Traceback: A Forensic Tool for Distributed Systems

Sushant Dinesh, Sriram Rao and K. Chandrasekaran

Abstract In spite of stringent security measures on the components of a distributed system and well-defined communication procedures between the nodes of the system, an exploit may be found that compromises a node, and may be propagated to other nodes. This paper describes an incident-response method to analyse an attack. The analysis is required to patch the vulnerabilities and may be helpful in finding and removing backdoors installed by the attacker. This analysis is done by logging all relevant information of each node in the system at regular intervals at a centralised store. The logs are compressed and sent in order to reduce network traffic and use lesser storage space. The state of the system is also stored at regular intervals. This information is presented by a replay tool in a lucid, comprehensible manner using a timeline. The timeline shows the saved system states (of each node in the distributed system) as something similar to checkpoints. The events and actions stored in the logs act on these states and this shows a replay of the events to the analyser. A time interval during which an attack that took place is suspected to have occurred can be analysed thoroughly using this tool.

Keywords Forensics · Incident response · Distributed system

S. Dinesh (✉) · S. Rao · K. Chandrasekaran
Department of Computer Science and Engineering, National Institute of Technology Karnataka, Surathkal, Mangalore, India
e-mail: sushantdinesh94@gmail.com

S. Rao
e-mail: sriram.rao@ieee.org

K. Chandrasekaran
e-mail: kchnitk@ieee.org

A. Nagar et al. (eds.), *Proceedings of 3rd International Conference on Advanced Computing, Networking and Informatics*, Smart Innovation, Systems and Technologies 44, DOI 10.1007/978-81-322-2529-4_2

1 Introduction

Computer Forensics refers to the field involving inspection and analysis of digital storage to extract information about the computer system and any changes that it may have undergone through processes running on the system. Forensic methods are largely used to gather information about a system after it suffers an illegal attack that may have resulted in compromised data or denial of service, or both, among other consequences. While this information is helpful as legal evidence, it can also be used to study the loopholes in the network and find methods to fix them.

In a distributed system, communication between components is secured using cryptographic methods. A client node (that is making the request) needs to be authenticated by the receiver node before processing the request. On a node, untrusted code from an external source that has to be executed is run with limitations, prohibiting access to most resources without authorisation. In spite of these measures, the system may be attacked by a malicious agent. In such cases, forensic tools are useful in analysing the method and extent of exploitation, and the information accumulated can be used to patch the system. Any backdoors included in the network by the attacker can also be found and removed. When regular system checkpoints are made, these can be used to restore the compromised system to a known uncompromised state. Currently, analysis involves going through millions of lines of logs from different components of the system manually. This is tedious, time-consuming and there are chances that investigators might miss a crucial part of the attack. A clever attacker might also delete the logs during his attack thereby making them useless in the analysis. With several missing pieces of information, analysing the attack and estimating damage would be very difficult. Due to lack of information about the type of attack, developers must manually go through the whole code to find the vulnerability used. All this suggests that we must have a more robust forensic analysis tool designed to operate on distributed system.

In this paper we conceptualise a forensic tool, Traceback, that helps analyse attacks by simulating events that occur starting from a chosen system state. This makes use of the logs that are kept by various processes that run on each system, and by the systems themselves. These logs are retrieved from storage and a timeline of events is constructed to understand the sequence of events. This timeline also includes regular system states that are stored as checkpoints. When an analyser wants to check a particular time interval for suspicious behaviour, he can choose an initial state and have the logged events perform transitions to this state up to a required point of time.

The paper has been categorised as follows. Section 2 describes related work. Section 3 gives an overview and a brief description of the proposed work. Section 4 details the work and responsibilities of the logging agent. Section 5 discusses the choices available for the central store. Section 6 outlines the tool and useful features. Section 7 concludes the paper.

2 Related Work

Topics relevant to this paper that are well-studied include efficient compression of files, specifically log files, and good forensic tools to inspect networks. Effective and thorough tools have been made which monitor the network packets at a particular computer system. In a distributed system, a more suited tool would allow detailed monitoring of packets exchanged within the system and with external sources. With respect to log files, there are papers which detail methods proposed to compress log files generated by particular tasks and discuss their compression ratio and speed.

2.1 Compression of Logs

Compression of logs a very well studied topic. Here we consider two papers which talk about two different compression mechanisms to improve the compression ratio and compression speed when dealing with logs. In "Lossless compression for large scale cluster logs" [1], Balakrishnan and Sahoo develop a compression algorithm to compress logs generated by Blue Gene/L supercomputer. Their compression algorithm has limited scope and produces best results only on logs generated by this supercomputer. In "Fast and efficient log file compression" [2], Skibiski and Swacha develop a generic compression scheme for logs with five different variants of it, each varying in degree and speed of compression. Both the papers have a significant improvement in compression ratio and compression speed as compared to general purpose compression schemes like gzip, bzip etc.

