{lookup}[i]$ 
7: for each  $s \in S$  do
8:    $RG_{derive} \leftarrow \{rg \in L_{sorted}[s] \mid (rg.r = \text{read} \wedge rg.t \geq t_{small} \wedge rg.e2 = i)\}$ 
9:    $RG_{source} \leftarrow \{rg \in L_{sorted}[s] \mid (rg.r = \text{write} \wedge rg.t \leq t_{large} \wedge rg.e2 = i)\}$ 
10:
11:    $n \leftarrow E[s]$ 
12:    $n.dIn \leftarrow n.dIn \cup \{rg \in RG_{derive} \mid (rg.e1 \notin DP(e))\}$ 
13:    $n.dOut \leftarrow n.dOut \cup \{rg \in RG_{source} \mid (rg.e1 \notin DP(e))\}$ 
14:
15:    $i.dOut \leftarrow i.dOut \cup (RG_{derive} \setminus n.dIn)$ 
16:    $i.dIn \leftarrow i.dIn \cup (RG_{source} \setminus n.dOut)$ 
17:    $E_{new} \leftarrow E_{new} \cup (\{n\} \setminus DP(e))$ 
18: end for
19: return  $E_{new}, i$ 

```

Another implication of the difference in definition is the interpretation of the data flow direction for each rg . Due to *object* entities being passive, *write* relations always originate outside of the object, causing *write* to be seen as an inflow of data from the *object's* perspective. *Read* relations are also interpreted in the same manner; reading data from an *object* is seen as data flowing out from the *object's* perspective. This contrasts how *read* and *write* relations are interpreted for *agents*.

Agent entities are viewed as active entities. Hence, provenance relations associated with an *agent* originate from the *agent* (i.e. *source_entity* is the agent). This changes how *read* and *write* are interpreted from an *agent's* perspective. A *read* is seen as obtaining data from other entities, thus is viewed as data flowing into the *agent*. Likewise a *write* is equivalent to data being output from the *agent*. The interpretation of data flow for both entity types is illustrated in Figure 5.10.

5.2 Algorithm for Reconstructing Data Provenance

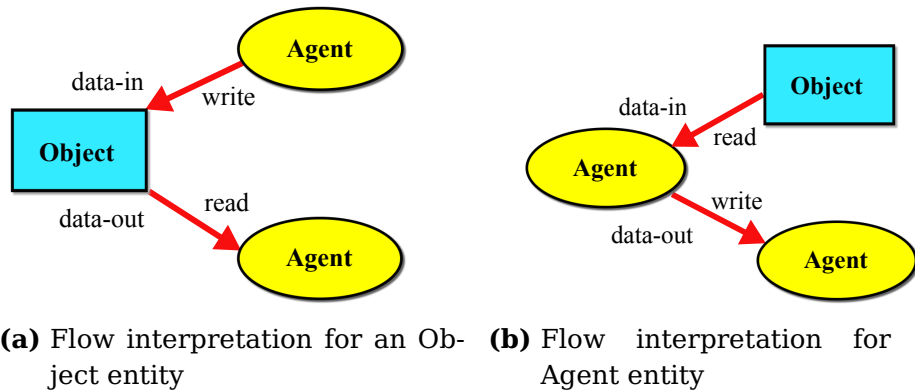


Figure 5.10: Difference in the interpretation of provenance relations in terms of data flow direction between Object and Agent entities

5.2.5 Reducing the Number of Iterations on Relation List

The path reconstruction approach described in Section 5.2.3 tries to find all matching pairs of rg for each rg in the respective data flow array⁵ of the entity being processed. This is due to a lack of information on data content being communicated between the two entities described by each provenance relation. As a result, the algorithm can only determine likelihood of a rg being related by determining if two rg are causally related. Figure 5.11 depicts an example scenario that illustrates the issue of not being able to accurately identify the relevant provenance relation.

It is assumed that both *write* provenance relations satisfy the matching conditions for determining causal relation when matched with the *read* provenance relation in Figure 5.11. Without knowing the actual data being communicated (shown as the letters in round brackets beneath each edge), the algorithm cannot accurately determine which *write* relation is the actual relevant activity. Therefore, the algorithm would have to include all rg that satisfy the causal order requirement so as not to overlook the ‘correct’ provenance relation. This results in the dependency explosion problem, which will be discussed in Section 6.4.

The algorithm iterates through the entity’s sorted relation list n times, where n is the number of rg in the respective data flow array, so as to

⁵Data-in and data-out for reconstructing derivative and source provenance respectively.

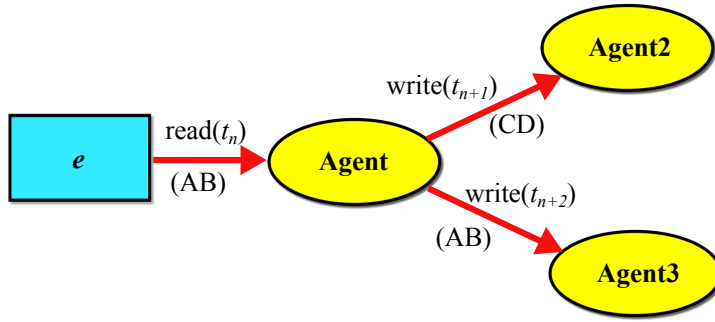


Figure 5.11: Example scenario for why considering causal order alone cannot accurately determine relevancy of a rg . Letters in round brackets under each edge represents the content of the data communicated at each edge

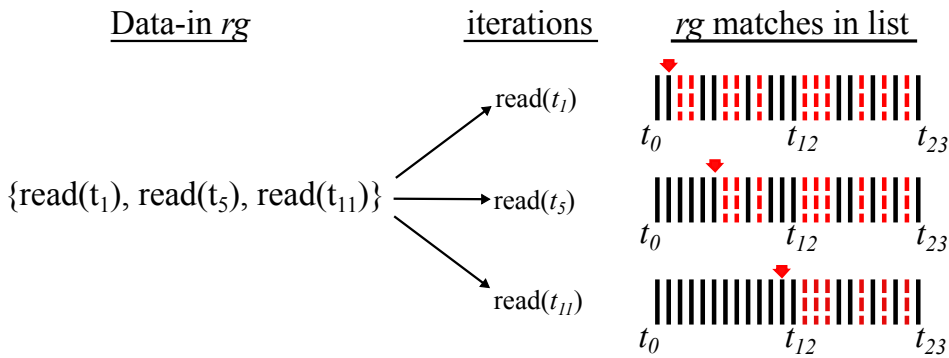


Figure 5.12: Overlapping of results (dotted bars) for each iteration in the path reconstruction process

find all rg that satisfy the causal order requirement. Since each iteration looks for rg of the same type (e.g. finding matching $write$ rg in the list for every rg in data-in and vice versa), the search can be optimised to just one pass through the list. A sample case, illustrated in Figure 5.12, where the algorithm is trying to match rg in an entity's data-in array is used to demonstrate how the number of iterations can be reduced.

In the sample case, we consider an entity with three rg in its data-in array being removed from Q for processing. The entity's data-in array is shown on the left side in Figure 5.12, with the rg simplified to show only their time sequence order. The assumed matched rg in the entity's provenance relation list is shown on the right hand side of the figure. The red arrow marks the starting point in the list for provenance relations that satisfy the causal order requirement for each iteration. The dotted

bars indicate the rg that complement the rg in the data-in array.

It can be observed from Figure 5.12 that the results returned from the rg with a larger timestamp is a subset of the results from matching the rg with the smallest timestamp. Based on this observation, simply matching the rg with the smallest and largest timestamp in the data-in and data-out array, for reconstructing derivation and source provenance respectively, will return the set of rg that satisfy all rg in the respective arrays. This optimisation is reflected in the implementation of the algorithm, where the respective timestamps are computed at the beginning of **findAgentEdges** and **findObjectEdges** in Algorithm 4 and 5 respectively.

5.3 Analysis of Algorithm

In this section, we analysis the complexity of the algorithm by analysing each phase of the algorithm. The sort phase attempts to sort relations in R according to their *source_entity* field (i.e. *agent*) and group relations into rg . Sorting and grouping is done in two separate parses, once through R and the other through the relation list for each identified *agent*.

In the first parse, the algorithm extracts the *source_entity* field and inserts each relation at the end of its respective relation list. This requires $|R| * (O(1) + O(1))$ number of operations. We can treat the cost of insertion as constant as the relation is always inserted at the end of the list. Thus the index for the insertion can be stored independently to speed up the insertion⁶.

In the second parse, each relation is passed to the **groupRelations** function. The function either advances its state by one step based on the input relation⁷ or outputs the result of the pattern matching. In either case, the number of operations scale with $|R|$. Hence, we can assume a worst-case complexity of $O(R)$ for grouping the relations. The lookup table that stores the set of entities related to each unique entity, T_{lookup} ,

⁶We already assumed the relations in R are sorted chronologically before hand.

⁷Advancement of the states is determined by the FSA patterns discussed in Section 4.4 of Chapter 4.

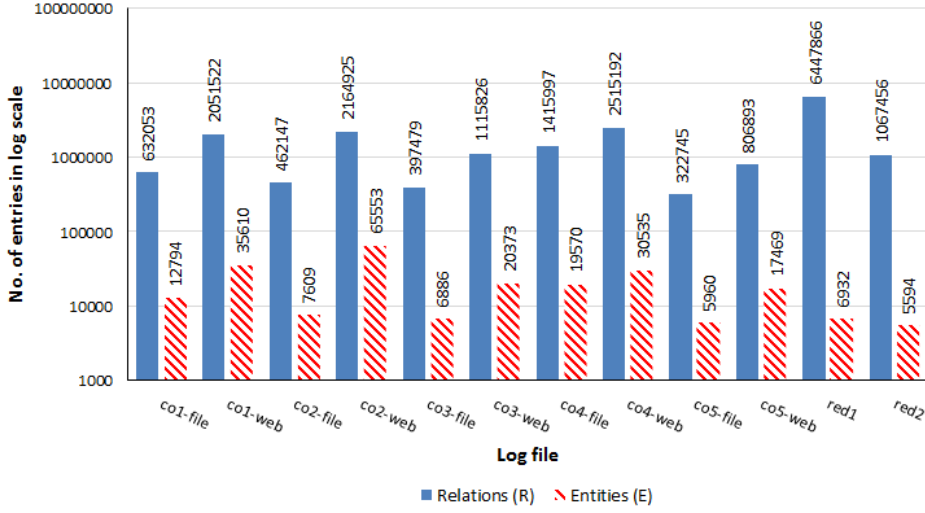


Figure 5.13: Comparison between the amount of relations and entities for system log files in the CSC'15 dataset

is also generated concurrently during this parse. This can be done by extracting the *target_entity* field from each relation during the parsing. By using a hash table with linked list data structure, the table can be generated using $|R| * (O(1) + O(1))$ number of operations. By summing the complexity for each parse, we can assume the time complexity of the sort phase to be $O(R * (1 + 1)) + O(R) + O(R * (1 + 1))$ and can be simplified to $O(R)$ worst-case complexity.

In terms of space complexity, since R needs to be stored in L_{sorted} , the amount of memory required by L_{sorted} is proportional to $|R|$. For the worst-case scenario, we can assume every entity is connected to every other entity. In such a case, the memory required to store the lookup table T_{lookup} will be $|E| * |E|$, where E is the entity set. Hence, the overall worst-case space complexity required for the reconstruction to take place is $O(R) + O(E^2)$.

Theoretically, the space required by the lookup table appears to be large. However, in practice, $|E|$ is significantly smaller than $|R|$. For example, system logs from the NZCSC'15 dataset shows $|E|$ is usually smaller than $|R|$ by at least a factor of 100 for systems used in the competition. Figure 5.13 shows the comparison between the two sets for each system log file collected from the NZCSC'15. The worst-case scenario

where every entity is inter-connected is also unlikely as not all processes would communicate with one another directly in a computer system (e.g. background processes performing routine system maintenance may not interact at all with user applications).

In the find-instance phase, the algorithm attempts to find all relations and entities directly connected to e , the object whose data provenance is to be reconstructed. Retrieving, S , the set of entities directly connected to e can be done in constant time using the hash lookup table, T_{lookup} and e . On the other hand, retrieving all relations directly associated to e will require the algorithm to scan the relation list for each s in the sorted relation list L_{sorted} . Since the size of the relation list of s refers to the amount of relations connected to s , $|L_{sorted}[s]|$ is the degree of s , $deg(s)$. As such, the time complexity for the find-instance phase can be said to be $O(1 + |S| * deg(s))$, where $|S|$ is the number of unique entities directly connected to e .

In the reconstruction phase, the algorithm uses the **findInOutEdges** function to determine which relation and entity should be considered when reconstructing the source and derivative provenance of e . In the following analysis, we assume the worst-case scenario where all entities in the input data are relevant (i.e. $E \subset DP(e)$).

To reconstruct both source and derivative provenance, the algorithm first computes the time window for relations to qualify. This is achieved by finding the smallest and largest timestamp using the data-in and data-out arrays respectively for each entity being processed. In the worst-case scenario, the arrays would represent the *in* and *out* degrees of an entity as all relations would be considered relevant to $DP(e)$. Hence, the complexity of finding the time window for each entity, i , being processed is $O(deg(i))$.

After computing the time window, the algorithm will search through the relation list for each unvisited entity, s , that is connected to i . As discussed above, the complexity of searching the relation list of each entity is $deg(s)$. Since every entity will be processed in the worst-case scenario, the worst-case time complexity of the reconstruction phase can be said to be $O(|E| * (2 * deg(s)))$.

5.4 Context-based Provenance Reconstruction Algorithm

Prior to the algorithm described in Section 5.2, an algorithm that leverage on Allen's interval and the context information of each provenance relation was explored. The context-based algorithm was inspired by the concepts of precondition used in classic workflow planning [Vassos 2012] and description logic matching used in web service composition [González-Castillo et al. 2001]. The underlying idea is to keep track of the data obtained by each entity.

Relevance between two potential causally related provenance relation (e.g. pair of *read* and *write* activities) is determined in two steps:

1. check if the temporal relationship between the two provenance relations falls into one of the following Allen's interval algebra: *meet*, *overlaps*, *starts*, *during*, *finishes*, *is equal to*.
2. match contextual information of pairs of provenance relations that satisfy the first check.

A data structure, *precondition*, is used to keep track of data held by each entity. In reconstructing derivative provenance, *read* relations from an entity are treated as the an entity obtaining a piece of data. An information set consisting of contextual information surrounding the data, such as size of the data read or information regarding the data channel, is constructed and assigned to the respective entity in *precondition*. When a *write* relation from the same entity is encountered, contextual information of the *write* relation is matched against those in *precondition*. The matching results in one of the following three possibilities:

1. **Equivalent** - *write*-information set matches exactly to at least one contextual information set in the *precondition* data structure.
2. **Sub-match** - *write*-information set is considered to be a subset for each information set matched in *precondition*.
(e.g. $size_{write} < size_{read}$)

5.5 Abstracting the Reconstructed Provenance

3. **Super-match** - *write*-information set is a superset of at least one information set in *precondition* (e.g. $\text{size}_{\text{read}} < \text{size}_{\text{write}}$) and is not a subset or equivalent to any information set in *precondition*.

For **equivalent** and **sub-match** results, the *write* relation is treated as having propagated data to other entities. Hence, the relation is added to $DP(e)$. In the case of **super-match**, the algorithm assumes that the data propagated is not relevant. This is because the data being written is larger than what was obtained by the entity and hence is likely not relevant to the data the provenance is concern of. As a result, the relation is ignored. Source provenance is reconstructed in the same manner. The only difference is instead of populating *precondition* when a *read* is observed, the update is done when a *write* is observed. Likewise, matching with information set in *precondition* is done when a *read* is found.

During testing, it was observed that the algorithm worked well in situations where data was propagated unmodified (i.e. creating duplicated copies of a document). However, in cases where data is being modified before being propagated to other entities, the logic used in the algorithm would fail. An example of such a case is when a process

Such cases are common in users' interaction with computer systems where

Hence, the context-based algorithm was scraped as it could not reconstruct the data provenance if the data is being modified during propagation. Having said that, the lesson learnt from the context-based algorithm led to the idea of leveraging the causality of relations for the reconstruction.

5.5 Abstracting the Reconstructed Provenance

The work by Coe et al. [2014] highlighted how provenance graphs generated at different granularities can differ in shape and size, even if they are describing the same events. These disparities between graphs of dif-

ferent granularity will result in false negatives⁸ when evaluating the reconstructed data provenance (i.e. $DP(e)$) against the ground truth provenance. This is because the ground truth provenance is derived from user layer events⁹ while the data provenance is reconstructed from log files generated at finer granularity layers. To resolve the difference in granularity, the FSA discussed in Section 4.4, is used to map $DP(e)$ to the granular layer which contains the ground truth provenance.

Provenance relations in $DP(e)$ are first sorted into separate lists of provenance relations based on their *source_entity* value and ordered chronologically by timestamp. Since the FSA focus on relations between a pair of *source_entity* and *target_entity*, sorting the provenance relations allows the FSA to perform the matching by simply iterating through each list.

The automaton takes in a list of provenance relations as a sequence and attempts to find sub-sequences that match the patterns defined by the FSA. Once matched, each sub-sequence is replaced with the corresponding coarser granularity activity ascribed to the matched pattern. For example, if a sub-sequence matches the pattern describing *modify*, the sub-sequence is summarised and replaced with a *modify* relation. The new provenance relation inherits the *source_entity* and *target_entity* values of the sub-sequence¹⁰. However, a single *time* field is insufficient to represent timestamps of the summarised relations. Instead, time is represented as a range in the new provenance relation. The timestamp of the first and last provenance relation in the sub-sequence is used as the start and end time of the range. Start time is captured in the time field as per the defined format, while end time is inserted into the context field with the tag '*endTime*' to denote the context of the value. Figure 5.14 illustrates the sorting and the outcome of the mapping by the automaton, using sample provenance relations simplified to show the *source_entity* and *target_entity* values of each provenance relation.

⁸A single *read* observed in the user layer may be made up of multiple *read* operations in the finer granularity layers.

⁹Derivation of the ground truth provenance was discussed in Section 3.6.

¹⁰Similar to how grouping of provenance relations is done in Section 5.2.1, the automaton only looks at provenance relations with the same *source_entity* and *target_entity* when searching for sub-sequences.

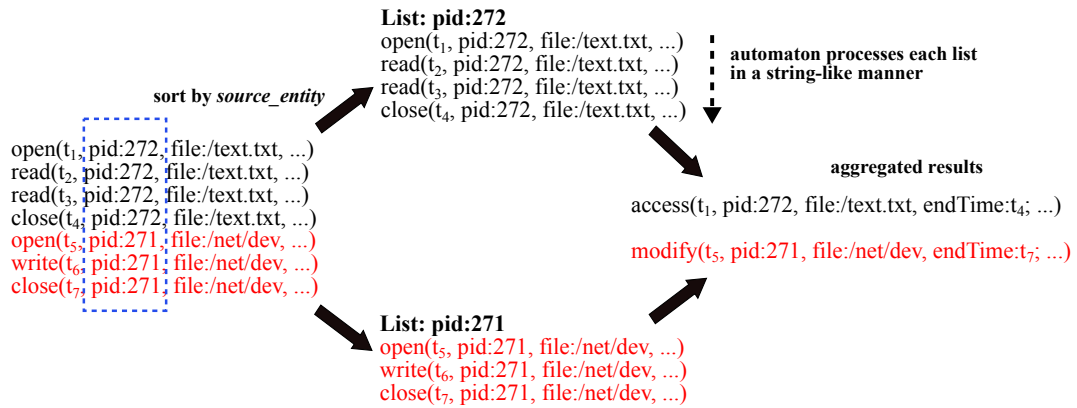


Figure 5.14: Example of the aggregation process

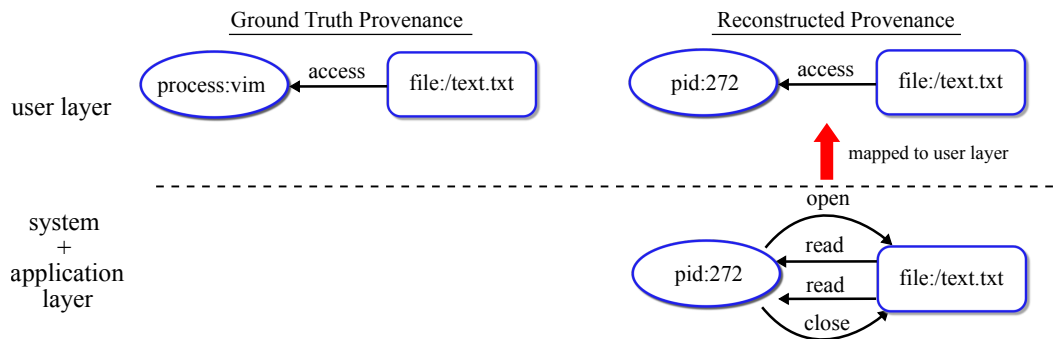


Figure 5.15: Illustration of the ground truth and reconstructed provenance graph

Once $DP(e)$ is mapped to the layer with the same granularity as the ground truth provenance, they can be compared equally as each observed event can be treated as structurally equivalent, as illustrated in Figure 5.15. Any difference between the two provenance graphs can now be treated as unexpected results produced by the reconstruction.

5.6 Summary

This chapter focuses on reconstructing the data provenance from provenance relations modelled from log files. An algorithm, based on Lamport’s *happened-before* causal relationship, is proposed for achieving this. Once reconstructed, the FSA discussed in Chapter 4 is used to map the data provenance to the same granularity layer as the ground

Chapter 5 Reconstructing Data Provenance

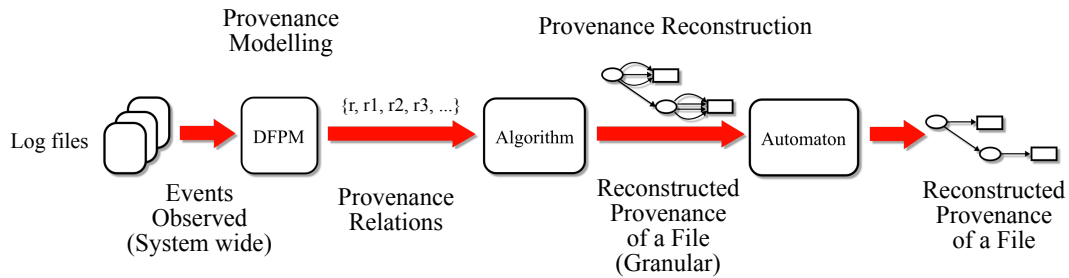


Figure 5.16: From log files to reconstructed provenance

truth provenance. This allows the reconstructed data provenance to be evaluated against the ground truth provenance without false negatives—caused by the difference in granularity in the results. Figure 5.16 illustrates the reconstruction process, from log files to the reconstructed data provenance.

Once the data provenance can be reconstructed from log files, the next step is to evaluate how complete is the reconstructed provenance with respect to the known derivation history of the data. However, review of existing literature showed the lack of a methodology for evaluating data provenance reconstruction. We discuss our proposed evaluation methodology and the evaluation of the reconstructed data provenance in Chapter 6.

Chapter 6

Experiments

To assess the reconstructed data provenance, in the context of addressing the thesis question, we require an evaluation methodology. However, Chapter 2’s review showed the absence of such a methodology in the literature. To address this gap, a methodology that evaluates a reconstructed data provenance based on its completeness and correctness is proposed in Section 6.1. Section 6.2 discusses the two errors found in the dataset and discussed how they were resolved. The setup used for the experiments is also discussed. The thesis question is then addressed in Section 6.3 through a series of experiments that look at the data provenance reconstructed from different types of log file. Based on observations obtained, Section 6.4 discusses approaches that can be applied to reducing the number of errors in the reconstructed output.

6.1 Methodology for Evaluating Data Provenance Reconstruction

The proposed methodology for evaluating reconstructed data provenance is based on two dimensions related to information quality: *correctness* and *completeness*. These dimensions were inspired by the work of Cheah and Plale [2012] on assessing the quality of provenance captured in an automated manner. The authors argued that other dimensions, such as timeliness, validity and uniqueness are more applicable when assessing information curated manually. This argument conforms to the context of

this research as the reconstruction is done in an automated manner.

Although inspired by the work of Cheah and Plale [2012], the definitions and approach to calculating the two dimensions differs in our proposed methodology. Correctness and completeness are defined as follow:

Correctness - denotes the extent to which the reconstructed provenance is error free with respect to the ground truth.

Completeness - denotes the extent to which the reconstructed provenance captured the provenance relations found in the ground truth.

To simplify the evaluation, any provenance relations in the reconstructed output but not in the ground truth are considered as errors. These errors include provenance relations that are duplicated, redundant (i.e. noise) or with erroneous fields.

Different approaches can be used to quantify these dimensions. For example, Cheah and Plale [2014] count and score the number of errors found on each node and edge of a provenance graph. A single quality score that defines the quality of a provenance graph is then calculated by averaging the sum of scores for each element over the number of expected nodes and edges. Such a quality score allows different provenance graphs to be compared and ranked based on their quality. However, applied to evaluating provenance reconstruction, the single scoring approach cannot highlight which dimensions the reconstructed provenance failed in. Our proposed methodology measures the two dimensions separately using the metrics *precision* and *recall*. We argue that by using different metrics, areas in the reconstructed data provenance that require improvement can be identified through the evaluation. This is demonstrated in the later sections on the experiments.

In the context of the proposed methodology, *precision* measures the number of reconstructed relations that are in the ground truth against the total number of reconstructed relations. It quantifies how much of the reconstructed output is correct. *Recall* measures the amount of ground truth being reconstructed and reflects completeness.

Comparison of provenance relations¹ between reconstructed and ground

¹A provenance relation is a construct that consists of two entities connected by an ac-

6.1 Methodology for Evaluating Data Provenance Reconstruction

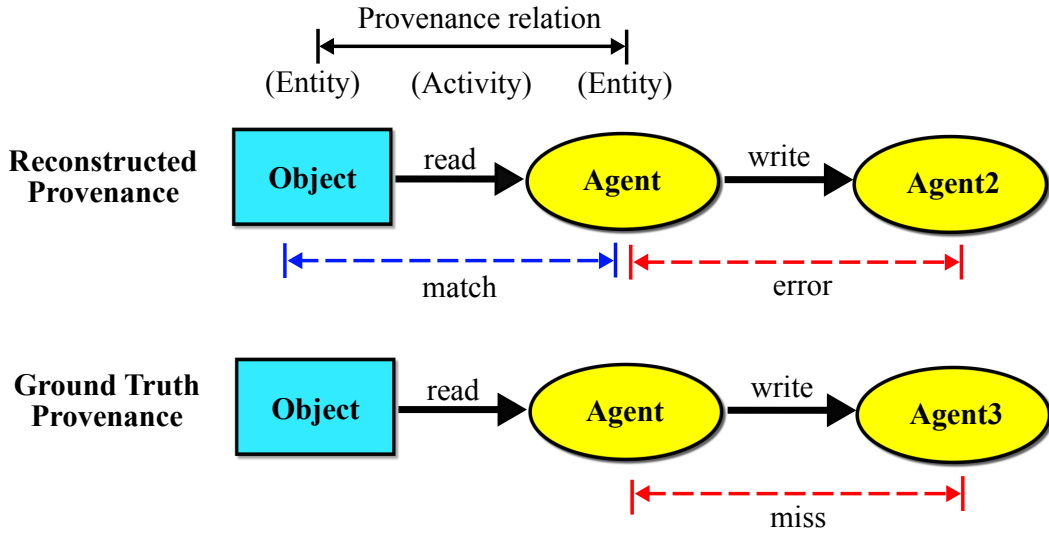


Figure 6.1: Illustration of a pair-wise provenance relation and the possible outcomes of comparison between the ground truth and reconstructed provenance

truth data provenance is done in a pair-wise manner. Each comparison results in one of the following outcomes:

1. **match** (true positive, tp) - the same pair-wise provenance relation exists in both the reconstructed and ground truth provenance
2. **error** (false positive, fp) - a pair-wise provenance relation exists only in the reconstructed provenance
3. **miss** (false negative, fn) - a pair-wise provenance relation exists only in the ground truth provenance

Figure 6.1 illustrates the three outcomes. Note that the approach to take the minimum count when computing tp , proposed by Papineni et al. [2002], is adopted in our proposed methodology².

Based on the tabulated outcomes, **recall** of the reconstructed data provenance can be computed as follows:

$$\mathbf{Recall} = \frac{\text{match}}{\text{length of ground truth}} = \frac{tp}{tp + fn} \quad (6.1)$$

²tivity that describes the relationship between the two entities, as defined previously in Section 4.3.

²This is with reference to the discussion in Section 2.4 on how tp or *matches* can be calculated differently.

Likewise, **precision** of the reconstructed data provenance can be computed as follows:

$$\mathbf{Precision} = \frac{\text{match}}{\text{length of reconstructed provenance}} = \frac{tp}{tp + fp} \quad (6.2)$$

Based on Equation 6.2, the complement of precision (i.e. $1 - \textit{precision}$) would inform how much error is in the reconstructed provenance. It should also be noted that the formulas used in Equation 6.1 and 6.2 conform to the standard formulas defined by Manning et al. [2009]. An ideal result of the reconstruction is where the reconstructed data provenance only contains matching relations and completely captures the ground truth. This implies precision and recall would equate to 1, indicating the reconstructed output mirrors the ground truth provenance.

6.2 Experimental Setup

This section details the setup used for the experiments described in this chapter. The first subsection describes issues found in the log files that can be resolved at the pre-processing stage (i.e. before running the reconstruction algorithm). The second subsection documents the experiment setup.

6.2.1 Data Pre-processing

Initial inspection of the NZCSC'15 dataset, discussed in Chapter 3, uncovered two types of sequencing error amongst the log entries. These errors can affect the execution of the reconstruction algorithm. Hence they need to be identified and rectified in the data pre-processing stage. In this section, we discuss what these errors are and how they can be resolved.

Since the procedure for generating kernel log entries by system logging mechanisms is the basis for the occurrences of the errors, it is first explained to facilitate the discussions later on.

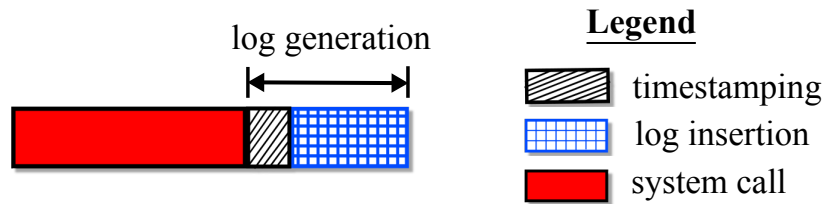


Figure 6.2: Illustration of the procedure for kernel log entry generation

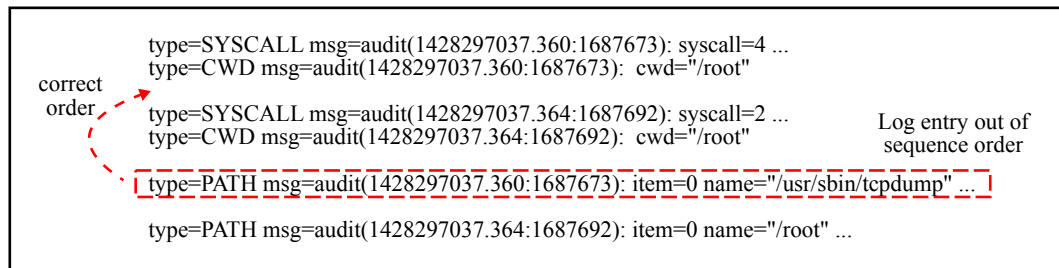


Figure 6.3: Log sequence error in Linux Audit Framework (LAF) logs

Before describing the errors found, the procedure of how kernel log entries are generated is explained first. This is so as to facilitate the discussions later on. The procedure can be divided into three phases, namely execution of the system call, event timestamping and log insertion. When a process invokes a system call, the kernel would execute the system call first. Upon completion, the event is timestamped by the logging mechanism. Parameters to be logged, including the timestamp, are then extracted from the various kernel data structures. These parameters are assembled according to the log format and inserted into the kernel ring buffer. Log entries are then retrieved from the ring buffer in a First-in-First-out (FIFO) order and output to log files in the user space. Depending on the log format, one or more log entries may be generated for each system call. Figure 6.2 briefly illustrates this procedure. It is important to note that while certain system calls are idempotent (e.g. cannot be interrupted during execution), the kernel regards the log generation portion to be non-critical and thus can be interrupted (e.g. pre-empted).

The first type of sequencing error found is **log sequence error**, where the order of two log entries in the same log file does not match the order of their timestamp. An example of a log sequence error in the LAF logs is shown in Figure 6.3.

Log sequence errors occur commonly in kernel log files of systems where the number of concurrent processes running challenge the system's capacity. In high load situations, processes may be context-switched out of the CPU before it is able to completely emit all required log entries. This results in log sequence errors.

Process concurrency in modern systems is achieved through context switching at the CPU. This is managed by the scheduler and is performed differently based on the scheduling algorithm used. In the rest of this discussion, the round robin scheduling algorithm³ is used as an example [Silberschatz et al. 2012]. Under round robin scheduling, CPU usage is managed through time-slices being allocated to each process. At the end of each time-slice, if the process is still running, the scheduler would pre-empt it from the CPU in favour of the next assigned process. By default, the size of a time-slice is usually based on the number of processes running on the system. As a result, each time-slice becomes significantly smaller when the system is heavily loaded.

Under a normal system load, each time-slice allocated to the process would be sufficiently large for the process to complete the entire procedure, from invocation to inserting log entries into the ring buffer, as shown in Figure 6.4a. However, when the system experiences heavy load, the time-slices become significantly smaller. This results in increased possibility of a process being pre-empted during the insertion of its log entries into the ring buffer. If the next process also does insertion, the log entries of the previous system call would be interleaved with entries from the current system call. Since the log entries are timestamped before the insertion and pre-emption, the interleaved entries will appear out of sequence. Figure 6.4 briefly illustrates how log sequence error could result from smaller time-slices. Note that, although other scheduling algorithms do not utilise time-slices for managing access to the CPU, as long as pre-emption of process is enabled, log sequence errors can take place when a process is pre-empted.

Log sequence error is an issue when extracting information from the log files for modelling provenance relations. In situations where an event

³Most modern operating systems use round robin as the default scheduling algorithm.

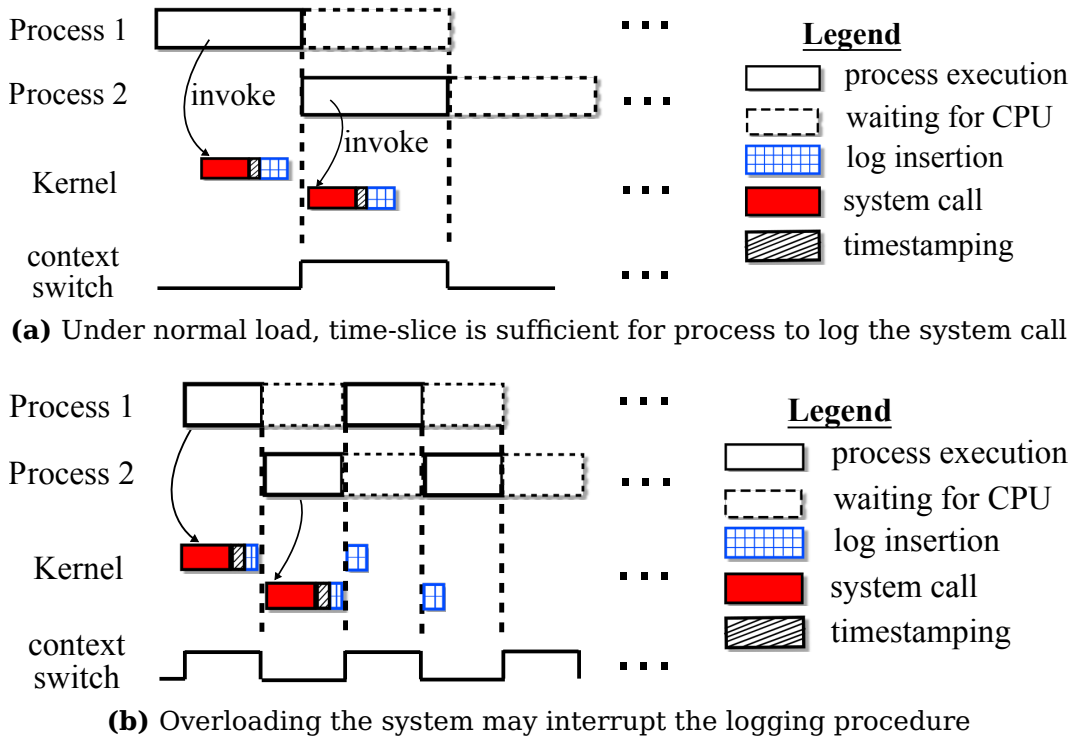


Figure 6.4: Impact of system load on logging

is described using multiple log entries (e.g. Figure 6.3), the parser would expect the respective entries to be in sequence. Log sequence errors may cause the parser to miss out on certain key information, hence affecting the modelling of entities. Log sequence errors can be remedied easily by sorting the log entries based on their timestamp.

The second type of sequencing error found is a **logic sequence error**, where the timestamps of a pair of causally dependent entries are logically incorrect. Assuming a pair of causally dependent events A and B, where B is dependent on A (e.g. $A \rightarrow B$), the timestamp of A is expected to be smaller than B as A has to happen before B. However, in logic sequence error, the timestamp of A is larger than B.

In idempotent system calls, such as *read* and *write*, the main function of the system call (e.g. during reading or writing of bytes) cannot be pre-empted. However, timestamping of the event is not considered critical and as such is susceptible to interruption by the scheduler. As a result,

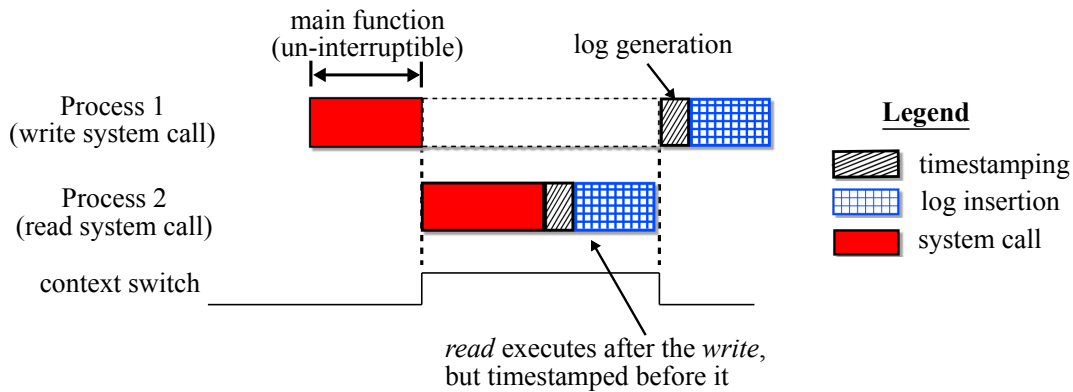


Figure 6.5: Logic sequence error due to system call being interrupted before it can be timestamped correctly

if the process invoking a system call was context-switched out by the scheduler before it managed to timestamp the system call event, logic sequence error would likely occur. With reference to the log generation procedure illustrated in Figure 6.2, Figure 6.5 demonstrates how a logic sequence error could result due to context-switching. It should be noted that logic sequence errors only manifest between two or more interacting processes. System call invocation within the same process is sequentially ordered (e.g. previous system call must be completed and logged first). As such, the order of events within the same process can be assumed to be logged correctly.

Our approach for resolving logic sequence errors in the NZCSC'15 dataset focuses on pairs of provenance relations that describe process communication. Such pairs can be identified by searching for write-read relation pairs, where the *write* shows process A writing to process B while the *read* shows process B reading from process A. Each identified pair is then checked for the error by comparing their timestamps.

Once found, the error can be fixed by changing the *write* timestamp to be smaller or equal to the *read* timestamp or changing the *read* timestamp such that it is larger or equal to the *write* timestamp. However, any changes made to the timestamps has to be done without violating the sequence order within the respective processes (e.g. if the *write* timestamp is changed, the sequence order of the *write* amongst the events observed

Algorithm 6 Resolving logic sequence error

```

1: Assume:
2:  $wbt$  = timestamp of relation before write
3:  $wt$  = timestamp of write relation
4:  $rt$  = timestamp of read relation
5:  $art$  = timestamp of relation after read
6:
7: if ( $wbt \geq rt$ ) and ( $art > wt$ ) then           ▷ Change read timestamp
8:   Set  $rt \leftarrow wt$ 
9: else if  $wbt < rt$  then                           ▷ Change write timestamp
10:  Set  $wt \leftarrow rt$ 
11: end if

```

for that process should remain the same). Algorithm 6 shows the pseudocode for determining which timestamp is to be changed.

6.2.2 Description of Experimental Setup

Our experimental setup is based on the NZCSC'15 dataset and use cases discussed in Chapter 3. Log entries in each log file of the dataset are first sorted to resolve log sequence errors. After which, log entries are modelled into a set of pair-wise provenance relations using the proposed DFPM, described in Chapter 4. The modelled set of provenance relations is then checked for logic sequence errors using Algorithm 6. The provenance relation set and the name of the file whose data provenance is to be reconstructed are given as inputs to the two-step reconstruction process⁴ to produce the reconstructed data provenance for each use case. Note that each use case is reconstructed from its own set of relations. Finally, the ground truth provenance for each use case, discussed in Section 3.6, is compared against the reconstructed provenance, for evaluation. The process of producing the reconstructed and ground truth provenance for evaluation is illustrated in Figure 6.6.

Precision and recall for the reconstructed data provenance for each

⁴First the data provenance is reconstructed using the proposed reconstruction algorithm then abstracted to a higher level of granularity using the automaton discussed in Chapter 4.

Chapter 6 Experiments

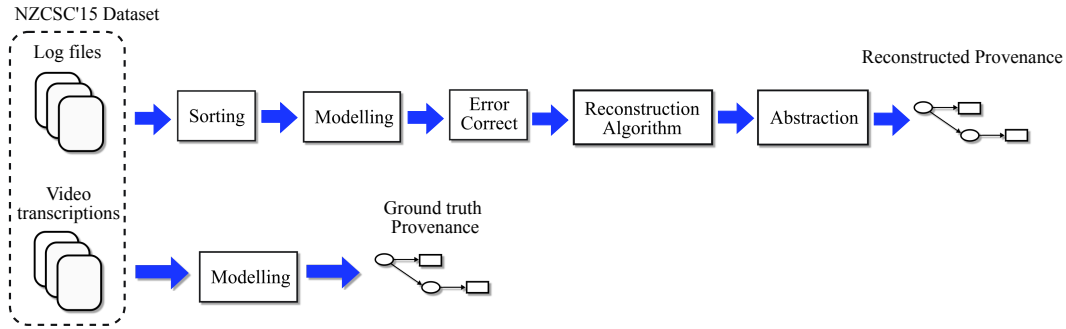


Figure 6.6: Workflow for reconstructing the data provenance and generating the ground truth from the NZCSC'15 dataset

use-case are then calculated using the methodology described in Section 6.1. To describe how matching is done, the notation defined in Equation 5.1, shown in Chapter 5, for a pair-wise provenance relation is used:

$$rg = (r, t, e1, e2)$$

where r is the activity between the two entities $e1, e2$, t is the timestamp indicating the time the activity happened, $e1$ is the source entity and $e2$ is the target entity. A pair of provenance relation between the reconstructed and ground truth provenance is considered a match only if:

1. the timestamp, t_{rp} , of the relation from the reconstructed provenance is within the threshold window δ , with respect to the timestamp of the relation from the ground truth provenance
2. the activity, r , source $e1$ and target $e2$ for both relations hold the same value

As the time granularity in the ground truth is in minutes, δ is set to one minute in the experiments described in Section 6.3.

The pseudocode for computing *precision* and *recall* is given in Algorithm 7. Agents which use process ID for their entity ID would have their entity ID field replaced with the corresponding process name, if captured

Algorithm 7 Algorithm for computing *precision* and *recall*

```

1: Input:  $RP, GT$ 
2:
3:  $GTlen \leftarrow GT.length$  ▷ Length of ground truth
4:  $RPlen \leftarrow RP.length$  ▷ Length of reconstructed prov
5:
6: for all  $rg_{rp}$  in  $RP$  do
7:    $mapIDtoName(rg_{rp})$ 
8: end for
9:
10: for all  $rg_{rp}$  in  $RP$  do
11:   if  $GT.length = 0$  then exit for-loop
12:
13:   for  $rg_{gt}$  in  $GT$  do
14:     if  $match(rg_{rp}, rg_{gt}) = 1$  then
15:        $GT.remove(rg_{gt})$ 
16:        $RP.remove(rg_{rp})$ 
17:       break
18:     end if
19:   end for
20: end for
21:
22:  $tp \leftarrow GTlen - GT.length$ 
23:  $precision \leftarrow tp/RPlen$  ▷ Initial length of RP is  $tp + fp$ 
24:  $recall \leftarrow tp/GTlen$  ▷ Initial length of GT is  $tp + fn$ 

```

in the context portion of the relation. This is to facilitate matching as Agents are known by their process name in the ground truth.

For evaluation, use cases discussed in Chapter 3 are reconstructed using the NZCSC'15 dataset. To recap, the use cases are derived from the NZCSC'15 video transcripts. Together, they cover different aspects of interactions between elements in an operating system and data provenance. These aspects include process-to-process communications, process-to-file interactions and the source and derivative provenance of file objects. A brief summary of the use cases derived is as follows:

Use-case 1 - The reading and backing up of the file *PrivateFile1* is captured in this use case. This use case aims to test whether the proposed reconstruction algorithm is able to reconstruct the propaga-

tion of data across different processes and over the network.

Use-case 2 - This use case shows the malicious modifications made to a system critical file, `/root/.ssh/authorized_keys`. It demonstrates file modifications by processes and shows the modified data being propagated to other processes.

Use-case 3 - The injection of the malicious shell code, `red.php`, through a running application by one of the red team is captured in this use case. The use cases tests if the algorithm is able to reconstruct sequence of events that describe the life cycle of a piece of data, from creation, access, modification to deletion.

Use-case 4 - The final use case shows the propagation of the malicious shell code injected in *use-case 3* through interactions with different applications. This use case tests the ability of the reconstruction algorithm to reconstruct the source and derivative provenance of the file `../test/red0.php`.

6.3 Reconstructing Data Provenance from Different Granularities

The first set of experiments investigates the outcome of the reconstruction when using log files of different granularity⁵. Recalling the NZCSC'15 dataset, described in Chapter 3, log files generated at the system and application layer of the operating system were collected. As such, the log files may differ in terms of the granularity of events and the observation scope of the logging mechanisms. To investigate the impact of different types of log files⁶ on the reconstruction, the NZCSC'15 log files are divided into the following three sets of data:

D1 - The first set of data consists of log files from the **system layer**. This includes the LAF, Sysdig and the network traffic log files.

⁵Granularity of a log file is determined based on the categorisation of log files using the system granularity view defined in Section 1.5.

⁶Log files are classified according to their granularity, as discussed in Section 1.5.

6.3 Reconstructing Data Provenance from Different Granularities

D2 - The second set of data comprises of log files from the **application layer**. Log files generated by the *Apache* web server applications form this set.

D3 - The third set of data is made up of log files from both the **system and application layers**. It is comprised of log files in both **D1** and **D2**.

6.3.1 D1 - System Log Files

Table 6.1 shows the precision and recall of the data provenance reconstructed for each use case using only system log files. Two observations can be made from the results. First, the reconstructed data provenance for all use cases exhibit low precision. This is especially so for *use-case 2, 3 and 4*, where *relevant*⁷ file objects were modified. Second, recall varies by a large margin across the different use cases. This is especially so with use cases involving the *Apache* process (i.e. *use-case 3 and 4*).

Low precision across the use cases indicates that the number of errors (i.e. false positives) in the reconstructed data provenance is consistently high. Analysis of the reconstructed output further revealed two insights regarding the erroneous provenance relations. First, shared system files and libraries made up the majority of erroneous entities in the graph. Second, the reconstructed data provenance manifest as a single connected graph. Erroneous entities are interconnected such that they eventually connect to at least one relevant entity. Figure 6.7 illustrates a captured screenshot, taken from a sample reconstructed data provenance, demonstrating the interconnectivity amongst entities.

Table 6.1: Results of reconstruction for each use case using the system only set

	Use-case 1	Use-case 2	Use-case 3	Use-case 4
Precision	0.2105	$5.1194e^{-5}$	$5.4317e^{-6}$	$2.9981e^{-5}$
Recall	0.8	1.0	0.5	0.3142

⁷An entity or provenance relation is considered *relevant* if it exists in the ground truth.

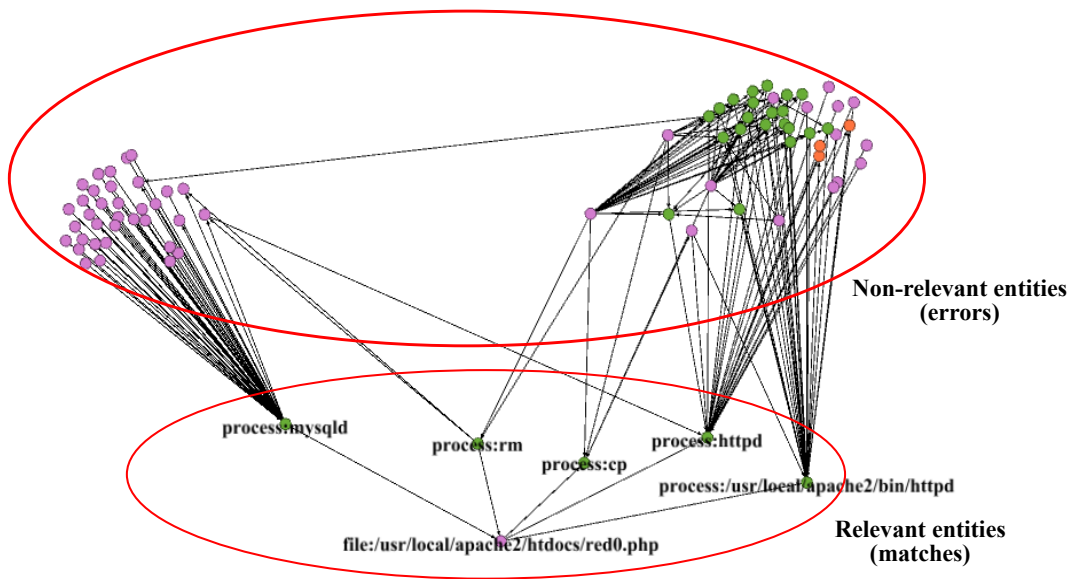


Figure 6.7: Sample screenshot showing how erroneous and relevant entities interconnects to form a single connected graph

As discussed in Section 5.2.3, reconstruction of paths is based on the principles of causality between events. The inclusion of a data flow relation (e.g. *read*, *write*, *transfer*) into the reconstructed data provenance will prompt the algorithm to search for relations that are causally related in the entity’s sorted list. Coupled with the two insights described above, it is deduced that the expanded search for causally related relations was the reason for the shared system files in the reconstructed output. This issue is referred to as the dependency explosion problem and is discussed in detail later in Section 6.4.

With respect to the second observation, our analysis of the reconstructed output also shows that there are some provenance relations that closely match the ground truth. These relations would overlap with the expected relations (e.g. those in the ground truth provenance) in terms of the time they happened. But they failed to satisfy the matching criteria due to a mismatch in one of the entity fields (e.g. source or target entity). As such, it is suspected that these relations are the expected relations but with errors. Table 6.2 lists three samples of such close match relations, taken from *use-case 1* and *4*. The bold entries highlight the mismatched entities in each pair.

6.3 Reconstructing Data Provenance from Different Granularities

Table 6.2: Examples of near matches between the ground truth and the reconstructed provenance

	Activity	Timestamp	Source Entity	Target Entity
GT	access	1442567040.0	process:httpd	file:/usr/local/apache2/htdocs/test/red0.php
RP	access	1442567045.0	process: /usr/local/ apache2/bin/ httpd	file:/usr/local/apache2/htdocs/test/red0.php
GT	access	1442567400.0	process:httpd	file:/usr/local/apache2/htdocs/test/red0.php
RP	access	1442567418.0	process: /usr/local/ apache2/bin/ httpd	file:/usr/local/apache2/htdocs/test/red0.php
GT	datatransfer	1442567640.0	process:/usr/sbin/sshd	host:192.168.120.50
RP	datatransfer	1442567695.0	process:/usr/sbin/sshd	host:192.168.120.51

Such mismatches can be traced back to errors in the parameters⁸ logged by the logging mechanisms. In his work on documenting problems with intercepting system calls, Garfinkel [2003] discussed how those problems can lead to the kind of erroneous parameters we observed in the log files.

The severity of the errors vary from minor inconsistencies that can be resolved through visual inspection to the need for other sources of information for resolution to be possible. For example, the kernel logging mechanisms used in the NZCSC’15 inconsistently label the web service process, *Apache*, either by its process name ‘*httpd*’ or by its fully qualified path name ‘*/usr/local/apache2/bin/httpd*’. This is shown by the first two pairs of mismatch in Table 6.2. Although automatic comparison of relations would flag these minor inconsistencies as errors, a visual inspection would reveal that the two labels are referring to the same process. Based on the observations gathered from such visual inspections,

⁸The term ‘parameter’ is defined in Section 2.1.

Table 6.3: Improvement in recall values after adapting the evaluation to the inconsistencies observed in the log files

	Use-case 1	Use-case 2	Use-case 3	Use-case 4
Recall before adaptation	0.8	1.0	0.5	0.3142
Recall after adaptation	0.8	1.0	1.0	1.0

the entity labels can be manually canonicalised to produce a semantically accurate evaluation of the reconstructed data provenance.

However, to automatically identify and canonicalise entity labels is not a straightforward task. For example, in their discussion on matching heterogeneous events between different log files, Zhu et al. [2014] pointed out similarity based approaches for automatically canonicalising labels would fail if the difference between the labels is “opaque”. Approaches such as typographic similarity (e.g. string cosine similarity) and linguistic similarity (e.g. using dictionary of ontology) would not work if the labels are encoded or represented differently by the different logging mechanisms.

Table 6.3 compares the recall before and after the adaptation. Entities that are inconsistently labelled can now be correctly identified under the adapted comparison scheme. The modification on the comparison resulted in a stark improvement in recall for *use-case 3* and *4*. The recall for those two use cases improved by at least two fold, showing the ground truth to be completely reconstructed.

On the other hand, errors in the parameters such as a mislabelled entity cannot be resolved without additional information (e.g. log files from other layers). One such example is the third mismatched pair shown in Table 6.2. From the ground truth, a provenance relation that describes a communication between the process *sshd* and a remote host *192.168.120.50* is expected. However, events captured at the system layer associated the network socket to the local host address instead. This results in a provenance relation that showed *sshd* communicating

6.3 Reconstructing Data Provenance from Different Granularities

Table 6.4: Results of the reconstruction using only application layer log files

	Use-case 1	Use-case 2	Use-case 3	Use-case 4
Precision	-	-	1	1
Recall	-	-	0.5	0.3428

with the local host, *192.168.120.51*⁹. Even though manual analysis could reveal the error¹⁰, it is difficult to assert the correct identity of the entity with whom *sshd* is communicating without additional sources of information. As Garfinkel [2003] pointed out in their discussion on their experiences with system level logging, errors in parameter logging is common due to various reasons, such as complex data structures used in the system and race conditions. We discuss this further in Section 6.3.3.

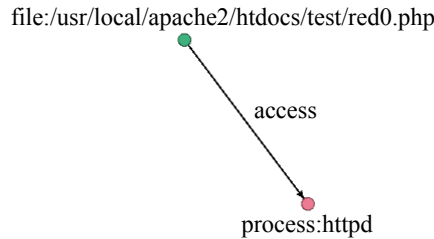
From this experiment, we can observe reconstructing using system log files can lead to data provenance with high recall but low precision. The results with improved recall, shown in Table 6.3, demonstrated that ground truth for most of the use cases were completely reconstructed. This is attributed to the shared nature of the system devices, such as the kernel and file system. Since applications running on the system all uses the same set of devices, events logged by the system layer logging mechanisms will be a mix of activities from these applications. As a result, the reconstruction algorithm has access to all necessary provenance relations to fully reconstruct the ground truth. However, because of the inability to accurately differentiate which pair of causally related provenance relations are relevant, data provenance reconstructed using only system log files suffers from large number of errors (i.e. low precision). In addition, using only log files from the system layer makes it difficult to resolve inconsistencies produced by the logging mechanisms.

⁹Association of IP addresses to hosts is based on knowledge of the network structure of the NZCSC'15 competition.

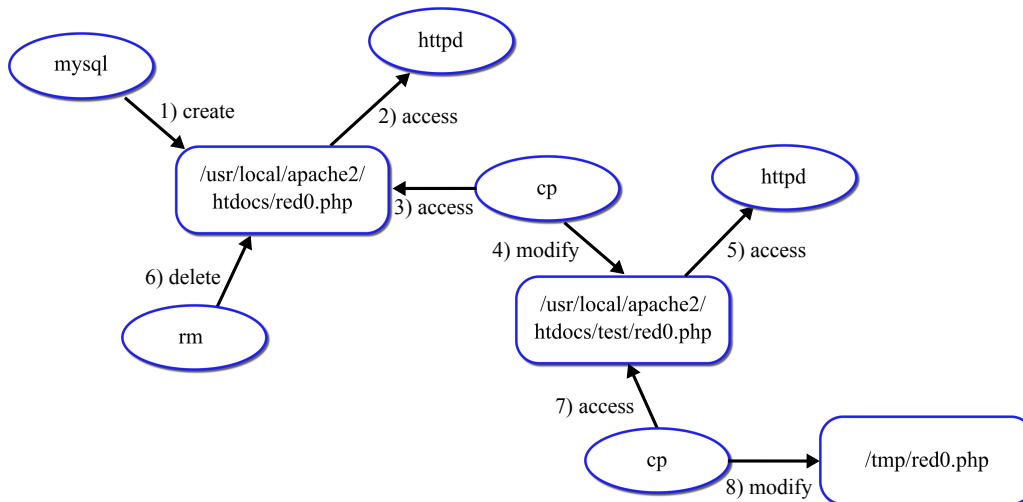
¹⁰A host computer sending data over the network to itself would appear to be abnormal.

6.3.2 D2 - Application Log Files

Table 6.4 shows the results of reconstructing the data provenance using only application layer log files. Note that *use-case 1* and *2* are not considered in this experiment as there were no application log files generated for applications used in those two use cases.



(a) Reconstructed data provenance for *use-case 4*



(b) Ground truth data provenance for *use-case 4*

Figure 6.8: Comparing the data provenance reconstructed from dataset **D2** to the ground truth

Recall values for the applicable use cases indicated that the ground truth could not be fully reconstructed from application log files alone. Figure 6.8 shows the reconstructed and ground truth data provenance for *use-case 4*¹¹. Figure 6.8a shows only the *access* from the *httpd* process to the file of interest, *../test/red0.php*, could be reconstructed.

¹¹Note that the graphs illustrated display only distinct activities between entities (e.g. multiple *access* to the file object, *../test/red0.php*, by the process *httpd* is simplified to a single *access* between the two entities).

6.3 Reconstructing Data Provenance from Different Granularities

Provenance relations describing interactions between `../htdocs/red0.php` and applications such as `mysql`, `rm` and `cp` were absent. The situation can be attributed to the fact that events from the missing applications took place outside the observation scope of the *Apache* logging mechanism. Hence they were not included in the *Apache* log files. Since, no log files were generated by the missing applications¹², the resulting set of provenance relations is incomplete. Likewise, the algorithm could not establish the relationship between `../htdocs/red0.php` and `../htdocs/test/red0.php` as the main provenance relations linking the two objects through `cp` are missing from the modelled set.

Even though the dataset used in this experiment is composed of log files from a single application type, the limitation surrounding the observation scope of logging mechanisms applies even if log files from multiple applications are used. The algorithm would only likely be able to reconstruct the data provenance if the scopes of the log files contiguously cover the ground truth (i.e. all relevant events are captured across the different log files). However, once contiguity is broken, such as deriving a new file object via applications that do not generate log files, the completeness of the reconstructed data provenance will be affected. The derivation relationship between `../htdocs/red0.php` and `../htdocs/test/red0.php` through `cp` in *use-case 4* is an example of how a break in the contiguity of the scope can affect the completeness of the reconstructed data provenance.

6.3.3 D3 - System and Application Log Files

Table 6.5 lists the results for using both system and application layer log files in the reconstruction. In comparison, the results are similar to those shown in Table 6.3, where the dataset **D1** is used but with adaptations made to the implementation of the comparison. Reconstructed data provenance with high recall is the common outcome for both ex-

¹²Either because operating system native applications such as `rm` and `cp` do not generate log files or the log files generated only showed error events, such as the case for `mysql`.

periments. However, unlike using only system log files, the experiment here does not require modification to the comparison for resolving the errors in the system log files. Instead, provenance relations modelled from the application log files acted as redundancy, providing the means to “self-correct” the inconsistencies.

Table 6.5: Results of reconstruction different use-cases using both system and application layer log files

	Use-case 1	Use-case 2	Use-case 3	Use-case 4
Precision	0.2105	5.1194e ⁻⁵	1.0861e ⁻⁵	9.538e ⁻⁵
Recall	0.8	1	1	1

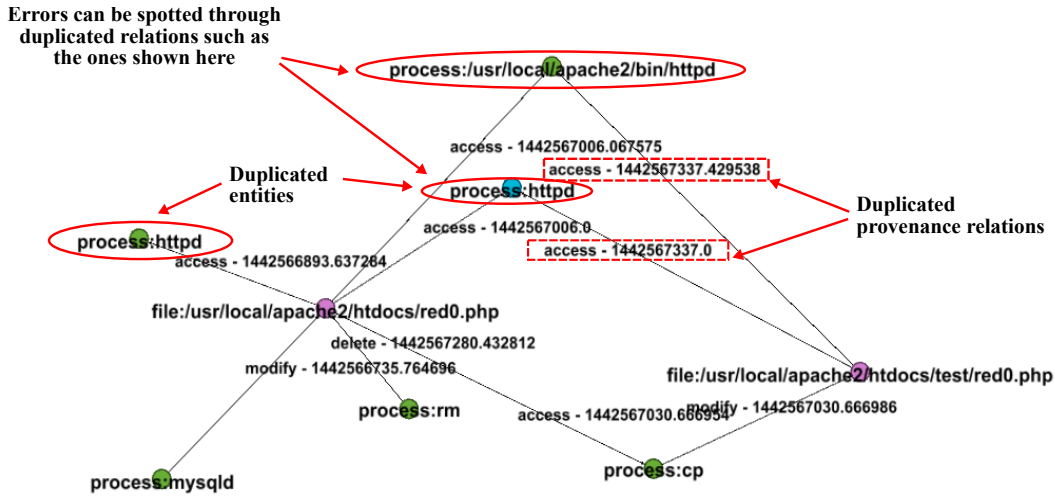


Figure 6.9: Duplicated provenance relations in data provenance reconstructed using dataset **D3**

In situations where both application and system layer logging mechanisms are logging events concurrently, the same event will likely be logged twice. Hence, duplicated provenance relations will result from the modelling of log files. In situations where an error such as the mislabelling of entities described in Section 6.3.1 occurs, the duplicated relations will instead appear as pair of closely matched relations. This is because most of the information captured in the two relations will be overlapping (e.g. activity observed, timestamp, entities involved), except for the mislabelled entity. In the absence of ground truth, these closely

6.3 Reconstructing Data Provenance from Different Granularities

matched relations can highlight the error to users analysing the provenance graph, as demonstrated in Figure 6.9.

However, using log files that overlap in terms of events observed will also lead to duplicated entities in the reconstructed provenance. This is because different logging mechanisms may assign different identifiers to the same entity. For example, a process would be identified by its process name in the application layer and its process ID in the system layer. This causes the provenance model to treat both as different entities. Graph entity resolution techniques such as those described by Bhattacharya and Getoor [2005] can be used for removing such duplicates.

Experiment using dataset **D3** also showed errors resulting from issues in the system logging mechanisms, such as those discussed by Garfinkel [2003], may be partially resolved using log files obtained in the application or higher layers. This highlights the importance of using multiple sources of information when reconstructing the data provenance.

From the experiments discussed thus far, we can conclude that the observation scope of the logging mechanism is crucial to reconstructing a complete data provenance. Obtaining a set of application layer log files that have an unified scope sufficient enough to cover all required events is difficult. This is so as not all applications generate log files. The lack of application log files for *use-case 1* and *2* in dataset **D2** substantiate this claim. In contrast, system layer log files contain events that describe activities across all applications running on the system. This is attributed to the shared nature of system devices. As a result, data provenance with high or perfect recall can be reconstructed using system layer log files. However, the reconstructed data provenance is difficult to analyse as the output contains large number of redundant provenance relations (e.g. errors). These errors obfuscate the relations that matter (e.g. ground truth) and are caused by the inability to differentiate relevant provenance relations from the set of causally related relations. In the next section, we investigate approaches for reducing errors mentioned in Section 6.3.1.

6.4 Dependency Explosion

This section first describes the dependency explosion problem associated with the use of system log files for provenance reconstruction, as highlighted in Section 6.3.1. Following which, possible approaches for reducing effects of the problem are discussed.

6.4.1 Understanding the Dependency Explosion Problem

Throughout the execution lifetime of a process, it may load and utilise different files and objects, as dependencies, for access to external functionalities or as part of its programmed execution. The dependency explosion problem occurs when the reconstruction algorithm mistakenly adds such dependencies into the provenance graph being reconstructed. In certain cases, dependencies can also be shared by different applications. Such shared behaviour of dependencies further worsen the effects of the problem.

During reconstruction, because of the inability to identify exactly the conjoining provenance relations from the set of causally related relations, some of the dependencies appear in the reconstructed output as errors. Figure 6.10 illustrates an example of how this can happen. Upon discovering the *write* relation to a file object, the algorithm would search for provenance relations that could influence the *write*. Since dependencies such as library files are mostly loaded (e.g. treated as *access* or *read*) during the early phase of a process (e.g. initialisation phase), relations that associate dependencies to the process would satisfy the causal dependency requirement and be added to the reconstructed data provenance.

However, in cases where the dependencies are used by different processes (e.g. a shared system resource), the search for relevant provenance relations may result in an explosion of dependencies being added to the reconstructed data provenance.

With reference to Figure 6.11, if a dependency is regularly accessed and modified by different processes, the search for relevant relations

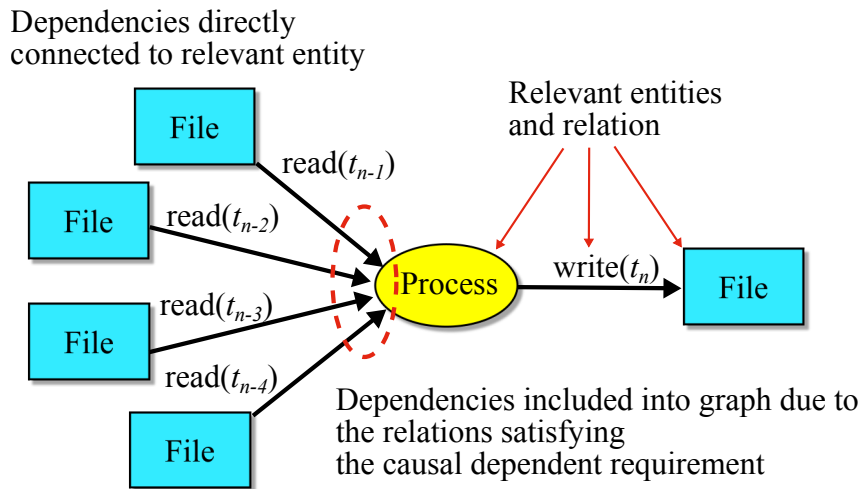


Figure 6.10: Dependencies directly connected to relevant process nodes in the reconstructed data provenance graph

may be chained along the different entities added to the reconstructed data provenance. Figure 6.12 captures a sample screenshot showing such a chained explosion of dependencies in the reconstructed output. The end result is the large number of errors observed in the experiment with using only system log files.

One approach to reduce the number of errors is by pruning dependencies from the set of modelled provenance relations before reconstruction. In the following subsections, we discuss approaches that can be used to identify and prune dependencies.

6.4.2 Blacklisting Dependencies

A naive approach for identifying and pruning dependencies is blacklisting. From the Linux system man pages [Faith et al. 2016], the following list of directory prefixes are put together as the blacklist. This list of directories represents known structures of the file system where shared system resources are stored. Note that application specific dependencies, such as those stored under the application’s home directories, are not included in the list.

Figure 6.13 compares the precision of the reconstructed data provenance between using system log files with no pruning done to those

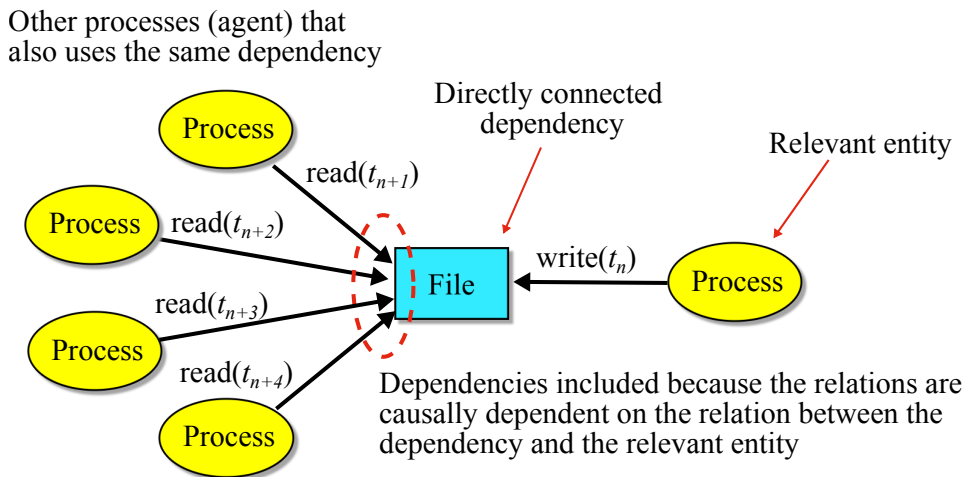


Figure 6.11: Indirect dependencies explosion when reconstructing derivation provenance

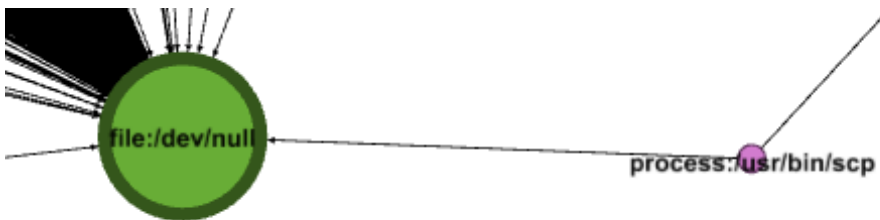


Figure 6.12: Chained dependency explosion through shared system resources

pruned using the blacklist approach. Note that the Y-axis is in log scale, with the best precision value, 1, located at the bottom. As such, a lower bar indicates better precision. By simply removing the dependencies, precision is improved significantly by approximately a factor of 100 in *use-case 2, 3 and 4*.

However, blindly pruning a fixed list of dependencies may result in the loss of relevant provenance relations. As shown in Table 6.6, completeness of the reconstructed data provenance for *use-case 4* is observed to suffer from pruning. Attackers are known to hide malware in directories used for storing dependencies. In other situations such as a binary planting attack [OWASP 2013], malicious users or hackers are known to replace dependencies or executables on the system with malicious codes of their own. As a result, provenance relations describing the activities

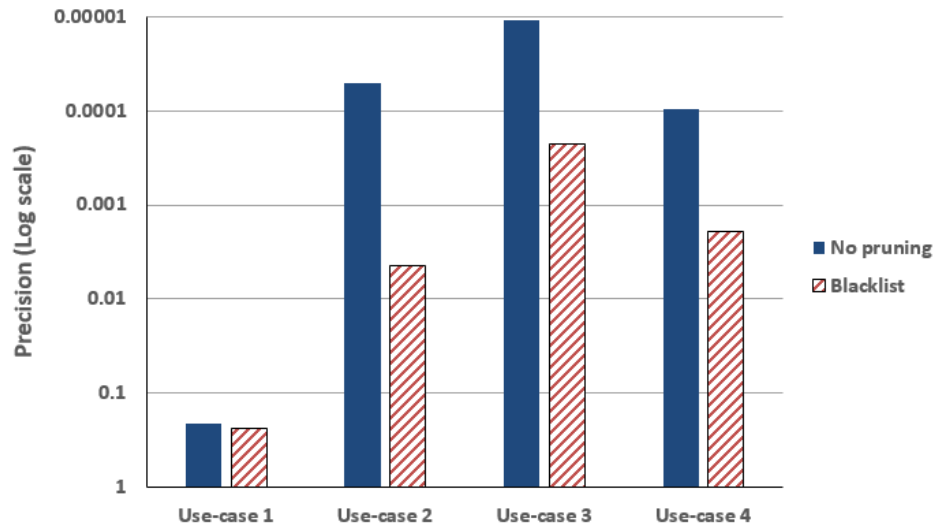


Figure 6.13: Comparing precision of the reconstructed output between using system log files with no pruning and blacklist pruning (Y-axis is in log scale)

Table 6.6: Results of reconstruction using blacklist-pruned system log files

	Use-case 1	Use-case 2	Use-case 3	Use-case 4
Precision	0.235	4.36e ⁻³	2.2465e ⁻⁴	1.8955e ⁻³
Recall	0.8	1	1	0.9714

of these malicious codes would be lost to the reconstruction algorithm as the corresponding entities would have been pruned away. In such situations, impact to the completeness of the reconstructed data provenance would affect its usage.

6.4.3 Process-access Based

A second approach for identifying dependencies to be pruned is by analysing each file's usage pattern and number of unique processes accessing it (e.g. process-access value). The premises are as follow:

1. Dependencies are likely to be files used by a large number of processes.
2. Dependencies that only have *read* relations can be pruned away

Directory `"/dev/", "/etc/", "/proc/", "/usr/lib/", "/usr/include/",`
list: `"/usr/share/", "/lib/", "/var/", "/tmp/", "/sys/"`

without affecting much of the ground truth as no new data is being propagated through these dependencies (e.g. pruning these dependencies will not result in loss of sub-graphs).

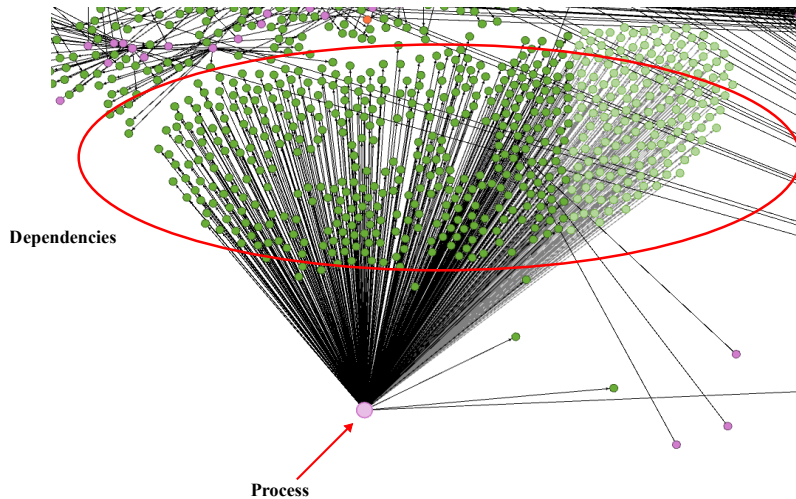


Figure 6.14: Explosion of dependencies connected to a process—the dependency explosion problem may lead to file dependencies used for the initialisation and execution of a process to be included into the data provenance

Based on the premises, the approach targets read-only dependencies. Read-only file objects (e.g. file objects with only *read* or *access* relations) are classed as dependencies to be pruned if its process-access value exceeds a pre-defined threshold. The goal is to avoid clusters of read-only dependencies such as those shown in Figure 6.14 while still allowing other activities pertaining to processes to be reconstructed. The decision to keep dependencies with *write* relations associated is mainly due to *write* representing a propagation of data to new entities. Removing such dependencies would risk losing important segments of the data provenance graph that may help identify abnormalities such as a disguised malware. Hence in this experiment, we simulate retaining dependencies with *write* relations.

File objects identified as dependencies are pruned before the reconstruction process so as to avoid incurring redundant computation cost

Table 6.7: Summary of results for process-access based pruning**(a)** Use-case 1

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
0.2105	0.8	1539	54.38%
0.2352	0.8	830	29.32%

(b) Use-case 2

Precision	Recall	No. of Process-access	Process-access to total unique processes ratio
$5.12e^{-5}$	1	2436	41.51%
$5.576e^{-3}$	1	7	0.11%

(c) Use-case 3

Precision	Recall	No. of Process-access	Process-access to total unique processes ratio
$1.09e^{-5}$	1	4367	54.26%
$1.04e^{-4}$	1	10	0.12%

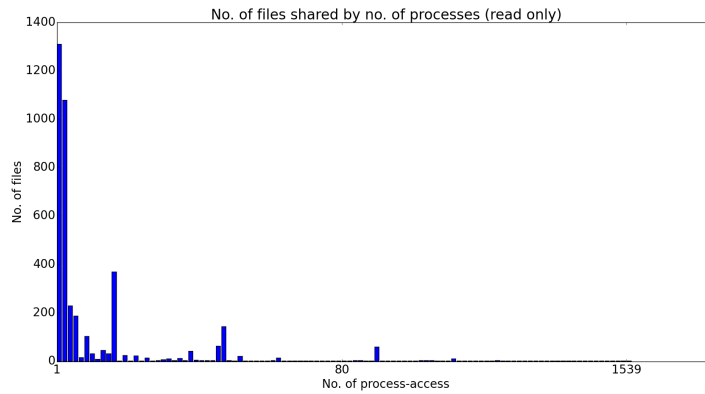
(d) Use-case 4

Precision	Recall	No. of Process-access	Process-access to total unique processes ratio
$9.54e^{-5}$	1	4367	54.26%
$1.474e^{-4}$	1	41	0.509%

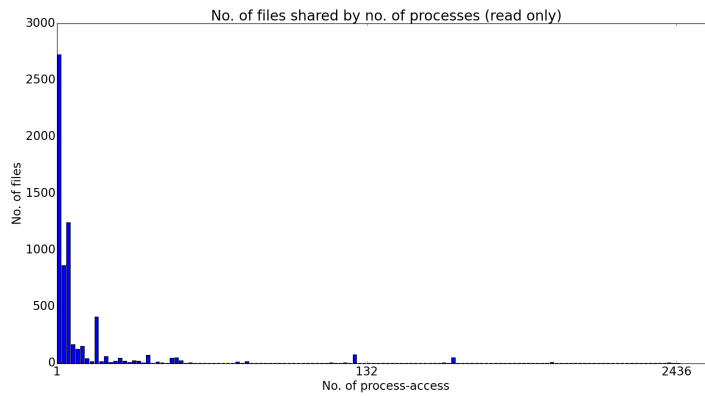
when searching for the next conjoining relation. However, the main challenge is defining a threshold value for the classification of file objects. Objects identified should help reduce the number of errors in the reconstructed data provenance while not compromising completeness. As such, finding the correct threshold is crucial. Figure 6.15 tabulates file objects having the same process-access value into histograms for each system log file used for the reconstruction of each use case. Note that *use-case 3* and *4* both use the same system log file, hence share the same histogram.

The x-axes show the process-access value for each group of files while y-axes count the number of files in each group. From the right end of each histogram, we can observe that only a small number of files, with respect to the total number of unique file objects in each system log file,

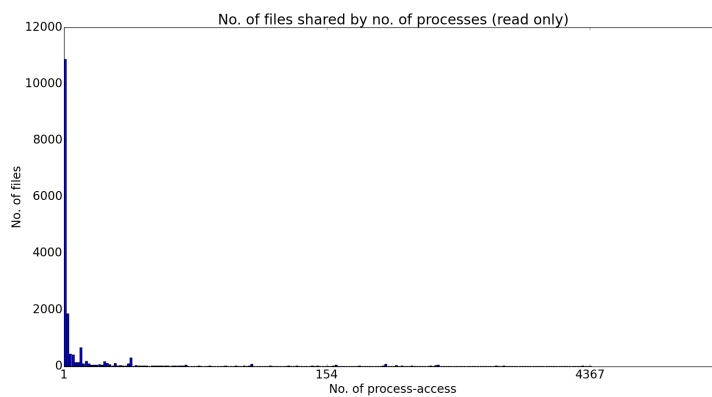
Chapter 6 Experiments



(a) Use-case 1



(b) Use-case 2



(c) Use-case 3 and 4

Figure 6.15: Histograms showing the number of dependencies sharing the same number of processes accessing them for the different system log files

fits the first premise.

To determine the threshold, file objects are iteratively added to a list of files to be pruned starting from files with the highest number of process-access (i.e. right end of histogram). This process iterates until recall is affected. Table 6.7 compares the results from the first iteration of the pruning with the first iteration that yields the best precision without compromising recall. Results for all iterations are listed in Appendix C for reference. The last column puts into perspective the percentage of total unique processes required to be sharing a file for it to be considered for pruning if the threshold value is set at the corresponding process-access value.

From the percentage of total processes and the process-access value, we can observe that the process-access value that provides the best precision varies across different use cases. One reason is because errors incurred in each use case may be caused by different dependencies. Since the file objects are ranked according to the process-access values, the dependencies that caused the errors may be pruned only at a later iteration. Hence, the use of process-access value as threshold may produce results that may differ based on the situation rendering this approach unreliable. Furthermore, determining the threshold becomes more difficult without the ground truth since it is required for calculating recall.

Figure 6.16 compares the precision of the reconstructed output using process-access based pruning against blacklist pruning and no pruning done. While process-access based pruning performs slightly better in *use-case 2*, the precision is significantly worse than blacklist pruning for *use-cases 3 and 4*. Further analysis of the reconstructed data provenance showed that there were many dependency files that were used only by a single process, such as temporary files. This observation refuted the first premise, listed in the beginning of this subsection, which hypothesize that dependencies are files used by a large number of processes. It also indicates that identifying dependencies based on the number of processes accessing the file objects cannot be generalised for all cases.

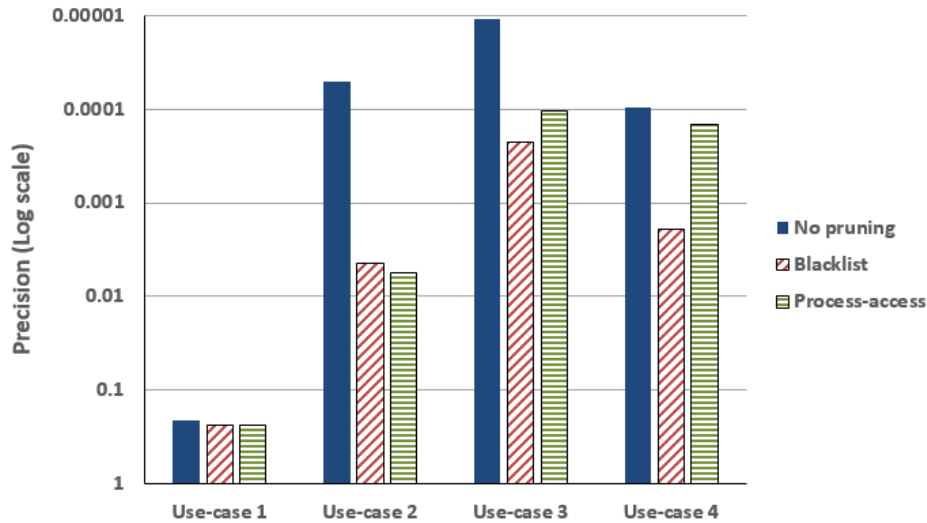


Figure 6.16: Comparing precision of the reconstructed output for process-access based pruning with blacklist and no pruning (Y-axis is in log scale)

6.4.4 Clustering

Instead of determining file objects to be pruned based on the number of processes accessing the object, the third approach uses clustering to identify groups of files used in a similar manner. Clusters can then be selected, according to usage pattern, for pruning. The following file object attributes are considered for the clustering:

- **Number of unique processes** - Determines how many unique processes uses the file object.
- **Read-to-Write ratio** - Defines a file object’s usage pattern. It ranges from 1 to -1, with 1 being read-only and -1 being write-only.
- **Interaction Mean** - Statistically denotes how frequent each file object is being used by a single process.
- **Interaction Variance** - Statistically denotes how each process uses the file object differently.

K-means is used as the clustering algorithm. The implementation, from the Python based machine learning package *scikit-learn* [scikit 2010], used in our experiment utilises *Euclidean distance* as the distance mea-

sure. Attributes for each file object are consolidated into vector form and normalised before clustering. The elbow-method [Ketchen and Shook 1996] is used to estimate the number of clusters, k .

The elbow method is based on the premise that as more clusters are added, the data could be split into distinct groups (e.g. better modelling of the data), until a point where adding more clusters would result in dissimilar clusters. By visualising the percentage of variance against the number of clusters, one can infer the value of k that best models the data by observing where the graph starts to flatten. Percentage of variance is computed as follows:

$$\text{Percentage of variance} = \frac{\text{between-cluster-variance}}{\text{total variance}} \quad (6.3)$$

where *between-cluster-variance* is calculated as:

$$\text{Between-cluster-variance} = \sum_{j=1}^k n_j (\bar{x}_j - \bar{x})^2 \quad (6.4)$$

and total variance is calculated as:

$$\text{Total variance} = \sum_{j=1}^k \sum_{i=1}^{n_j} (x_{ij} - \bar{x})^2 \quad (6.5)$$

where k is the number of clusters, n_j is the number of file objects in cluster j , \bar{x} is the overall mean and \bar{x}_j is the mean for cluster j . To estimate k , the file object vector is computed and clustered using k-means, with k varying from 2 to 10. The percentage of variance is calculated for each k and plotted. To obtain a general value for k , such that it can be applied generally to the system log files in the NZCSC'15 dataset, we apply the described process on each system log file in the dataset.

Figure 6.17 illustrates the plot for the system log file used for use-case 2. Results for other system log files are not shown here as they are similar. Instead, they are listed in Appendix D for reference. It can be observed that gradient of the curve starts decreasing when k is 5. As a result, we estimate k to be 5.

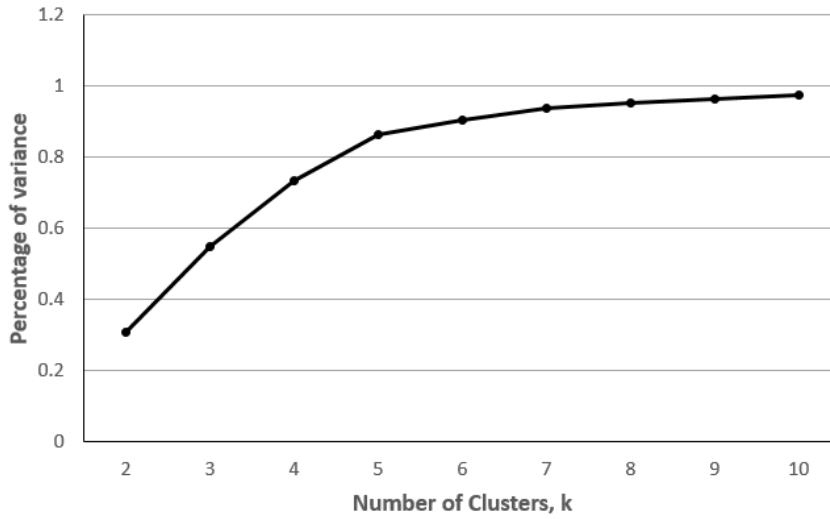


Figure 6.17: Percentage of variance measured for different values of k for system log file used in the reconstruction of *use-case 2*

Table 6.8: Attributes of the best-performing cluster for each use case

	Use-case 1	Use-case 2	Use-case 3	Use-case 4
No. of processes	1.47	457.7	8	8
R/W ratio	-0.868	0.991	0.757	0.757
Mean	11.91	2.519	3418.48	3418.48
Variance	7.88	0.109	22255786	22255786

To evaluate the number of errors that can be reduced using the clusters produced, each use case is reconstructed five times, once for each cluster produced. In each iteration, files from one of the five clusters are pruned from the corresponding system log file before running the reconstruction algorithm. Summary of the precision, from pruning each cluster, is shown in Figure 6.18. Note, missing bars for each use case indicates the reconstruction failed to reconstruct anything as the required files were pruned away (e.g. zero recall and precision). The arrows in the figure mark the best performing clusters.

Table 6.8 shows the attributes defining each of the best performing clusters. It shows the usage patterns for each cluster of files to be different from one another. This is attributed to the reason mentioned previously with the process-access based pruning approach; errors in each of the use case may be caused by different dependencies. However,

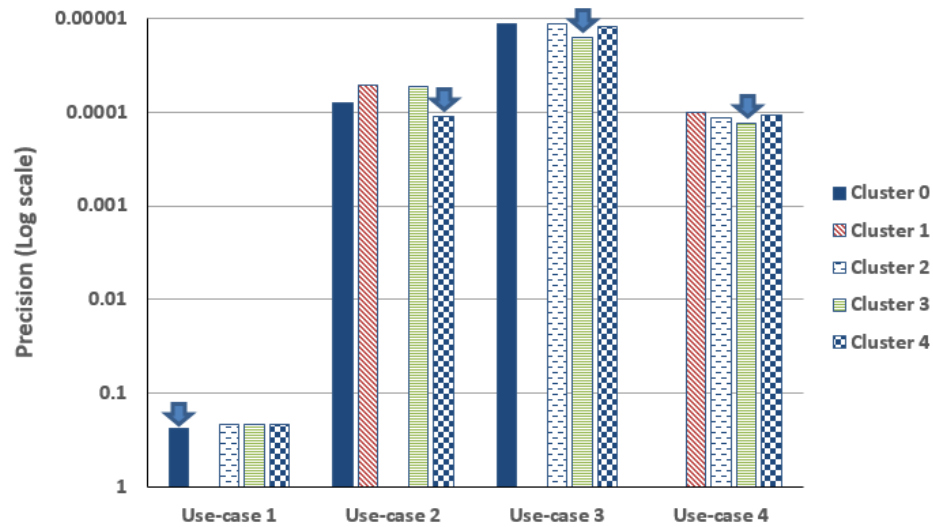


Figure 6.18: Summary of precision for pruning each cluster from system log file, for each use case (Y-axis in log scale)

Table 6.9: Attributes of clusters that resulted in zero recall (no ground truth)

	Use-case 1	Use-case 2	Use-case 3	Use-case 4
No. of processes	8.534	7.452	1.204	7.435
R/W ratio	0.989	0.988	-0.962	0.99
Mean	6.252	6.437	9.342	6.671
Variance	3032.2	3791.2	54.62	1550.36

in comparison with pruning using a threshold value, clustering allows files used in the same manner to be selected in one iteration. Clustering files according to their usage patterns also prevents abnormal files, such as dependencies being replaced in a binary planting attack, to be pruned. When dependencies behave abnormally, the behavioural change will cause the clustering algorithm to cluster it with other more similar behaving files.

Although the usage patterns shown in Table 6.8 hint that dependencies cannot be described by a fixed set of usage patterns, it is interesting to note clusters that result in an impact on recall (e.g. zero recall) showed certain consistent patterns among them. Table 6.9 summarises the usage patterns of files for clusters which resulted in the lost of ground truth when pruned from the system log files (e.g. recall is zero). These files

generally have low mean and high variance. This observation indicates the usage patterns for the same file differs between processes and that the interaction between each process and the corresponding file is not frequent (i.e. in comparison to clusters which exhibits over three thousand interactions).

Building on this observation, the reconstruction is executed again. However, this time around, instead of pruning one cluster from the system log files, clusters which do not impact the recall in Figure 6.18 are all pruned instead. Figure 6.19 compares the outcome of the reconstruction with other approaches discussed thus far for removing dependencies. Although the clustering approach generally performs better than the process-access approach, it performed the worst for *use-case 2*. However, the clustering approach provides more flexibility in choosing files to prune compared to the process-access approach as groups of files can be quickly selected based on patterns in the clustering attributes. The clustering approach can also pick up on changes in file behaviour, such as the case of a binary replacement attack, since clustering is based on file usage patterns. In certain cases, such as *use-case 3*, it even performed better than the blacklisting approach. This is because the clustering approach was able to pick up some of the application specific dependencies that are not included in the blacklist.

Although the clustering shows some promise for separating relevant files objects from the set of file objects in the modelled set of provenance relations, the observations drawn here are not representative of dependencies. This is mainly because the NZCSC'15 dataset only captures a specific type of user interaction with the underlying system¹³. Datasets that describe different types of system behaviour (e.g. system log files from data centres, file servers or other types of system) would be required for a more extensive study.

¹³Because of the nature of the competition, the participants' interaction with the systems are influenced and targeted at achieving a specific set of goals, such as monitoring the system for attacks. As a result, the dataset is not well suited for studies aimed at understanding expected usage patterns of files.

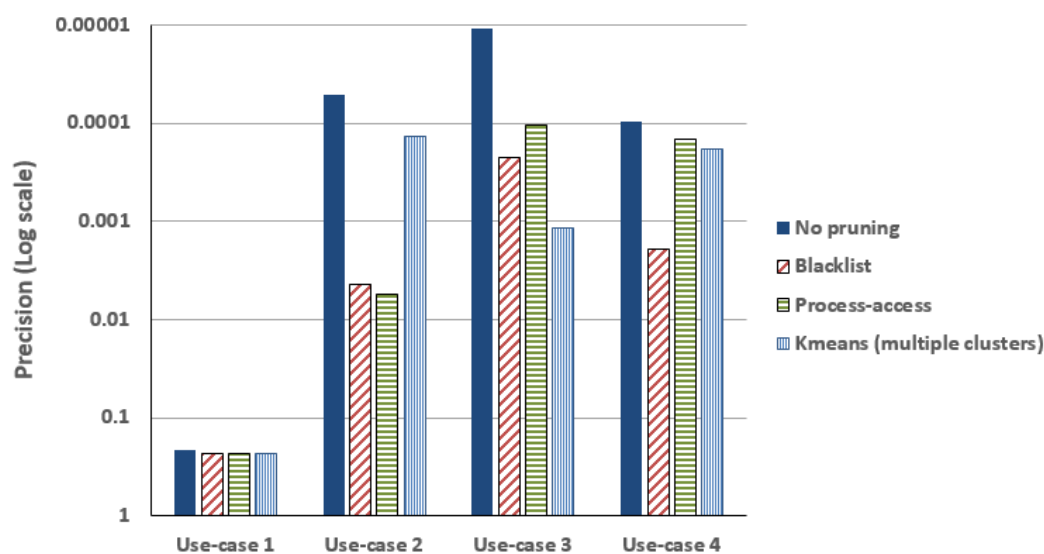


Figure 6.19: Comparing precision of reconstructed data provenance for clustering based pruning with other discussed approaches (Y-axis in log scale)

6.5 Summary

In this chapter, a methodology for evaluating reconstructed data provenance against the ground truth provenance is presented. Using the proposed methodology, data provenance reconstructed using different sets of log files from the NZCSC'15 dataset are evaluated.

Results of the experiments showed data provenance with high recall, with respect to the ground truth, can be reconstructed from system log files. The experiment that combined the use of application and system layer log files has also shown relations modelled from the application log files can resolve inconsistencies generated during the logging of system events.

Having said that, provenance reconstructed using system log files were observed to have low precision (i.e. contains a large number of redundant relations). The poor precision is attributed mainly to the inability to accurately identify relevant relations from the set of causally related relations. Low precision may induce ambiguities when attempting to deduce the derivation history of the data from the provenance. While having low precision provenance is undesirable in critical scenarios (e.g. data audit

or system security), low precise provenance may still provide useful insights to figuring out information such as potential processes that holds the data and how they obtained a copy of the data. Having said that, we do not discuss the impact and possible applications of data provenance with low precision as exploring the applications of the reconstructed data provenance is out of the scope of this thesis.

To reduce the number of errors produced from reconstructing using system log files, several approaches for identifying and pruning dependencies are discussed. In most cases, blacklist-based pruning was able to improve precision by a factor of ten to a factor of a hundred. However, statically identifying file objects as dependencies and pruning them may result in losing important segments of the provenance graph. These segments may be crucial for identifying abnormalities in the system. Identifying file objects for pruning based on the number of processes accessing them faces the issue of selecting a suitable threshold for the classification. In addition, observation drawn from the results also showed that dependencies cannot be identified from just the number of processes accessing the file object. Clustering provided a more dynamic approach as it allows a set of file attributes to be considered. However, further studies on the attributes used will be required as the experimental results showed that current attributes used cannot definitively identify dependencies.

Through the experiments discussed in this chapter, we can observe data provenance with high recall can be reconstructed using system layer log files. However, the resulting provenance suffers from low precision due to the dependency explosion problem. Although having low precision in the reconstructed provenance may affect its usage, the applications of the reconstructed provenance is not the focus of this thesis. Hence we do not discuss the implications of using reconstructed data provenance with low precision. It should also be noted that results of the experiments and the conclusion is based on the event causality based algorithm proposed for the reconstruction.

Chapter 7

Conclusions and Future Work

7.1 Conclusion

Motivated by the issues surrounding actively collecting data provenance, this thesis investigates the reconstruction of data provenance from log files obtained off a computer system. Log events are first modelled into pair-wise provenance relations using the proposed multi-layered provenance model, DFPM. An algorithm, based on the principles of causality between events, then reconstructs the provenance for a given piece of data from the modelled relations.

A methodology that evaluates the correctness and completeness of the reconstructed output, using the metrics precision and recall respectively, is proposed. The methodology helps address the second part of the thesis question; how the reconstructed data provenance compares to its known derivation history. Results from experimenting with log files of different granularity revealed high recall provenance can be reconstructed from system log files. This is attributed to the system wide scope of system layer devices.

Experiments also showed the use of application log files in conjunction with system log files can help resolve inconsistencies that are generated during logging in the system layer. Doing so improved the recall of the reconstructed data provenance. Concurrent use of log files from different layers of the system can also help to mitigate the effects of not being able to automatically canonicalising entity labels. In three out of the four use-cases used in the experiments, the ground truth was shown to have

Chapter 7 Conclusion and Future Work

been completely reconstructed without having to adjust the comparison to compensate for the inconsistencies.

The experimental results also showed a large number of redundant relations in the provenance reconstructed using system log files. These relations have the adverse effect of reducing the precision of the reconstructed output, obfuscating the ground truth in the provenance. Resulting from the dependency explosion problem, these relations are mostly interactions with shared library files or program dependencies. Analysis showed the dependency explosion problem is attributed to an inability to accurately identify relations relevant to the data from the set of causally related relations. Although low precision in the reconstructed output may impact the usability of the data provenance, the usefulness of low precise reconstructed data provenance is not discussed as the discussing the application of the reconstructed data provenance is out of the scope of this thesis.

Three approaches for identifying and pruning dependencies are considered for improving the precision of the provenance reconstructed from system log files. The blacklist approach statically identifies file objects as dependencies and prunes them from the modelled relations. Although precision can be improved by a factor of a hundred in some use-cases, the approach can affect the detection of abnormal objects in the system. While the other two approaches do not perform as well as the blacklist approach, they factor in the behaviour of the file objects, hence are less susceptible to abnormalities such as malicious activities that involves masking potential malware as file objects. Identifying and pruning of file objects, based on the number of processes accessing them, has the issue of selecting a suitable threshold for classifying file objects. Also, observations drawn from the analysis of the results showed that dependencies cannot be identified solely from the number of processes accessing the file object. Clustering file objects based on their properties was able to perform equally or better than process-access approach in most use-cases. However, analysis on the clusters showed the attributes used are not able to definitively link dependencies to a specific pattern in the attributes.

Overall, the blacklist and process-access approaches for identifying and pruning dependencies showed promising results in terms of improving the precision. However, it is not easy to generalise the two approaches due to the dynamic nature of how processes interact with dependencies. While the clustering approach for identifying dependencies showed promising results, due to the lack of a comprehensive dataset, further studies on identifying attributes that can be used to identify dependencies would be required.

In conclusion, although data provenance with high recall can be reconstructed from system log files, the output suffers from a large number of redundant relations caused by the dependency explosion problem. While approaches to prune dependencies have significantly improved the precision, further studies on identifying dependencies is required for a more accurate classification. It should also be noted that this conclusion is drawn based on reconstructing data provenance using an event causality based approach. The reconstruction workflow and evaluation methodology established in this research provides the platform for future research on provenance reconstruction.

7.1.1 Summary of Contributions

Besides addressing the question of data provenance reconstruction, this thesis makes the following contributions to the area of provenance:

- Dataset for Provenance Reconstruction Research

Having a suitable dataset that can be used to evaluate solutions for provenance reconstruction is crucial. However, a review of existing literature and publicly available datasets showed the lack of such a dataset. In this thesis, a set of requirements is defined to guide the selection of datasets that can be used for provenance reconstruction research. Leveraging on the NZCSC'15 competition, held at the University of Waikato, a dataset that comprises a set of log files and transcripts, that describe participants' screen activities, is constructed. The transcripts allow the derivation of ground truth provenance that can be used to evaluate provenance reconstructed from the log files.

- Data Flow Provenance Model (DFPM)

Most existing provenance models operate with the assumption that provenance is of single granularity. However, log events captured at different granularity layers of a computer system can differ in terms of level of detail, even if they are describing the same observed event. As a result, modelling log events using existing provenance models will lead to disparities among the resulting provenance. This is an issue for the reconstruction as the ground truth provenance and the reconstructed output are of different granularity.

The Data Flow Provenance Model (DFPM) is proposed for modelling provenance relations from a data flow perspective. Provenance concepts defined in the model are organised into granularity layers that reflect the hierarchically layered view of an operating system. A set of patterns are then defined for mapping concepts between the different granularity layers. These patterns are implemented using FSA so as to allow the mapping of the reconstructed data provenance to be done automatically.

- Application and Evaluation of a Causal-based Approach for Reconstructing Data Provenance from Log Files

An algorithm that is based on Lamport's *happened-before* causal relationship is proposed for reconstructing data provenance from the modelled provenance relations. Building on further, a evaluation methodology was also proposed for evaluating the completeness and correctness of the reconstructed data provenance. In doing so, we were able to study and understand some issues that could be encountered when applying a causal-based approach for reconstructing data provenance from log files.

7.2 Future Work

Two areas that can benefit from further research have been identified throughout the course of this research.

7.2.1 Translation between Log and Provenance Domain

As discussed in Chapter 4 and in most existing provenance models, the translation of events to the provenance domain has always been assumed. Although knowledge of the log files can be used to derive the translation, different groups of people may have a different interpretation of log events captured. This may in turn induce inaccuracy or inconsistency in the modelling of log events.

One approach is to infer the translation from a more static and neutral source of information, such as the source code of the respective applications. The objective of the analysis will be to uncover the functions or portions of the application that emitted those log events. Techniques such as source code analysis [Huq et al. 2013] can be applied to analyse the syntax used, so as to deduce the nature of those functions. Based on the deduced understanding, the translation can be inferred. Approaches based on other types of analysis, such as binary [Rosenblum et al. 2008] or control flow analysis [Cesare and Xiang 2010], can also be studied and compared.

7.2.2 Resolving the Dependency Explosion Problem

Resolving the dependency explosion problem is currently an open research problem. Approaches proposed, such as those from Lee et al. [2013], involve instrumenting the logging mechanism to segment the system call traces during logging. However such approaches are often considered intrusive and cannot be applied to legacy systems or log files that are generated from non-instrumented mechanisms. Other approaches such as the framework proposed by Khan et al. [2007] adopt a learning approach, using neural networks to learn the behaviour and patterns of running applications. These approaches are not flexible as the trained network can only segment system call traces of applications it is trained for.

Drawing from the experiments conducted on improving precision of the reconstructed provenance, further studies into how dependencies

can be identified may help mitigate the effects of the problem. Through analysing and mining datasets that represent different types of system usage and process behaviours, an attribute set that can be used for classifying dependencies from other file objects can be obtained. Once identified, dependencies can be pruned from the input dataset, hence eliminating the cause of the dependency explosion problem. Although pruning can be said to be an indirect approach to resolving the problem (i.e. it does not address the issue of how to accurately identify relevant relations or events), it can help reduce the impact of the problem.

7.2.3 Extending the Data Flow Provenance Model

The current proposed DFPM model focuses on modelling file interactions. However, in a computer system, transient data can flow between processes through memory regions. Such data flows are important as data may be transformed in-memory (e.g. arithmetic computation). The current proposed model does not model transient data flows because such events are not captured in normal log files. As a result, tools that can monitor and capture in-memory system activities would need to be developed first.

Another aspect in which DFPM can benefit from future work is extending the activities modelled at the user layer. Currently, only *copy* and *merge* are defined. Depending on the situation and nature of the provenance, other activities that captures a richer form of user interaction may be required. For example, to model data leakage, extending the user layer to include activities such as taking of screenshots or the downloading of objects through different avenues may benefit the analysis on the modelled provenance.

Appendix A

List of Attacks Carried Out in Challenge

The following table lists the labels and categories of the attacks observed to have been carried out by the red teams in round 2 of the New Zealand Cyber Security Challenge 2015.

Table A.1: Categories and labels for attacks carried out by the red teams

Category	Attack Label	Brief Description
Fingerprinting	nmap	Uses the tool, nmap, to perform hosts and ports scanning.
DoS ¹	crontab	Edits victim host's crontab to periodically kill Apache service.
DoS	config-tamper	Tampers with Apache service's configuration file so as to achieve the effect of a DoS attack.
DoS	permission-tamper	Tampers with permission of ssh key file so as to deny access to host.
DoS	process-kill	Forcefully terminate running process on terminal of remote host. Attacker used this to either terminates running service or disconnect victim from their host (remote connection).

¹Denial of Service attacks.

Appendix A List of Attacks Carried Out in Challenge

Table A.1: Categories and labels for attacks carried out by the red teams (continued)

Category	Attack Label	Brief Description
DoS	firewall	Tampers with victim host's firewall rules to drop packets for target services. This is to achieve the effect of a DoS attack.
Remote	shellshock	Exploits a vulnerability in CGI script execution that allows arbitrary code execution via Bash.
Remote	brute force password	Uses brute force approach to guess the password on a login page on the web browser.
Remote	php-backdoor	A Metasploit exploit that exploits an arbitrary code execution vulnerability in phpmyadmin v.3.5.2.2. Runs a reverse shell to connect to host.
Remote	php-preg-replace	A Metasploit exploit that exploits phpmyadmin's replace_prefix_tbl vulnerability. Runs a reverse shell to connect to host.
Remote	sql-injection	Exploit the lack of parameter sanitisation for injecting and executing arbitrary code and queries through the SQL database.
Others	inject-key	Attacker injects their own ssh keys into victim's ssh authorized key list. Usually the prequel for an unauthorised ssh access.
Others	unauthorised ssh access	Attacker logs in to victim's machine through ssh.
Others	exe-tamper	Rename program executable so as to hide applications or masquerade malicious executables.
Others	content-tamper	Tamper with content of files residing on victim host machine.

Appendix B

Scenarios for Designing Finite State Automata

This appendix lists the sets of scenarios designed for discovering patterns that can represent the application and user layer activities defined in DFPM.

Table B.1: Scenarios for *CreateFile*

Description of Scenario	Sample Command	Comments
Create a new file using Linux native application, touch	<code>touch file.txt</code>	
Create a new file using Linux redirect and input terminal	<code>echo "data" > file.txt</code>	
Create a new file using the editor, vim	<code>vim file.txt</code>	
Create a new file using the editor, vim , and write data into the file	<code>vim file.txt</code>	data is written using the vim interface. File is saved and closed.
Create a new file using the editor, nano	<code>nano file.txt</code>	

Appendix B Scenarios for Designing Finite State Automata

Table B.1: Scenarios for *CreateFile* (continued)

Description of Scenario	Sample Command	Comments
Create a new file using C file I/O library	<code>./createfile.o</code>	'createfile.o' is a compiled C program. File created using <i>fopen</i> in the <i>stdio.h</i> library. File object was given executable permission.

Table B.2: Scenarios for *Access*

Description of Scenario	Sample Command	Comments
Read data off a file using Linux native application, cat	<code>cat file.txt</code>	
Read data off a file using Linux native application, tail	<code>tail file.txt</code>	
Read data off a file using Linux native application, fmt	<code>fmt file.txt</code>	
Read data off a file using the editor, nano	<code>nano file.txt</code>	
Read and redirect content of a file using Linux pipes	<code>grep "string" < file.txt</code>	<i>grep</i> acts as a receiving process for the redirected data.
Read and redirect content of a file using Linux pipes	<code>fmt < file.txt</code>	<i>fmt</i> acts as a receiving process for the redirected data.
Read content of a file using the editor, vim	<code>vim file.txt</code>	

Table B.2: Scenarios for Access (continued)

Description of Scenario	Sample Command	Comments
Read content of a file using C file I/O library and print to terminal	<code>./readfile.o</code>	'readfile.o' is a compiled C program and made executable. File is accessed using <i>fopen</i> in the <i>stdio.h</i> library. Data read is printed to terminal
Read content of a file using cat and passing it to another application for processing	<code>cat file.txt grep "string"</code>	
Read content of a file using cat and passing it to another application for processing	<code>cat file.txt fmt -w 5</code>	

Table B.3: Scenarios for Modify

Description of Scenario	Sample Command	Comments
Write data into an empty file using Linux native application, echo and pipes	<code>echo "string" > file.txt</code>	Empty file is newly created separately.
Append data into an empty file using Linux native application, echo and redirect with append mode	<code>echo "string" » file.txt</code>	Empty file is newly created separately.
Write data into an empty file using the editor, vim	<code>vim file.txt</code>	uses the <i>vim</i> interface and writes "string" into file. Empty file is newly created separately.

Appendix B Scenarios for Designing Finite State Automata

Table B.3: Scenarios for *Modify* (continued)

Description of Scenario	Sample Command	Comments
Write data into an empty file using the editor, nano	<code>nano file.txt</code>	uses the <i>nano</i> interface and writes “string” into file. Empty file is newly created separately.
Edit content of a file using the editor, vim	<code>vim file.txt</code>	uses the <i>vim</i> interface and edit the content.
Edit content of a file using the editor, nano	<code>nano file.txt</code>	uses the <i>nano</i> interface and edit the content.

Table B.4: Scenarios for *DataTransfer*

Description of Scenario	Sample Command	Comments
Sending a file over the network, unencrypted, using netcat	<code>nc <dest_add> <port> <file.txt</code>	
Receiving a file over the network, unencrypted, using netcat	<code>nc -l <port> > file.txt</code>	
Sending a file over the network, encrypted, using scp	<code>scp file.txt <dest_add></code>	

Table B.5: Scenarios for *Copy*

Description of Scenario	Sample Command	Comments
Copy content of a file using Linux native application, cp into a new file	<code>cp file.txt result.txt</code>	

Table B.5: Scenarios for *Copy* (continued)

Description of Scenario	Sample Command	Comments
Copy content of a file into a new file using Linux native application, cat and Linux redirect	cat file.txt > result.txt	

Table B.6: Scenarios for *Merge*

Description of Scenario	Sample Command	Comments
Append content of a file into another existing file using Linux redirect with append mode and cat	cat file.txt » result.txt	
Append "string" from input terminal to an existing file using Linux redirect with append mode and echo	echo "string" » file.txt	

Appendix C

Experimental Results for Process-access based Pruning

This appendix documents the full list of results for the experiment on pruning file objects based on the number of processes accessing the file object (i.e. process-access).

Table C.1: Results for use-case 1

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
0.2105	0.8	1539	54.38%
0.2105	0.8	1343	47.45%
0.2105	0.8	1302	46.00%
0.2105	0.8	1292	45.65%
0.2105	0.8	1289	45.54%
0.2105	0.8	1226	43.32%
0.2105	0.8	1129	39.89%
0.2105	0.8	891	31.48%
0.2105	0.8	856	30.24%
0.2352	0.8	830	29.32%
0.2352	0.8	695	24.55%
0.2352	0.8	636	22.47%
0.2352	0.8	602	21.27%
0.2352	0.8	586	20.70%

Appendix C Experimental Results for Process-access based Pruning

Table C.1: Results for use-case 1 (continued)

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
0.2352	0.8	506	17.87%
0.2352	0.8	486	17.17%
0.2352	0.8	455	16.07%
0.2352	0.8	453	16.00%
0.2352	0.8	401	14.16%
0.2352	0.8	361	12.75%
0.2352	0.8	353	12.47%
0.2352	0.8	308	10.88%
0.2352	0.8	307	10.84%
0.2352	0.8	304	10.74%
0.2352	0.8	280	9.89%
0.2352	0.8	278	9.82%
0.2352	0.8	268	9.46%
0.2352	0.8	248	8.76%
0.2352	0.8	247	8.72%
0.2352	0.8	245	8.65%
0.2352	0.8	239	8.44%
0.2352	0.8	238	8.40%
0.2352	0.8	231	8.16%
0.2352	0.8	227	8.02%
0.2352	0.8	223	7.87%
0.2352	0.8	222	7.84%
0.2352	0.8	212	7.49%
0.2352	0.8	197	6.96%
0.2352	0.8	192	6.78%
0.2352	0.8	182	6.43%
0.2352	0.8	172	6.07%
0.2352	0.8	161	5.68%

Table C.1: Results for use-case 1 (continued)

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
0.2352	0.8	128	4.52%
0.2352	0.8	120	4.24%
0.2352	0.8	109	3.85%
0.2352	0.8	103	3.63%
0.2352	0.8	101	3.56%
0.2352	0.8	99	3.49%
0.2352	0.8	95	3.35%
0.2352	0.8	90	3.18%
0.2352	0.8	88	3.10%
0.2352	0.8	82	2.89%
0.2352	0.8	80	2.82%
0.2352	0.8	78	2.75%
0.2352	0.8	77	2.72%
0.2352	0.8	69	2.43%
0.2352	0.8	67	2.36%
0.2352	0.8	64	2.26%
0.2352	0.8	62	2.19%
0.2352	0.8	58	2.04%
0.2352	0.8	56	1.97%
0.2352	0.8	55	1.94%
0.2352	0.8	53	1.87%
0.2352	0.8	52	1.83%
0.2352	0.8	51	1.80%
0.2352	0.8	50	1.76%
0.2352	0.8	47	1.66%
0.2352	0.8	44	1.55%
0.2352	0.8	43	1.51%
0.2352	0.8	41	1.44%

Appendix C Experimental Results for Process-access based Pruning

Table C.1: Results for use-case 1 (continued)

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
0.2352	0.8	39	1.37%
0.2352	0.8	37	1.30%
0.2352	0.8	36	1.27%
0.2352	0.8	35	1.23%
0.2352	0.8	34	1.20%
0.2352	0.8	33	1.16%
0.2352	0.8	32	1.13%
0.2352	0.8	29	1.02%
0.2352	0.8	28	0.98%
0.2352	0.8	27	0.95%
0.2352	0.8	26	0.91%
0.2352	0.8	24	0.84%
0.2352	0.8	23	0.81%
0.2352	0.8	22	0.77%
0.2352	0.8	21	0.74%
0.2352	0.8	20	0.70%
0.2352	0.8	19	0.67%
0.2352	0.8	18	0.63%
0.2352	0.8	17	0.60%
0.2352	0.8	16	0.56%
0.2352	0.8	15	0.53%
0.2352	0.8	14	0.49%
0.2352	0.8	13	0.45%
0.2352	0.8	12	0.42%
0.2352	0.8	11	0.38%
0.2352	0.8	10	0.35%
0.2352	0.8	9	0.31%
0.2352	0.8	8	0.28%

Table C.1: Results for use-case 1 (continued)

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
0.2352	0.8	7	0.24%
0.2352	0.8	6	0.21%
0.2352	0.8	5	0.17%
0.2352	0.8	4	0.14%
0	0	3	0.10%
0	0	2	0.07%
0	0	1	0.03%

Table C.2: Results for use-case 2

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
5.12e ⁻⁵	1	2436	41.51%
5.39e ⁻⁵	1	1901	32.39%
5.39e ⁻⁵	1	1798	30.64%
5.39e ⁻⁵	1	1785	30.41%
5.39e ⁻⁵	1	1702	29%
5.56e ⁻⁵	1	1691	28.81%
5.73e ⁻⁵	1	1573	26.8%
5.73e ⁻⁵	1	1561	26.6%
5.97e ⁻⁵	1	1386	23.61%
5.97e ⁻⁵	1	1315	22.4%
6.44e ⁻⁵	1	1039	17.7%
6.44e ⁻⁵	1	1025	17.46%
6.61e ⁻⁵	1	881	15.01%
6.61e ⁻⁵	1	868	14.79%
6.89e ⁻⁵	1	859	14.63%
6.89e ⁻⁵	1	785	13.37%

Appendix C Experimental Results for Process-access based Pruning

Table C.2: Results for use-case 2 (continued)

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
7.10e ⁻⁵	1	723	12.32%
6.92e ⁻⁵	1	709	12.08%
7.33e ⁻⁵	1	675	11.5%
7.45e ⁻⁵	1	649	11.05%
7.45e ⁻⁵	1	645	10.99%
7.45e ⁻⁵	1	633	10.78%
8.85e ⁻⁵	1	619	10.54%
9.05e ⁻⁵	1	605	10.31%
9.21e ⁻⁵	1	586	9.98%
1.00e ⁻⁴	1	570	9.71%
1.01e ⁻⁴	1	544	9.27%
1.01e ⁻⁴	1	540	9.2%
1.01e ⁻⁴	1	539	9.18%
1.01e ⁻⁴	1	537	9.15%
1.01e ⁻⁴	1	530	9.03%
1.01e ⁻⁴	1	525	8.94%
1.01e ⁻⁴	1	523	8.91%
1.01e ⁻⁴	1	522	8.89%
9.80e ⁻⁵	1	507	8.64%
9.80e ⁻⁵	1	505	8.6%
9.80e ⁻⁵	1	504	8.58%
1.19e ⁻⁴	1	503	8.57%
1.19e ⁻⁴	1	502	8.55%
1.19e ⁻⁴	1	496	8.45%
1.22e ⁻⁴	1	417	7.1%
1.22e ⁻⁴	1	412	7.02%
1.22e ⁻⁴	1	372	6.33%
1.22e ⁻⁴	1	368	6.27%

Table C.2: Results for use-case 2 (continued)

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
1.22e ⁻⁴	1	358	6.1%
1.22e ⁻⁴	1	356	6.06%
1.22e ⁻⁴	1	343	5.84%
1.24e ⁻⁴	1	338	5.76%
1.24e ⁻⁴	1	327	5.57%
1.24e ⁻⁴	1	326	5.55%
1.24e ⁻⁴	1	325	5.53%
1.24e ⁻⁴	1	323	5.5%
1.24e ⁻⁴	1	318	5.41%
1.24e ⁻⁴	1	316	5.38%
1.24e ⁻⁴	1	300	5.11%
1.24e ⁻⁴	1	261	4.44%
1.24e ⁻⁴	1	235	4%
1.24e ⁻⁴	1	230	3.91%
1.25e ⁻⁴	1	221	3.76%
1.25e ⁻⁴	1	180	3.06%
1.25e ⁻⁴	1	179	3.05%
1.25e ⁻⁴	1	177	3.01%
1.27e ⁻⁴	1	147	2.5%
1.27e ⁻⁴	1	137	2.33%
1.27e ⁻⁴	1	136	2.31%
3.82e ⁻⁴	1	134	2.28%
3.82e ⁻⁴	1	132	2.24%
3.82e ⁻⁴	1	118	2.01%
3.82e ⁻⁴	1	102	1.73%
3.82e ⁻⁴	1	101	1.72%
3.82e ⁻⁴	1	100	1.7%
3.82e ⁻⁴	1	99	1.68%

Appendix C Experimental Results for Process-access based Pruning

Table C.2: Results for use-case 2 (continued)

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
3.82e ⁻⁴	1	98	1.67%
3.82e ⁻⁴	1	97	1.65%
3.82e ⁻⁴	1	96	1.63%
3.82e ⁻⁴	1	95	1.61%
3.82e ⁻⁴	1	90	1.53%
3.82e ⁻⁴	1	88	1.49%
3.82e ⁻⁴	1	74	1.26%
3.82e ⁻⁴	1	72	1.22%
3.82e ⁻⁴	1	70	1.19%
3.82e ⁻⁴	1	69	1.17%
3.82e ⁻⁴	1	63	1.07%
3.82e ⁻⁴	1	62	1.05%
3.82e ⁻⁴	1	61	1.03%
3.82e ⁻⁴	1	60	1.02%
3.82e ⁻⁴	1	59	1%
3.82e ⁻⁴	1	57	0.97%
3.82e ⁻⁴	1	55	0.93%
3.82e ⁻⁴	1	53	0.9%
3.82e ⁻⁴	1	52	0.88%
4.01e ⁻⁴	1	51	0.86%
4.01e ⁻⁴	1	49	0.83%
4.01e ⁻⁴	1	47	0.8%
4.01e ⁻⁴	1	46	0.78%
4.01e ⁻⁴	1	45	0.76%
4.01e ⁻⁴	1	44	0.74%
4.05e ⁻⁴	1	43	0.73%
4.05e ⁻⁴	1	40	0.68%
4.05e ⁻⁴	1	37	0.63%

Table C.2: Results for use-case 2 (continued)

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
4.05e ⁻⁴	1	36	0.61%
4.05e ⁻⁴	1	35	0.59%
4.05e ⁻⁴	1	34	0.57%
4.05e ⁻⁴	1	33	0.56%
4.05e ⁻⁴	1	32	0.54%
4.05e ⁻⁴	1	31	0.52%
4.05e ⁻⁴	1	27	0.46%
4.05e ⁻⁴	1	26	0.44%
4.05e ⁻⁴	1	25	0.42%
4.05e ⁻⁴	1	24	0.4%
4.05e ⁻⁴	1	23	0.39%
4.05e ⁻⁴	1	22	0.37%
4.05e ⁻⁴	1	21	0.35%
4.05e ⁻⁴	1	20	0.34%
4.05e ⁻⁴	1	19	0.32%
4.05e ⁻⁴	1	18	0.3%
4.05e ⁻⁴	1	17	0.28%
4.05e ⁻⁴	1	16	0.27%
4.72e ⁻³	1	15	0.25%
5.49e ⁻³	1	14	0.23%
5.49e ⁻³	1	13	0.22%
5.50e ⁻³	1	12	0.2%
5.50e ⁻³	1	11	0.18%
5.50e ⁻³	1	10	0.17%
5.50e ⁻³	1	9	0.15%
5.50e ⁻³	1	8	0.13%
5.58e ⁻³	1	7	0.11%
5.58e ⁻³	1	6	0.1%

Appendix C Experimental Results for Process-access based Pruning

Table C.2: Results for use-case 2 (continued)

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
$5.58e^{-3}$	1	5	0.08%
$5.58e^{-3}$	1	4	0.06%
0	0	3	0.05%
0	0	2	0.03%
0	0	1	0.01%

Table C.3: Results for use-case 3

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
$1.09e^{-5}$	1	4367	54.26%
$1.11e^{-5}$	1	3855	47.9%
$1.12e^{-5}$	1	2378	29.54%
$1.12e^{-5}$	1	2339	29.06%
$1.12e^{-5}$	1	2325	28.88%
$1.12e^{-5}$	1	2309	28.69%
$1.13e^{-5}$	1	2239	27.82%
$1.14e^{-5}$	1	2102	26.11%
$1.14e^{-5}$	1	2003	24.88%
$1.16e^{-5}$	1	1574	19.55%
$1.16e^{-5}$	1	1541	19.14%
$1.17e^{-5}$	1	1362	16.92%
$1.17e^{-5}$	1	1344	16.69%
$1.17e^{-5}$	1	1309	16.26%
$1.17e^{-5}$	1	1232	15.3%
$1.17e^{-5}$	1	1152	14.31%
$1.17e^{-5}$	1	1067	13.25%
$1.17e^{-5}$	1	1011	12.56%

Table C.3: Results for use-case 3 (continued)

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
1.17e ⁻⁵	1	1010	12.54%
1.22e ⁻⁵	1	999	12.41%
1.22e ⁻⁵	1	996	12.37%
1.22e ⁻⁵	1	981	12.18%
1.22e ⁻⁵	1	908	11.28%
1.22e ⁻⁵	1	881	10.94%
1.22e ⁻⁵	1	811	10.07%
1.23e ⁻⁵	1	784	9.74%
1.24e ⁻⁵	1	762	9.46%
1.24e ⁻⁵	1	749	9.3%
1.24e ⁻⁵	1	748	9.29%
1.24e ⁻⁵	1	746	9.26%
1.26e ⁻⁵	1	740	9.19%
1.26e ⁻⁵	1	711	8.83%
1.26e ⁻⁵	1	699	8.68%
1.26e ⁻⁵	1	692	8.59%
1.26e ⁻⁵	1	676	8.39%
1.26e ⁻⁵	1	672	8.34%
1.26e ⁻⁵	1	663	8.23%
1.26e ⁻⁵	1	659	8.18%
1.26e ⁻⁵	1	636	7.9%
1.26e ⁻⁵	1	628	7.8%
1.26e ⁻⁵	1	598	7.43%
1.26e ⁻⁵	1	586	7.28%
1.26e ⁻⁵	1	559	6.94%
1.26e ⁻⁵	1	550	6.83%
1.26e ⁻⁵	1	511	6.34%
1.26e ⁻⁵	1	502	6.23%

Appendix C Experimental Results for Process-access based Pruning

Table C.3: Results for use-case 3 (continued)

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
1.26e ⁻⁵	1	500	6.21%
1.26e ⁻⁵	1	499	6.2%
1.26e ⁻⁵	1	474	5.88%
1.27e ⁻⁵	1	473	5.87%
1.27e ⁻⁵	1	463	5.75%
1.27e ⁻⁵	1	434	5.39%
1.62e ⁻⁵	1	420	5.21%
1.62e ⁻⁵	1	380	4.72%
1.62e ⁻⁵	1	369	4.58%
1.62e ⁻⁵	1	361	4.48%
1.62e ⁻⁵	1	347	4.31%
1.62e ⁻⁵	1	344	4.27%
1.62e ⁻⁵	1	342	4.24%
1.62e ⁻⁵	1	341	4.23%
1.62e ⁻⁵	1	333	4.13%
1.62e ⁻⁵	1	316	3.92%
1.63e ⁻⁵	1	303	3.76%
1.63e ⁻⁵	1	302	3.75%
1.64e ⁻⁵	1	285	3.54%
1.64e ⁻⁵	1	241	2.99%
1.64e ⁻⁵	1	240	2.98%
1.64e ⁻⁵	1	238	2.95%
1.64e ⁻⁵	1	231	2.87%
1.64e ⁻⁵	1	225	2.79%
1.65e ⁻⁵	1	224	2.78%
1.65e ⁻⁵	1	222	2.75%
1.65e ⁻⁵	1	215	2.67%
1.65e ⁻⁵	1	214	2.65%

Table C.3: Results for use-case 3 (continued)

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
1.65e ⁻⁵	1	213	2.64%
1.65e ⁻⁵	1	210	2.6%
1.65e ⁻⁵	1	208	2.58%
1.65e ⁻⁵	1	207	2.57%
1.65e ⁻⁵	1	197	2.44%
1.65e ⁻⁵	1	196	2.43%
1.65e ⁻⁵	1	195	2.42%
1.65e ⁻⁵	1	189	2.34%
1.65e ⁻⁵	1	187	2.32%
1.65e ⁻⁵	1	186	2.31%
1.65e ⁻⁵	1	182	2.26%
1.65e ⁻⁵	1	180	2.23%
1.65e ⁻⁵	1	179	2.22%
1.65e ⁻⁵	1	175	2.17%
1.65e ⁻⁵	1	174	2.16%
1.65e ⁻⁵	1	173	2.14%
1.65e ⁻⁵	1	172	2.13%
1.65e ⁻⁵	1	171	2.12%
1.65e ⁻⁵	1	170	2.11%
1.65e ⁻⁵	1	169	2.09%
1.67e ⁻⁵	1	165	2.05%
1.67e ⁻⁵	1	164	2.03%
1.67e ⁻⁵	1	163	2.02%
1.67e ⁻⁵	1	158	1.96%
1.67e ⁻⁵	1	157	1.95%
1.67e ⁻⁵	1	156	1.93%
1.67e ⁻⁵	1	154	1.91%
1.67e ⁻⁵	1	151	1.87%

Appendix C Experimental Results for Process-access based Pruning

Table C.3: Results for use-case 3 (continued)

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
1.67e ⁻⁵	1	147	1.82%
1.67e ⁻⁵	1	145	1.8%
1.67e ⁻⁵	1	139	1.72%
1.67e ⁻⁵	1	138	1.71%
1.67e ⁻⁵	1	137	1.7%
1.67e ⁻⁵	1	136	1.68%
1.67e ⁻⁵	1	135	1.67%
1.67e ⁻⁵	1	132	1.64%
1.67e ⁻⁵	1	130	1.61%
1.67e ⁻⁵	1	126	1.56%
1.67e ⁻⁵	1	125	1.55%
1.67e ⁻⁵	1	124	1.54%
1.67e ⁻⁵	1	119	1.47%
1.67e ⁻⁵	1	109	1.35%
1.67e ⁻⁵	1	108	1.34%
1.67e ⁻⁵	1	107	1.32%
1.67e ⁻⁵	1	106	1.31%
1.67e ⁻⁵	1	105	1.3%
1.67e ⁻⁵	1	104	1.29%
1.67e ⁻⁵	1	103	1.27%
1.67e ⁻⁵	1	93	1.15%
1.67e ⁻⁵	1	92	1.14%
1.67e ⁻⁵	1	88	1.09%
1.67e ⁻⁵	1	87	1.08%
1.67e ⁻⁵	1	85	1.05%
1.67e ⁻⁵	1	84	1.04%
1.67e ⁻⁵	1	83	1.03%
1.67e ⁻⁵	1	82	1.01%

Table C.3: Results for use-case 3 (continued)

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
1.67e ⁻⁵	1	81	1%
1.67e ⁻⁵	1	80	0.99%
1.67e ⁻⁵	1	78	0.96%
1.67e ⁻⁵	1	77	0.95%
1.67e ⁻⁵	1	76	0.94%
1.67e ⁻⁵	1	75	0.93%
1.67e ⁻⁵	1	73	0.9%
1.67e ⁻⁵	1	72	0.89%
1.67e ⁻⁵	1	71	0.88%
1.67e ⁻⁵	1	70	0.86%
1.67e ⁻⁵	1	69	0.85%
1.67e ⁻⁵	1	66	0.82%
1.67e ⁻⁵	1	65	0.8%
1.67e ⁻⁵	1	62	0.77%
1.67e ⁻⁵	1	61	0.75%
1.67e ⁻⁵	1	60	0.74%
1.67e ⁻⁵	1	57	0.7%
1.67e ⁻⁵	1	56	0.69%
1.67e ⁻⁵	1	55	0.68%
1.67e ⁻⁵	1	54	0.67%
1.67e ⁻⁵	1	53	0.65%
1.67e ⁻⁵	1	52	0.64%
1.67e ⁻⁵	1	51	0.63%
1.67e ⁻⁵	1	49	0.6%
1.67e ⁻⁵	1	48	0.59%
1.67e ⁻⁵	1	47	0.58%
1.67e ⁻⁵	1	46	0.57%
1.67e ⁻⁵	1	45	0.55%

Appendix C Experimental Results for Process-access based Pruning

Table C.3: Results for use-case 3 (continued)

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
1.67e ⁻⁵	1	44	0.54%
1.67e ⁻⁵	1	43	0.53%
1.67e ⁻⁵	1	42	0.52%
1.67e ⁻⁵	1	41	0.5%
1.67e ⁻⁵	1	40	0.49%
1.67e ⁻⁵	1	39	0.48%
1.67e ⁻⁵	1	38	0.47%
1.67e ⁻⁵	1	37	0.45%
1.67e ⁻⁵	1	36	0.44%
1.67e ⁻⁵	1	35	0.43%
1.67e ⁻⁵	1	34	0.42%
1.67e ⁻⁵	1	33	0.41%
1.67e ⁻⁵	1	31	0.38%
1.67e ⁻⁵	1	30	0.37%
1.67e ⁻⁵	1	29	0.36%
1.67e ⁻⁵	1	28	0.34%
1.67e ⁻⁵	1	27	0.33%
1.67e ⁻⁵	1	26	0.32%
1.67e ⁻⁵	1	25	0.31%
1.67e ⁻⁵	1	24	0.29%
1.67e ⁻⁵	1	23	0.28%
1.67e ⁻⁵	1	22	0.27%
1.67e ⁻⁵	1	21	0.26%
1.67e ⁻⁵	1	20	0.24%
1.67e ⁻⁵	1	19	0.23%
1.67e ⁻⁵	1	18	0.22%
1.67e ⁻⁵	1	17	0.21%
1.67e ⁻⁵	1	16	0.19%

Table C.3: Results for use-case 3 (continued)

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
$1.67e^{-5}$	1	15	0.18%
$1.67e^{-5}$	1	14	0.17%
$1.67e^{-5}$	1	13	0.16%
$1.67e^{-5}$	1	12	0.14%
$1.67e^{-5}$	1	11	0.13%
$1.04e^{-4}$	1	10	0.12%
$1.04e^{-4}$	1	9	0.11%
$1.04e^{-4}$	1	8	0.09%
$1.04e^{-4}$	1	7	0.08%
$1.04e^{-4}$	1	6	0.07%
$1.04e^{-4}$	1	5	0.06%
$1.04e^{-4}$	1	4	0.04%
0	0	3	0.03%
0	0	2	0.02%
0	0	1	0.01%

Table C.4: Results for use-case 4

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
$9.54e^{-5}$	1	4367	54.26%
$9.75e^{-5}$	1	3855	47.9%
$9.83e^{-5}$	1	2378	29.54%
$9.83e^{-5}$	1	2339	29.06%
$9.83e^{-5}$	1	2325	28.88%
$9.83e^{-5}$	1	2309	28.69%
$9.61e^{-5}$	1	2239	27.82%
$9.73e^{-5}$	1	2102	26.11%

Appendix C Experimental Results for Process-access based Pruning

Table C.4: Results for use-case 4 (continued)

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
9.73e ⁻⁵	1	2003	24.88%
9.89e ⁻⁵	1	1574	19.55%
9.89e ⁻⁵	1	1541	19.14%
9.95e ⁻⁵	1	1362	16.92%
9.96e ⁻⁵	1	1344	16.69%
1.03e ⁻⁴	1	1309	16.26%
1.03e ⁻⁴	1	1232	15.3%
1.03e ⁻⁴	1	1152	14.31%
1.03e ⁻⁴	1	1067	13.25%
1.03e ⁻⁴	1	1011	12.56%
1.03e ⁻⁴	1	1010	12.54%
1.06e ⁻⁴	1	999	12.41%
1.06e ⁻⁴	1	996	12.37%
1.07e ⁻⁴	1	981	12.18%
1.07e ⁻⁴	1	908	11.28%
1.07e ⁻⁴	1	881	10.94%
1.08e ⁻⁴	1	811	10.07%
1.08e ⁻⁴	1	784	9.74%
1.09e ⁻⁴	1	762	9.46%
1.09e ⁻⁴	1	749	9.3%
1.09e ⁻⁴	1	748	9.29%
1.09e ⁻⁴	1	746	9.26%
1.10e ⁻⁴	1	740	9.19%
1.10e ⁻⁴	1	711	8.83%
1.10e ⁻⁴	1	699	8.68%
1.10e ⁻⁴	1	692	8.59%
1.10e ⁻⁴	1	676	8.39%
1.10e ⁻⁴	1	672	8.34%

Table C.4: Results for use-case 4 (continued)

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
1.10e ⁻⁴	1	663	8.23%
1.10e ⁻⁴	1	659	8.18%
1.10e ⁻⁴	1	636	7.9%
1.10e ⁻⁴	1	628	7.8%
1.10e ⁻⁴	1	598	7.43%
1.11e ⁻⁴	1	586	7.28%
1.11e ⁻⁴	1	559	6.94%
1.11e ⁻⁴	1	550	6.83%
1.11e ⁻⁴	1	511	6.34%
1.11e ⁻⁴	1	502	6.23%
1.11e ⁻⁴	1	500	6.21%
1.11e ⁻⁴	1	499	6.2%
1.11e ⁻⁴	1	474	5.88%
1.11e ⁻⁴	1	473	5.87%
1.11e ⁻⁴	1	463	5.75%
1.11e ⁻⁴	1	434	5.39%
1.38e ⁻⁴	1	420	5.21%
1.38e ⁻⁴	1	380	4.72%
1.38e ⁻⁴	1	369	4.58%
1.38e ⁻⁴	1	361	4.48%
1.38e ⁻⁴	1	347	4.31%
1.38e ⁻⁴	1	344	4.27%
1.38e ⁻⁴	1	342	4.24%
1.38e ⁻⁴	1	341	4.23%
1.38e ⁻⁴	1	333	4.13%
1.38e ⁻⁴	1	316	3.92%
1.39e ⁻⁴	1	303	3.76%
1.39e ⁻⁴	1	302	3.75%

Appendix C Experimental Results for Process-access based Pruning

Table C.4: Results for use-case 4 (continued)

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
1.4e ⁻⁴	1	285	3.54%
1.4e ⁻⁴	1	241	2.99%
1.4e ⁻⁴	1	240	2.98%
1.4e ⁻⁴	1	238	2.95%
1.4e ⁻⁴	1	231	2.87%
1.4e ⁻⁴	1	225	2.79%
1.406e ⁻⁴	1	224	2.78%
1.406e ⁻⁴	1	222	2.75%
1.406e ⁻⁴	1	215	2.67%
1.406e ⁻⁴	1	214	2.65%
1.406e ⁻⁴	1	213	2.64%
1.406e ⁻⁴	1	210	2.6%
1.406e ⁻⁴	1	208	2.58%
1.406e ⁻⁴	1	207	2.57%
1.406e ⁻⁴	1	197	2.44%
1.406e ⁻⁴	1	196	2.43%
1.406e ⁻⁴	1	195	2.42%
1.406e ⁻⁴	1	189	2.34%
1.406e ⁻⁴	1	187	2.32%
1.406e ⁻⁴	1	186	2.31%
1.406e ⁻⁴	1	182	2.26%
1.406e ⁻⁴	1	180	2.23%
1.406e ⁻⁴	1	179	2.22%
1.406e ⁻⁴	1	175	2.17%
1.406e ⁻⁴	1	174	2.16%
1.406e ⁻⁴	1	173	2.14%
1.406e ⁻⁴	1	172	2.13%
1.406e ⁻⁴	1	171	2.12%

Table C.4: Results for use-case 4 (continued)

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
1.406e ⁻⁴	1	170	2.11%
1.406e ⁻⁴	1	169	2.09%
1.43e ⁻⁴	1	165	2.05%
1.43e ⁻⁴	1	164	2.03%
1.43e ⁻⁴	1	163	2.02%
1.43e ⁻⁴	1	158	1.96%
1.43e ⁻⁴	1	157	1.95%
1.43e ⁻⁴	1	156	1.93%
1.43e ⁻⁴	1	154	1.91%
1.43e ⁻⁴	1	151	1.87%
1.43e ⁻⁴	1	147	1.82%
1.43e ⁻⁴	1	145	1.8%
1.43e ⁻⁴	1	139	1.72%
1.43e ⁻⁴	1	138	1.71%
1.43e ⁻⁴	1	137	1.7%
1.43e ⁻⁴	1	136	1.68%
1.43e ⁻⁴	1	135	1.67%
1.43e ⁻⁴	1	132	1.64%
1.43e ⁻⁴	1	130	1.61%
1.43e ⁻⁴	1	126	1.56%
1.43e ⁻⁴	1	125	1.55%
1.43e ⁻⁴	1	124	1.54%
1.43e ⁻⁴	1	119	1.47%
1.43e ⁻⁴	1	109	1.35%
1.43e ⁻⁴	1	108	1.34%
1.43e ⁻⁴	1	107	1.32%
1.43e ⁻⁴	1	106	1.31%
1.43e ⁻⁴	1	105	1.3%

Appendix C Experimental Results for Process-access based Pruning

Table C.4: Results for use-case 4 (continued)

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
1.43e ⁻⁴	1	104	1.29%
1.43e ⁻⁴	1	103	1.27%
1.43e ⁻⁴	1	93	1.15%
1.43e ⁻⁴	1	92	1.14%
1.43e ⁻⁴	1	88	1.09%
1.43e ⁻⁴	1	87	1.08%
1.43e ⁻⁴	1	85	1.05%
1.43e ⁻⁴	1	84	1.04%
1.43e ⁻⁴	1	83	1.03%
1.43e ⁻⁴	1	82	1.01%
1.43e ⁻⁴	1	81	1%
1.43e ⁻⁴	1	80	0.99%
1.43e ⁻⁴	1	78	0.96%
1.43e ⁻⁴	1	77	0.95%
1.43e ⁻⁴	1	76	0.94%
1.43e ⁻⁴	1	75	0.93%
1.43e ⁻⁴	1	73	0.9%
1.43e ⁻⁴	1	72	0.89%
1.43e ⁻⁴	1	71	0.88%
1.43e ⁻⁴	1	70	0.86%
1.43e ⁻⁴	1	69	0.85%
1.43e ⁻⁴	1	66	0.82%
1.43e ⁻⁴	1	65	0.8%
1.43e ⁻⁴	1	62	0.77%
1.43e ⁻⁴	1	61	0.75%
1.43e ⁻⁴	1	60	0.74%
1.43e ⁻⁴	1	57	0.7%
1.43e ⁻⁴	1	56	0.69%

Table C.4: Results for use-case 4 (continued)

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
1.43e ⁻⁴	1	55	0.68%
1.43e ⁻⁴	1	54	0.67%
1.43e ⁻⁴	1	53	0.65%
1.43e ⁻⁴	1	52	0.64%
1.43e ⁻⁴	1	51	0.63%
1.43e ⁻⁴	1	49	0.6%
1.43e ⁻⁴	1	48	0.59%
1.43e ⁻⁴	1	47	0.58%
1.43e ⁻⁴	1	46	0.57%
1.43e ⁻⁴	1	45	0.55%
1.43e ⁻⁴	1	44	0.54%
1.43e ⁻⁴	1	43	0.53%
1.43e ⁻⁴	1	42	0.52%
1.47e ⁻⁴	1	41	0.5%
1.47e ⁻⁴	1	40	0.49%
1.47e ⁻⁴	1	39	0.48%
1.47e ⁻⁴	1	38	0.47%
1.47e ⁻⁴	1	37	0.45%
1.47e ⁻⁴	1	36	0.44%
1.47e ⁻⁴	1	35	0.43%
1.47e ⁻⁴	1	34	0.42%
1.47e ⁻⁴	1	33	0.41%
1.47e ⁻⁴	1	31	0.38%
1.47e ⁻⁴	1	30	0.37%
1.47e ⁻⁴	1	29	0.36%
1.47e ⁻⁴	1	28	0.34%
1.47e ⁻⁴	1	27	0.33%
1.47e ⁻⁴	1	26	0.32%

Appendix C Experimental Results for Process-access based Pruning

Table C.4: Results for use-case 4 (continued)

Precision	Recall	No. of Process-access	Process-access w.r.t. total no. of unique processes
1.47e ⁻⁴	1	25	0.31%
1.47e ⁻⁴	1	24	0.29%
1.47e ⁻⁴	1	23	0.28%
1.47e ⁻⁴	1	22	0.27%
1.47e ⁻⁴	1	21	0.26%
1.47e ⁻⁴	1	20	0.24%
1.47e ⁻⁴	1	19	0.23%
1.47e ⁻⁴	1	18	0.22%
1.47e ⁻⁴	1	17	0.21%
1.47e ⁻⁴	1	16	0.19%
1.47e ⁻⁴	1	15	0.18%
1.47e ⁻⁴	1	14	0.17%
1.47e ⁻⁴	1	13	0.16%
1.47e ⁻⁴	1	12	0.14%
1.47e ⁻⁴	1	11	0.13%
0	0	10	0.12%
0	0	9	0.11%
0	0	8	0.09%
0	0	7	0.08%
0	0	6	0.07%
0	0	5	0.06%
0	0	4	0.04%
0	0	3	0.03%
0	0	2	0.02%
0	0	1	0.01%

Appendix D

Graph Plots for Determining Number of Clusters

This appendix lists the graph plots, generated from each system log file in the NZCSC'15 dataset, for determining the number of clusters, k . Based on the elbow method, used for estimating the number of clusters to be used with the k-means algorithm, the plots demonstrate k is estimated to be 5 for majority of the system log files in the dataset.

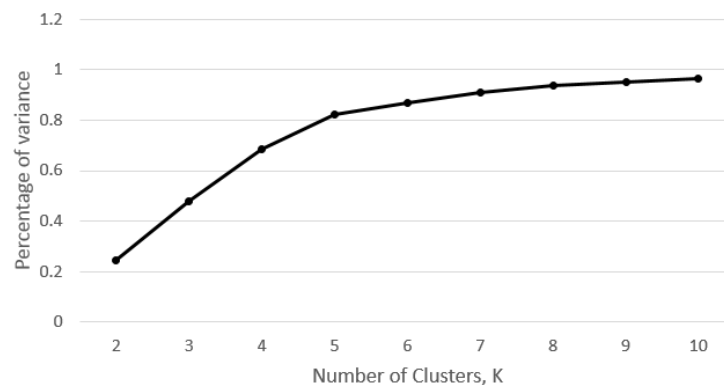


Figure D.1: Blue Team 1 - Web server system log file

Appendix D Graph Plots for Determining Number of Clusters

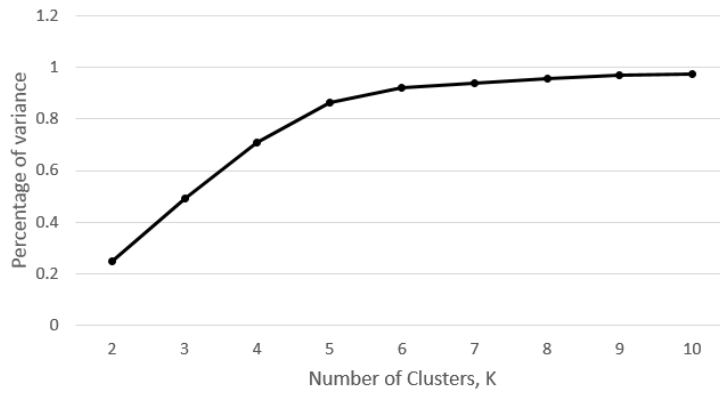


Figure D.2: Blue Team 1 - File server system log file

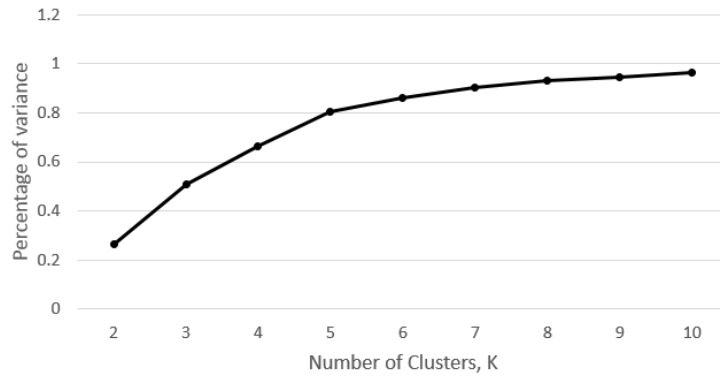


Figure D.3: Blue Team 2 - Web server system log file

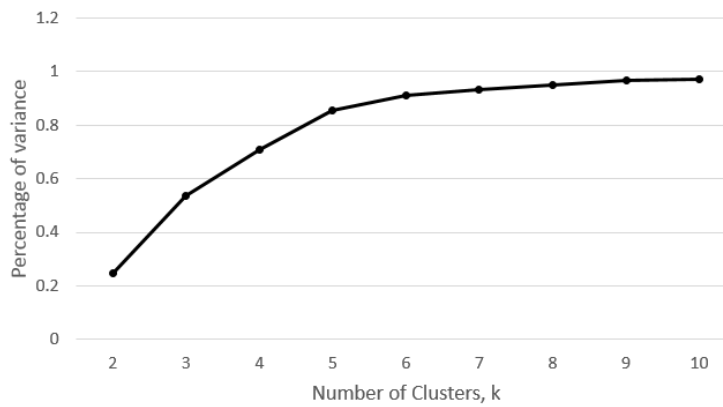


Figure D.4: Blue Team 2 - File server system log file

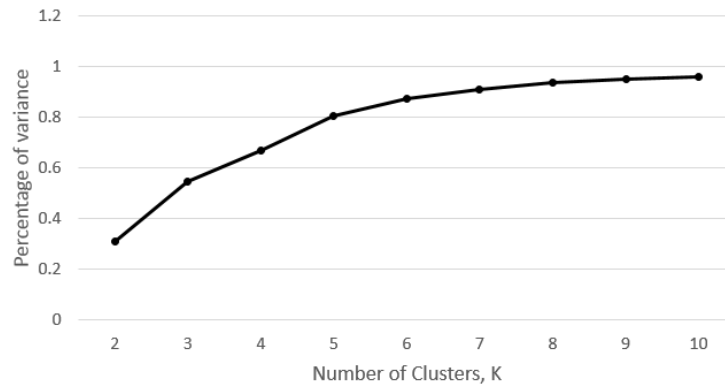


Figure D.5: Blue Team 3 - Web server system log file

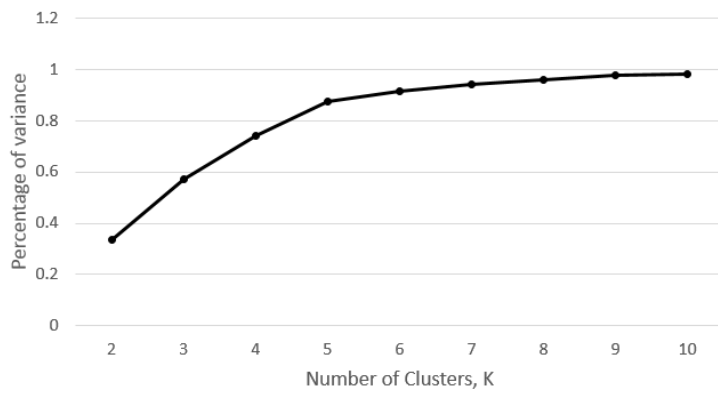


Figure D.6: Blue Team 3 - File server system log file

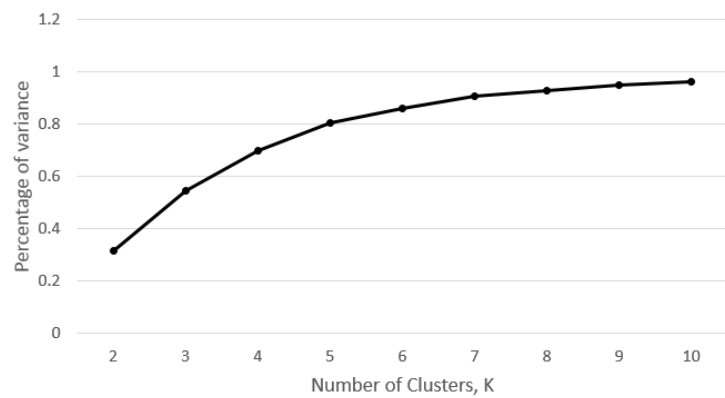


Figure D.7: Blue Team 4 - Web server system log file

Appendix D Graph Plots for Determining Number of Clusters

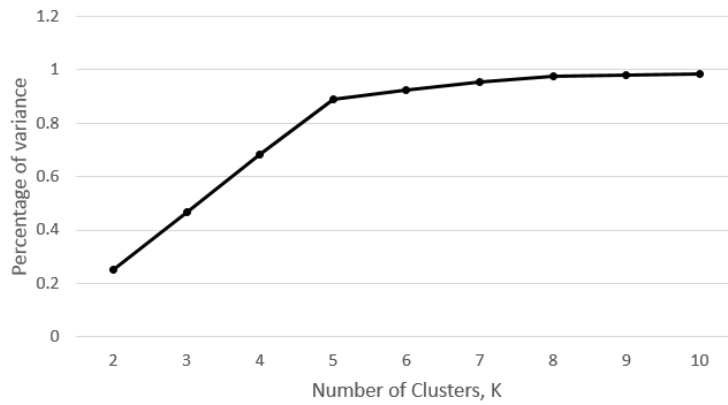


Figure D.8: Blue Team 4 - File server system log file

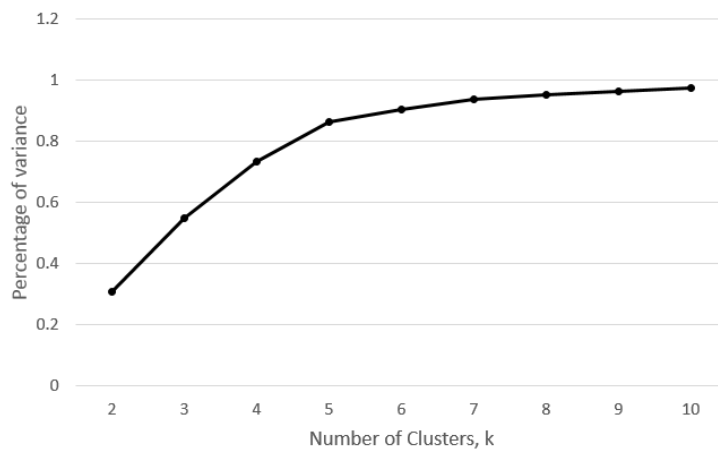


Figure D.9: Blue Team 5 - Web server system log file

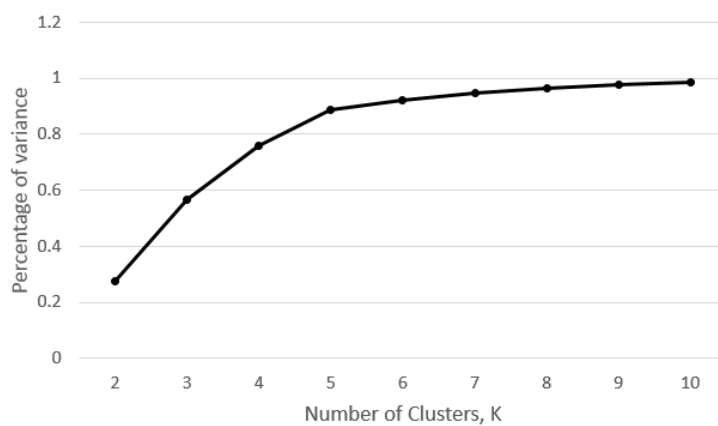


Figure D.10: Blue Team 5 - File server system log file

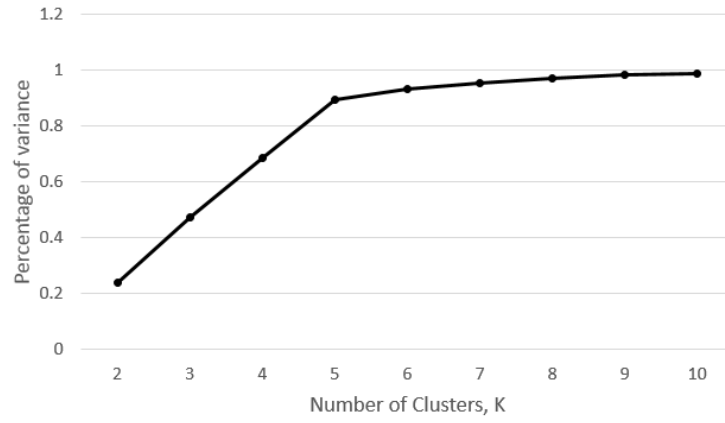


Figure D.11: Red Team 1

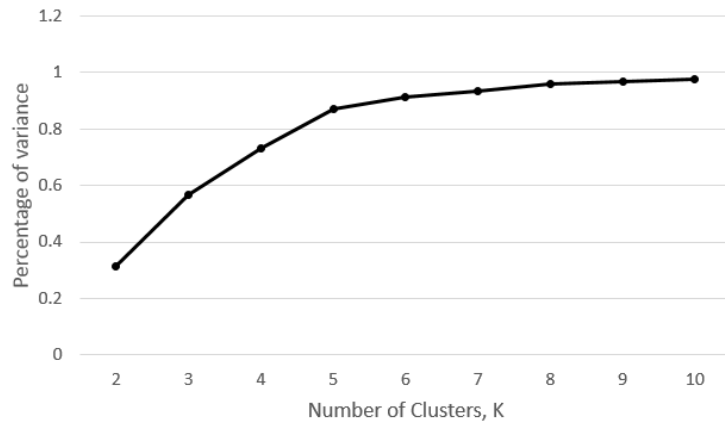


Figure D.12: Red Team 2

Bibliography

- van der Aalst, W. M. P., Weijters, A. J. M. M., Maruster, L. (2004). Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16, 1128–1142. doi: 10.1109/TKDE.2004.47. IEEE.
- Abreu, R., Archer, D., Chapman, E., Cheney, J., Eldardiry, H., Gascon, A. (2016). Provenance Segmentation. In *Proceedings of 8th USENIX Workshop on the Theory and Practice of Provenance (TaPP'16)*. Berkeley, CA: USENIX Association. Washington, D.C, USA.
- Ahmad, A., Ruighaver, A. (2002). FIRESTORM: Exploring the Need for a Forensic Tool for Pattern Correlation in Windows NT Audit Logs. In *Proceedings of the 3rd Australian Information Warfare and Security Conference*.
- Aierken, A., Davis, D. B., Zhang, Q., Gupta, K., Wong, A., Asuncion, H. U. (2014). A Multi-level Funneling Approach to Data Provenance Reconstruction. In *Proceedings of the IEEE 10th International Conference on e-Science, E-SCIENCE '14*, pp. 71–74. Washington, DC, USA: IEEE Computer Society. ISBN 978-1-4799-4287-9. doi:10.1109/eScience.2014.54.
<http://dx.doi.org/10.1109/eScience.2014.54>
- Aldeco-Pérez, R., Moreau, L. (2008). Provenance-based auditing of private data use. In *Proceedings of the 2008 International Conference on Visions of Computer Science: BCS International Academic Conference, VoCS'08*, pp. 141–152. Swindon, UK: BCS Learning & Development Ltd.
<http://dl.acm.org/citation.cfm?id=2227536.2227549>

Bibliography

- Allen, J. F. (1983). Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26 Issue 11, 832–843. doi:10.1145/182.358434.
- Allen, M. D., Chapman, A., Blaustein, B., Seligman, L. (2010a). Capturing Provenance in the Wild. In *Third International Provenance and Annotation Workshop, IPAW 2010*. Springer Berlin Heidelberg. ISBN 978-3-642-17819-1. doi:10.1007/978-3-642-17819-1_12. Troy, NY, USA.
- Allen, M. D., Seligman, L., Blaustein, B., Chapman, A. (2010b). Provenance Capture and Use: A Practical Guide. Tech. rep., The MITRE Corporation.
<https://www.mitre.org/sites/default/files/publications/practical-provenance-guide-MP100128.pdf>
- Anderson, J. A. (2006). *Automata Theory with Modern Applications*. Cambridge University Press.
- archLinux (2012). Linux Audit Framework. https://wiki.archlinux.org/index.php/Audit_framework (Last accessed: 23/06/2016).
- Backus, J. W., Bauer, F. L., Green, J., Katz, C., McCarthy, J., Naur, P., Perlis, A. J., Rutishauser, H., Samelson, K., Vauquois, B., Wegstein, J. H., van Wijngaarden, A., Woodger, M. (1960). Report on the algorithmic language ALGOL 60. *Numerische Mathematik*, 2, 106–136.
- Bambenek, J. (2007). Distilling Data in a SIM: A Strategy for the Analysis of Events in the ArcSight ESM. <https://www.sans.org/reading-room/whitepapers/detection/distilling-data-sim-strategy-analysis-events-arcsight-esm-1916> (Last accessed: 21/07/2016).
- Barga, R. S., Digiampietri, L. A. (2007). Automatic Capture and Efficient Storage of eScience Experiments Provenance. *Concurrency and Computation: Practice & Experience, Journal*, vol. 20, pp. 419–429. Published online 8 Aug 2007.
- Bavoil, L., Callahan, S. P., Crossno, P. J., Freire, J., Scheidegger, C. E., Silva, C. T., Vo, H. T. (2005). Vistrails: Enabling interactive multiple-

- view visualizations. In *Proceedings of the IEEE Visualization*, pp. 135–142. IEEE.
- Beazley, D. (2001). Ply (python lex-yacc). <http://www.dabeaz.com/ply/> (Last accessed: 01/08/2017).
- Bhattacharya, I., Getoor, L. (2005). Entity Resolution in Graphs. Tech. rep., University of Maryland. <http://drum.lib.umd.edu/bitstream/handle/1903/4021/4758.pdf;sequence=1>
- Biton, O., Cohen-Boulakia, S., B. Davidson, S., Hara, C. S. (2008). Querying and managing provenance through user views in scientific workflows. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering (ICDE'08)*, pp. 1072–1081.
- Borkin, M. A., Yeh, C. S., Boyd, M., Macko, P., Gajos, K. Z., Seltzer, M., Pfister, H. (2013). Evaluation of Filesystem Provenance Visualization Tools. *IEEE Transactions on Visualization and Computer Graphics*, 19(12), 2476–2485. doi:10.1109/TVCG.2013.155. <http://dx.doi.org/10.1109/TVCG.2013.155>
- Buchholz, F., Falk, C. (2005). Design and Implementation of Zeitline: a Forensic Timeline Editor. In *Digital Forensic Research Workshop (DFRWS)*. New Orleans, USA.
- Buneman, P., Cheney, J., Kostylev, E. V. (2012). Hierarchical Models of Provenance. In *Proceedings of the 4th USENIX Conference on Theory and Practice of Provenance (TaPP'12)*, TaPP'12, pp. 10–10. Berkeley, CA, USA: USENIX Association. <http://dl.acm.org/citation.cfm?id=2342875.2342885>
- Carata, L., Akoush, S., Balakrishnan, N., Bytheway, T., Sohan, R., Seltzer, M., Hopper, A. (2014). A Primer on Provenance. *Communications of the ACM*, 57(5), pp. 52–60.
- Carrier, B., Spafford, E. (2004a). An event-based digital forensic investigation framework. In *Digital Forensic Research Workshop*, pp. 11–13. Baltimore, MD, USA.

Bibliography

- Carrier, B. D., Spafford, E. H. (2004b). Defining Event Reconstruction of Digital Crime Scenes. *Journal of Forensic Sciences*, 49(6), 1291 – 1298.
- Case, A., Cristina, A., Marziale, L., Richard, G. G., Roussev, V. (2008). FACE: Automated Digital Evidence Discovery and Correlation. In *Proceedings of the 8th Digital Forensics Research Conference (DFRWS)*, pp. 65–75.
- Cesare, S., Xiang, Y. (2010). A Fast Flowgraph Based Classification System for Packed and Polymorphic Malware on the Endhost. In *Proceedings in the 24th IEEE International Conference on Advanced Information Networking and Applications*. IEEE. doi:10.1109/AINA.2010.121.
- Chabot, Y., Bertaux, A., Nicolle, C., Kechadi, T. (2014). Automatic Timeline Construction and Analysis for Computer Forensics Purposes. In *Proceedings of the 2014 IEEE Joint Intelligence and Security Informatics Conference, JISIC '14*, pp. 276–279. Washington, DC, USA: IEEE Computer Society. ISBN 978-1-4799-6364-5. doi:10.1109/JISIC.2014.54.
<http://dx.doi.org/10.1109/JISIC.2014.54>
- Cheah, Y.-W., Plale, B. (2012). Provenance Analysis: Towards Quality Provenance. In *Proceedings of the 9th International Conference on E-Science*. doi:10.1109/eScience.2012.6404480.
- Cheah, Y.-W., Plale, B. (2014). Provenance Quality Assessment Methodology and Framework. *Data and Information Quality, Journal*, 5(3), 9:1–9:20. doi:10.1145/2665069.
<http://doi.acm.org/10.1145/2665069>
- Chen, K., Clark, A., Vel, O. D., Mohay, G. (2003). ECF - Event Correlation For Forensics. In *1st Australian Computer Network, Information and Forensics Conference*.
<http://dspace.dsto.defence.gov.au/dspace/handle/1947/2419>

- Chen, Y., Wah, B. W., Hsu, C.-W. (2006). Temporal Planning using Sub-goal Partitioning and Resolution in SGPlan. *Artificial Intelligence and Research, Journal*, 26, 323–369.
- Cheney, J., Acar, U. A., Ahmed, A. (2008). Provenance Traces. Tech. rep., Cornell University.
- Coe, G. B., Doty, R. C., Allen, M. D., Chapman, A. (2014). Provenance Capture Disparities Highlighted through Datasets. In *Proceedings of 6th Workshop on the Theory and Practice of Provenance, TaPP'14*. Cologne, Germany.
- Creech, G., Hu, J. (2013). Generation of a New IDS Test Dataset: Time to Retire the KDD Collection. In *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 4487–4492. IEEE. doi:10.1109/WCNC.2013.6555301. Shanghai, China.
- Cruz, S. M. S. D., Campos, M. L. M., Mattoso, M. (2009). Towards a Taxonomy of Provenance in Scientific Workflow Management Systems. In *IEEE Congress on Services*, pp. 259–266.
- Cugola, G., Margara, A. (2012). Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys*, 44, 15:1–15:62. doi:10.1145/2187671.2187677.
- Cushing, W., Kambhampati, S., Mausam, Weld, D. S. (2007). When is Temporal Planning Really Temporal? In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI'07*, pp. 1852–1859. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. <http://dl.acm.org/citation.cfm?id=1625275.1625575>
- Daniel, D. O., Vítor, S., Marta, M. (2015). How Much Domain Data Should Be in Provenance Databases? In *Proceedings of the 7th USENIX Conference on Theory and Practice of Provenance, TaPP'15*, pp. 9–9. Berkeley, CA, USA: USENIX Association. <http://dl.acm.org/citation.cfm?id=2814579.2814588>

Bibliography

- Deolalikar, V., Laffitte, H. (2009). Provenance as data mining: combining file system metadata with content analysis. In *Proceedings of 1st Workshop on Theory and Practice of Provenance (TAPP'09)*.
- Draios Inc (2016). Sysdig. <http://www.sysdig.org/> (Last Accessed: 18/11/2016).
- Dumitras, T., Neamtiu, I. (2011). Experimental Challenges in Cyber Security: A Story of Provenance and Lineage for Malware. In *Proceedings of 4th USENIX Conference on Cyber Security Experimentation and Test (CSETT'11)*, p. 9.
- Faith, R. E., Wheeler, D. A., Kerrisk, M. (2016). Linux Programmer's Manual. <http://man7.org/linux/man-pages/man7/man-pages.7.html> (Last accessed: 11/01/2017).
- Fikes, R. E., Nilsson, N. J. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence (IJCAI)*, IJCAI'71, pp. 608–620. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.
<http://dl.acm.org/citation.cfm?id=1622876.1622939>
- ForensicsWiki (2015). MAC times. http://www.forensicswiki.org/wiki/MAC_times (Last Accessed: 17/11/2016).
- Fu, Q., Lou, J.-G., Wang, Y., Li, J. (2009). Execution Anomaly Detection in Distributed Systems Through Unstructured Log Analysis. In *Proceedings of the 9th IEEE International Conference on Data Mining, ICDM '09*, pp. 149–158. Washington, DC, USA: IEEE Computer Society. ISBN 978-0-7695-3895-2. doi:10.1109/ICDM.2009.60.
<http://dx.doi.org/10.1109/ICDM.2009.60>
- Garfinkel, T. (2003). Traps and Pitfalls: Practical Problems in System Call Interposition Based Security Tools. In *Proceedings of the 10th Annual Network and Distributed Systems Security Symposium*.

- Gehani, A., Tariq, D. (2012). SPADE: Support for Provenance Auditing in Distributed Environments. In Narasimhan, P., Triantafillou, P. (Eds.), *Middleware 2012*, vol. 7662 of *Lecture Notes in Computer Science*, pp. 101–120. Springer Berlin Heidelberg. ISBN 978-3-642-35169-3. doi: 10.1007/978-3-642-35170-9_6.
http://dx.doi.org/10.1007/978-3-642-35170-9_6
- Georgievski, I., Aiello, M. (2014). An Overview of Hierarchical Task Network Planning. ArXiv preprint arXiv: 1403.7425.
<https://arxiv.org/pdf/1403.7426v1.pdf>
- Ghoshal, D., Plale, B. (2013). Provenance from Log Files: A BigData Problem. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, EDBT '13, pp. 290–297. New York, NY, USA: ACM. ISBN 978-1-4503-1599-9. doi:10.1145/2457317.2457366.
<http://doi.acm.org/10.1145/2457317.2457366>
- Gil, Y., Miles, S. (2013). PROV Model Primer. <https://www.w3.org/TR/2013/NOTE-prov-primer-20130430/#the-complete-example> (Last accessed: 08/07/2016).
- González-Castillo, J., Trastour, D., Bartolini, C. (2001). Description Logics for Matchmaking of Services. In *Proceedings of KI2001 Workshop on Applications of Description Logics*. Vienna, Austria.
- Gotz, D., Zhou, M. X. (2009). Characterizing User's Visual Analytic Activity for Insight Provenance. *Information Visualization*, vol. 8, pp. 42–55.
- Govindan, K., Wang, X., Khan, M., Dogan, G., Zeng, K., Powell, G. M., Brown, T., Abdelzaher, T., Mohapatra, P. (2011). PRONET: Network trust assessment based on incomplete provenance. In *Proceedings of Military Communications Conference (MILCOM'11)*, pp. 1213–1218. doi:10.1109/MILCOM.2011.6127466.
- Green, T. J., Karvounarakis, G., Tannen, V. (2007). Provenance Semirings. In *Proceedings of the 26th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS'07)*, pp. 31–40.

Bibliography

- Groth, P. (2014). Provenance Week 2014. <https://sites.google.com/site/provbench/home/provbench-provenance-week-2014/schedule-provenance-reconstruction-provbench/> (Last accessed: 17/11/2016).
- Groth, P., Gil, Y., Magliacane, S. (2012). Automatic Metadata Annotation Through Reconstructing Provenance. In *Proceedings of 3rd International Workshop on the role of Semantic Web in Provenance Management (ESWC)*.
- Groth, P., Moreau, L. (2011). Representing Distributed Systems Using the Open Provenance Model. *Future Generation Computer Systems*, 27(6), 757–765. doi:10.1016/j.future.2010.10.001. <http://dx.doi.org/10.1016/j.future.2010.10.001>
- Grune, D., Jacobs, C. J. (1990). *Parsing Techniques - A Practical Guide*. Ellis Horwood, 1st edn.
- Hallè, S. (2017). From complex event processing to simple event processing. Tech. rep., Cornell University Library. <https://arxiv.org/abs/1702.08051>
- Hallè, S., Varvaressos, S. (2014). A formalisation of complex event stream processing. In *18th International Enterprise Distributed Object Computing (EDOC'14)*. IEEE. doi:10.1109/EDOC.2014.12. Ulm, Germany.
- Hargreaves, C., Patterson, J. (2012). An automated timeline reconstruction approach for digital forensic investigations. *Digital Investigation*, 9, Supplement, S69 – S79. doi:<http://dx.doi.org/10.1016/j.diin.2012.05.006>. The Proceedings of the Twelfth Annual Conference 12th Annual Digital Forensics Research Conference (DFRWS). <http://www.sciencedirect.com/science/article/pii/S174228761200031X>
- Hopcroft, J. E., Motwani, R., Ullman, J. D. (1979). *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 2nd edn.
- Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M. R., Li, P., Oinn, T. (2006). Taverna: a tool for building and running workflows of

- services. *Nucleic Acids Research*, 34, W729 – W732. doi:10.1093/nar/gkl320.
- Huq, M. R., Apers, P. M. G., Wombacher, A. (2013). ProvenanceCurious: A Tool to Infer Data Provenance from Scripts. In *Proceedings of the 16th International Conference on Extending Database Technology (EDBT)*, pp. 765–768. doi:10.1145/2452376.2452475.
- Huq, M. R., Wombacher, A., Apers, P. M. G. (2011). *Inferring Fine-Grained Data Provenance in Stream Data Processing: Reduced Storage Cost, High Accuracy*, pp. 118–127. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-23091-2. doi:10.1007/978-3-642-23091-2_11. Toulouse, France.
http://dx.doi.org/10.1007/978-3-642-23091-2_11
- Iwouters, Beltguese, Codeuser, Hymans (2014). Hime parser generator. <https://himeparser.codeplex.com/> (Last accessed: 01/08/2017).
- James, J. I., Gladyshev, P., Zhu, Y. (2011). *Signature Based Detection of User Events for Post-mortem Forensic Analysis*, pp. 96–109. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-19513-6. doi:10.1007/978-3-642-19513-6_8.
http://dx.doi.org/10.1007/978-3-642-19513-6_8
- Kalber, S., Dewald, A., Freiling, F. C. (2013). Forensic Application - Fingerprinting Based on File System Metadata. In *7th International Conference on IT Security Incident, Management and IT Forensics*, pp. 98–112. doi:10.1109/IMF.2013.20.
- Ketchen, D. J., Shook, C. L. (1996). The Application of Cluster Analysis in Strategic Management Research: An Analysis and Critique. *Strategic Management Journal*, 17, pp. 441–458. doi:10.1002/(SICI)1097-0266(199606)17:6<441::AID-SMJ819>3.0.CO;2-G.
- Khan, M. N. A., Chatwin, C. R., D., R. C. (2007). A Framework for Post-event Timeline Reconstruction Using Neural Networks. *Digital Investigation: The International Journal of Digital Forensics and Incident*

Bibliography

- Response*, 4(3-4), 146 –157. doi:10.1016/j.diin.2007.11.001.
<http://dx.doi.org/10.1016/j.diin.2007.11.001>
- King, S. T., Chen, P. M. (2003). Backtracking Intrusions. In *Proceedings of 19th ACM Symposium on Operating Systems Principles (SOSP'03)*.
- King, S. T., Mao, Z., Lucchetti, D. G., Chen, P. M. (2005). Enriching Intrusion Alerts Through Multi-Host Causality. In *InProceedings of Network and Distributed System Security Symposium (NDSS'05)*.
- Knutson, T. (2016). Filesystem Timestamps: What Makes Them Tick? <https://www.sans.org/reading-room/whitepapers/forensics/filesystem-timestamps-tick-36842> (Last Accessed: 17/11/2016).
- Ko, R. K. L., Will, M. A. (2014). Progger: A Efficient, Tamper-Evident Kernel-Space Logger for Cloud Data Provenance Tracking. In *Proceedings of the 7th IEEE International Conference on Cloud Computing (CLOUD'14)*. Anchorage, Alaska, USA.
- Laboratoire d'informatique Formelle (2015). Beepbeep. <https://liflab.github.io/beepbeep-3/features.html> (Last accessed: 01/08/2017).
- Lamport, L. (1978). Time, Clocks, and the Ordering of Events in a Distributed System. *Communication of the ACM*, 21(7), 558–565. doi: 10.1145/359545.359563.
<http://doi.acm.org/10.1145/359545.359563>
- Lee, K. H., Zhang, X., Xu, D. (2013). High Accuracy Attack Provenance via Binary-based Execution Partition. In *Proceedings of Annual Network and Distributed System Security Symposium*. San Diego, CA.
- Leskovec, J., Krevl, A. (2014). SNAP Datasets: Stanford Large Network Dataset Collection. <http://snap.stanford.edu/data> (Last accessed: 14/06/2016).
- Lincoln Laboratory (1998). DARPA Intrusion Detection Datasets - Massachusetts Institute of Technology. <https://www.ll.mit.edu/ideval/data/> (Last accessed: 16/06/2106).

- Ling, T. (2013). The Study of Computer Forensics on Linux. In *Proceedings of 5th International Conference on Computational and Information Sciences (ICCIS)*, pp. 294–297. doi:10.1109/ICCIS.2013.85.
- Linz, P. (2012). *An Introduction to Formal Languages and Automata*. Jones & Bartlett Learning, 5th edn.
- Lois, J. E. (2015). It can happen to you:know the anatomy of a cyber intrusion. http://www.navy.mil/submit/display.asp?story_id=91603 (Last accessed: 25/07/2017).
- Lou, J.-G., Fu, Q., Yang, S., Xu, Y., Li, J. (2010). Mining Invariants from Console Logs for System Problem Detection. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference, USENIXATC'10*, pp. 24–24. Berkeley, CA, USA: USENIX Association. <http://dl.acm.org/citation.cfm?id=1855840.1855864>
- Luckham, D. C., Frasca, B. (1998). Complex Event Processing in Distributed Systems. Tech. rep., Computer Systems Laboratory, Stanford University. CSL-TR-98-754. http://www.unix.com/pdf/CEP_in_distributed_systems.pdf
- Luckham, D. C., Kenney, J. J., Perrochon, L., Mann, W. (1998). The Stanford Rapide Project. <http://complexevents.com/stanford/rapide/examples/index.html> (Last accessed: 01/08/2017).
- Luttenberger, M., Schlund, M. (2014). Regular Expressions for Provenance. In *Proceedings of 6th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2014)*. Cologne: USENIX Association. <https://www.usenix.org/conference/tapp2014/agenda/presentation/luttenberger>
- Macko, P., Margo, D., Seltzer, M. (2013). Local Clustering in Provenance Graphs. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management, CIKM '13*, pp. 835–840. New York, NY, USA: ACM. ISBN 978-1-4503-2263-8. doi:10.1145/2505515.2505624. <http://doi.acm.org/10.1145/2505515.2505624>

Bibliography

- Macko, P., Seltzer, M. (2011). Provenance Map Orbiter: Interactive Exploration of Large Provenance Graphs. In *Proceedings of 3rd USENIX Workshop on the Theory and Practice of Provenance (TaPP'11)*. Berkeley, CA: USENIX Association. Heraklion, Crete, Greece.
- Magliacane, S. (2012). Reconstructing Provenance. *The Semantic Web - ISWC*, pp. 399–406.
- Magliacane, S., Groth, P. (2012). Towards Reconstructing the Provenance of Clinical Guidelines. In *Proceedings of 5th International Workshop on Semantic Web Applications and Tools for Life Science (SWAT4LS)*, vol. 952.
http://ceur-ws.org/Vol-952/paper_36.pdf
- Magliacane, S., Groth, P. T. (2013). Repurposing Benchmark Corpora for Reconstructing Provenance. In *Proceedings of the 3rd Workshop on Semantic Publishing*, pp. 39–50. Montpellier, France.
<http://ceur-ws.org/Vol-994/paper-04.pdf>
- Manning, C. D., Raghavan, P., Schütze, H. (2009). *Introduction to Information Retrieval*. Cambridge University Press, online edition edn.
<http://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf>
- McCracken, D. D., Reilly, E. D. (2003). Backus-naur form (bnf). *Encyclopedia of Computer Science*, pp. 129–131.
- Missier, P., Dey, S., Belhajjame, K., Cuevas-Vicenttin, V., Ludäscher, B. (2013). D-PROV: Extending the PROV Provenance Model with Workflow Structure. In *Proceedings of 5th USENIX Workshop on the Theory and Practice of Provenance (TaPP'13)*. Lombard, IL: USENIX Association.
<https://www.usenix.org/conference/tapp13/technical-sessions/presentation/missier>
- MITRE (2017). Common vulnerabilities and exposures.
<https://cve.mitre.org/> (Last accessed: 21/07/2017).

- Moore, S., Gehani, A. (2013). Declaratively Processing Provenance Metadata. In *Proceedings of the 5th USENIX Workshop on the Theory and Practice of Provenance (TaPP'13)*. Lombard, IL: USENIX.
- Moreau, L. (2010). The Foundation for Provenance on the Web. *Foundations and Trends in Web Science, Journal*, vol. 2, pp. 99–241.
- Moreau, L., Clifford, B., Freire, J., Futrelle, J., Gil, Y., Groth, P., Kwasnikowska, N., Miles, S., Missier, P., Myers, J., Plale, B., Simmhan, Y., Stephan, E., den Bussche, J. V. (2011). The Open Provenance Model core specification (v1.1). *Future Generation Computer Systems*, 27(6), 743–756. doi:doi:10.1016/j.future.2010.07.005.
<http://eprints.soton.ac.uk/271449/>
- Moreau, L., Groth, P. (2013). *Provenance: An Introduction to PROV. Synthesis Lectures on the Semantic Web: Theory and Technology*. Morgan & Claypool. doi:10.2200/S00528ED1V01Y201308WEB007.
- Moreau, L., Groth, P., Miles, S. (2010). Provenance Challenge Wiki. <http://twiki.ipaw.info/bin/view/Challenge/> (Last accessed: 09/11/2016).
- Moreau, L., Missier, P. (2013). PROV DM - W3C Working Group Note. <https://www.w3.org/TR/2013/REC-prov-dm-20130430/> (Last accessed: 13/06/2016).
- Nagappan, M., Vouk, M. A. (2010). Abstracting log lines to log event types for mining software system logs. In *Proceedings of 7th IEEE Working Conference on Mining Software Repositories (MSR)*, pp. 114–117. doi:10.1109/MSR.2010.5463281.
- Nari, S., Ghorbani, A. A. (2013). Automated malware classification based on network behavior. In *International Conference on Computing, Networking and Communications (ICNC'13)*. IEEE. doi:10.1109/ICCNC.2013.6504162. San Diego, CA, USA.
- National Security Agency (1999). Archived U.S. Government Approved Protection Profile - Controlled Access Protection Profile. <https://www.niap-ccevs.org/Profile/Info.cfm?id=4> (Last accessed: 11/12/2016).

Bibliography

- Nau, D., Cao, Y., Lotem, A., Munoz-Avila, H. (1999). SHOP: Simple Hierarchical Ordered Planner. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, vol. 2 of *IJCAI'99*, pp. 968–973. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.
<http://dl.acm.org/citation.cfm?id=1624312.1624357>
- Netresec (2010). Publicly available PCAP files - for Network Security and Forensics. <http://www.netresec.com/?page=PcapFiles> (Last accessed: 14/06/2016).
- Nies, T. D., Coppens, S., Deursen, D. V., Mannens, E., de Walle, R. V. (2012). Automatic Discovery of High-Level Provenance using Semantic Similarity. In *Proceedings of the 4th International Provenance and Annotation Workshop (IPAW'12)*, pp. 97–110. doi:10.1007/978-3-642-34222-6_8.
- Nies, T. D., Mannens, E., de Walle, R. V. (2016). Reconstructing Human-Generated Provenance Through Similarity-Based Clustering. In *Proceedings of the 6th International Provenance and Annotation Workshop (IPAW'16)*, IPAW 2016, pp. 191–194. New York, NY, USA: Springer-Verlag New York, Inc. ISBN 978-3-319-40592-6. doi:10.1007/978-3-319-40593-3_19.
http://dx.doi.org/10.1007/978-3-319-40593-3_19
- Nies, T. D., Taxidou, I., Dimou, A., Verborgh, R., Fischer, P. M., Mannens, E., de Walle, R. V. (2015). Towards Multi-level Provenance Reconstruction of Information Diffusion on Social Media. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, CIKM '15*, pp. 1823–1826. New York, NY, USA: ACM. ISBN 978-1-4503-3794-6. doi:10.1145/2806416.2806642.
<http://doi.acm.org/10.1145/2806416.2806642>
- OWASP (2013). Binary planting. https://www.owasp.org/index.php/Binary_planting (Last accessed: 11/01/2017).
- Papineni, K., Roukos, S., Ward, T., Zhu, W.-J. (2002). BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of*

- the 40th Annual Meeting on Association for Computational Linguistics, ACL '02*, pp. 311–318. Stroudsburg, PA, USA: Association for Computational Linguistics. doi:10.3115/1073083.1073135.
<http://dx.doi.org/10.3115/1073083.1073135>
- Pechanec, J. (2007). How the scp protocol works.
https://blogs.oracle.com/janp/entry/how_the_scp_protocol_works(Lastaccessed:3/7/2016)
- Probst, C. W., Hansen, R. R. (2008). An Extensible Analysable System Model. *Journal of Information Security Tech. Report*, 13(4), 235–246. doi:10.1016/j.istr.2008.10.012.
<http://dx.doi.org/10.1016/j.istr.2008.10.012>
- Probst, C. W., Hansen, R. R. (2013). Model-based Abstraction of Data Provenance. In *Proceedings of 6th USENIX Workshop on the Theory and Practice of Provenance (TaPP13)*.
- ProvBench (2012). A Provenance Repository for Benchmarking.
<https://github.com/provbench> (Last accessed: 13/06/2016).
- Rapid7 (2016). MetaSploit - World's most used penetration testing software. <https://www.metasploit.com/> (Last accessed: 23/06/2016).
- Roesch, M. (2006). Daemonlogger. <https://github.com/vrtadmin/Daemonlogger> (Last accessed: 23/06/2016).
- Rohrmann, T. (2016). Introducing complex event processing (cep) with apache flink. <https://flink.apache.org/news/2016/04/06/cep-monitoring.html> (Last accessed: 01/08/2017).
- Rosenblum, N., Zhu, X., Miller, B., Hunt, K. (2008). Learning to Analyze Binary Computer Code. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2, AAAI'08*, pp. 798–804. AAAI Press. ISBN 978-1-57735-368-3.
<http://dl.acm.org/citation.cfm?id=1620163.1620196>
- Schatz, B., Mohay, G., Clark, A. (2004). Rich Event Representation for Computer Forensics. In *Proceedings of the 5th Asia-Pacific Industrial*

Bibliography

- Engineering and Management Systems Conference (APIEMS)*. Gold Coast, Queensland, Australia.
- scikit (2010). sklearn clustering - Kmeans. <http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html> (Last accessed: 18/01/2017).
- Shiravi, A., Shiravi, H., Tavallaee, M., Ghorbani, A. A. (2012). Toward Developing a Systematic Approach to Generate Benchmark Datasets for Intrusion Detection. *Computers & Security*, 31(3), 357–374. doi: 10.1016/j.cose.2011.12.012. <http://www.sciencedirect.com/science/article/pii/S0167404811001672>
- Silberschatz, A., Galvin, P. B., Gagne, G. (2005). *Operating System Concepts*. John Wiley & Sons, 7th edn.
- Silberschatz, A., Galvin, P. B., Gagne, G. (2012). *Operating System Concepts*. John Wiley & Sons, 9th edn.
- Simmhan, Y. L., Plale, B., Gannon, D. (2005). A Survey of Data Provenance in e-Science. *ACM SIGMOD Record*, vol. 34(3), pp. 31–36. doi: 10.1145/1084805.1084812. <http://doi.acm.org/10.1145/1084805.1084812>
- Souilah, I., Francalanza, A., Sassone, V. (2009). A Formal Model of Provenance in Distributed Systems. In *Proceedings of First Workshop on Theory and Practice of Provenance (TaPP'09)*.
- Splunk Inc (2005). Splunk: Operational Intelligence Platform. <https://www.splunk.com/> (Last accessed: 21/07/2016).
- van der Stock, A., Goncalves, I. R., Correa, J. (2015). Owasp top 10 web vulnerabilities. https://www.owasp.org/index.php/OWASP_Top_Ten_Cheat_Sheet (Last accessed: 21/07/2017).
- Suriarachchi, I., Zhou, Q. G., Plale, B. (2015). Komadu: A Capture and Visualization System for Scientific Data Provenance. *Open Research Software*, 3(1). doi:<http://doi.org/10.5334/jors.bq>.

- Sysdig (2014). Universal System Visibility with Native Container Support. <http://www.sysdig.org/> (Last accessed: 23/06/2016).
- Tanenbaum, A. S., Bos, H. (2008). *Modern Operating Systems*. Pearson Education, 4th edn.
- The OpenStack Foundation (2010). OpenStack - Open source software for creating private and public clouds. <https://www.openstack.org/> (Last accessed: 23/06/2016).
- UIC (1999). KDD Cup 1999 Data - Information and Computer Science, University of California, Irvine. <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html> (Last accessed: 16/06/2106). The UCI KDD Archive.
- Vaarandi, R. (2002). SEC - a Lightweight Event Correlation Tool. In *Proceedings of Workshop on IP Operations and Management*. IEEE. doi:10.1109/IPOM.2002.1045765.
- Vaarandi, R. (2003). A Data Clustering Algorithm for Mining Patterns From Event Logs. In *Proceedings of the 3rd IEEE Workshop on IP Operations and Management (IPOM)*, pp. 119–126. IEEE. doi:10.1109/IPOM.2003.1251233.
- Vassos, S. (2012). Real-time Action Planning with Preconditions and Effects. *Game Coder Magazine*. <http://stavros.lostre.org/files/Vassos12ActionPlanning.pdf>
- Wai, C. T. (2002). Conducting a penetration test on a organisation. Tech. rep., SANS Institute <https://www.sans.org/reading-room/whitepapers/auditing/conducting-penetration-test-organization-67>.
- Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M. I. (2009). Detecting Large-scale System Problems by Mining Console Logs. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, SOSP '09*, pp. 117–132. New York, NY, USA: ACM. ISBN 978-1-60558-752-3. doi:10.1145/1629575.1629587. <http://doi.acm.org/10.1145/1629575.1629587>

Bibliography

- Zhao, J., Gomadam, K., Prasanna, V. (2011). Predicting Missing Provenance Using Semantic Associations in Reservoir Engineering. In *Proceedings of the 2011 IEEE 5th International Conference on Semantic Computing, ICSC '11*, pp. 141–148. Washington, DC, USA: IEEE Computer Society. ISBN 978-0-7695-4492-2. doi:10.1109/ICSC.2011.42. <http://dx.doi.org/10.1109/ICSC.2011.42>
- Zhou, W., Ding, L., Haeberlen, A., Ives, Z., Loo, B. T. (2011). TAP: Time-aware Provenance for Distributed Systems. In *Proceedings of USENIX Workshop on Theory and Practice of Provenance (TaPP'11)*. Heraklion, Greece.
- Zhou, W., Sherr, M., Marczak, W. R., Zhang, Z., Tao, T., Loo, B. T., Lee, I. (2010). Towards a data-centric view of cloud security. In *2nd International Workshop on Cloud Data Management (CloudDB), in conjunction with CIKM*, pp. 25–32. Toronto, Canada.
- Zhu, X., Song, S., Lian, X., Wang, J., Zou, L. (2014). Matching heterogeneous event data. In *ACM SIGMOD International Conference on Management of Data, SIGMOD '14*, pp. 1211–1222. New York, NY, USA: ACM. ISBN 978-1-4503-2376-5. doi:10.1145/2588555.2588570. <http://doi.acm.org/10.1145/2588555.2588570>