2.2 Forensic Tools Used in Network Security

Extensive work has been done in making effective methods to provide network security. Many of these methods, such as those outlined in "Distributed agent-based real time network intrusion forensics system architecture design" [3] are for intrusion detection at the time of attack—to detect real-time attacks. "Achieving Critical Infrastructure Protection through the Interaction of Computer Security and Network Forensics" [4] also concentrates on acquisition of forensic data for real-time security purposes. The Incident Response Support System outlined in [5] is a tool for automatic response to attacks on a system. The attack being made is compared to an information base of known exploits and the response used in the known case is adapted to respond to the current attack. All these procedures are aimed at a general computer system connected to a network. Our paper concentrates on analysis of the attack to gain complete knowledge of the vulnerability in the environment of a distributed system.

3 System Overview

Figure 1 describes an overview of the final Traceback tool. The Traceback tool has a timeline which allows forensic investigators to go back to the system state during the time of the attack and analyse the actions of the attacker in detail. Different logs from various components of the distributed system are put together by the tool to simulate the situation during the time of attack. This helps the investigators understand the exact vulnerability in the system and take the required measures to patch it up. Additionally, as the alteration in the state of the system due to the attack is logged, any backdoors, worms and residual files left behind by the attacker can be cleaned up and also used as evidence for further forensic analysis.

Agents are used to "checkpoint" the system states, aggregate logs from the component, compress and then send them to the central log storage. This allows us to have a small local log storage and helps us maintain all logs in a central place.

A system state may include information like number of processes running, CPU usage, Memory usage, network usage etc. Information regarding each process is stored in the initial state. Each successive checkpoint only stores the changes from this initial state, in a Git inspired "diff"-like manner. Similar process is followed for important directories in the file-system. This allows us to think of logs as transformations on a system state.

Logs and checkpoints collected by the agents are compressed before sending to the central log storage. Compression is done by the agents to avoid excessive network usage as log files tend to get very big. We can achieve a high compression ratio as log files tend to have a lot of repeated information. The central log storage collects logs from all the sources and stores it for future retrieval.

We proceed with a detailed description of the following points:

- Logging agent
- Compression of logs
- Central log storage
- Traceback application

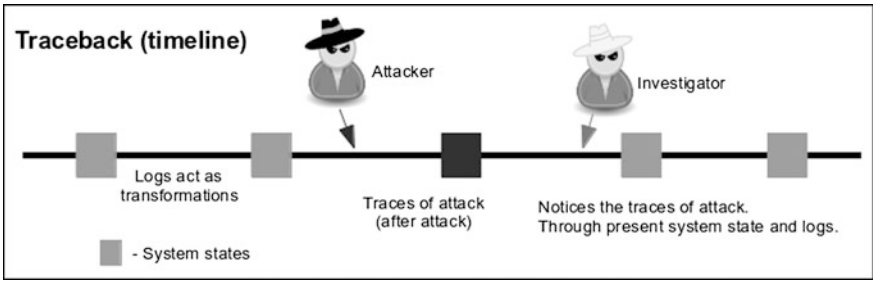

Fig. 1 Overview

4 The Logging Agent

Each component of a distributed system that is open to external communication has an instance of the logging agent running. Briefly, the responsibilities of the logging agent are to save the system state at regular intervals of time, and to log all relevant local events at the system. If there is a centralised storage of all logs, this agent compresses the logs and periodically sends them to the central storage.

4.1 Access to the Agent and Logs

The agent process must run at a higher privilege than other processes so that it is not terminated by other (possibly user-level) processes. Also, the agent must have enough access to collect all relevant logs from the system, and other system-related information such as a list of running processes and their details. The log files written by the agent should be accessible only by the agent and the root of the system so that user-level processes do not interfere or contaminate the logging process. Processes running untrusted code must also not have access to these logs. These reasons point to elevating the agent process to a high privilege.

4.2 Description of a System State

To analyse an attack on the system, investigators must look at the state of the system before the attack and analyse the changes made to this state during and after the attack interval. The changes made to the system are studied to single out those initiated by the attacker as part of the exploit and accurately define the attack interval and method. To aid this procedure, it is beneficial to save the states of each system in the distributed network at regular intervals. System snapshots have large storage size, so it is not practical to intermittently take snapshots of each system. Instead, specific data such as a list of the details of all running processes and the general file system information can be stored. The information about the file system tells us what files have been added or deleted, and when a file was last modified. Process-relevant information that can be stored includes CPU usage, Memory usage, Process ID and Parent Process ID, and Process running time.

After an initial system state containing complete information as outlined, newer save states can be stored as a difference from this initial state. This again reduces the storage space occupied. It also helps visualise the changes made to the initial system state as state transitions, making the changes easier to understand. The differences that need to be considered are changes in the file system, and changes in the processes that are running. Every change in the details of running processes need not be considered as the CPU and memory usage are not strictly constant, and the

process running time is different at different instants of time. Only a significant difference in usage of resources (above a configurable threshold) can initiate noting this in the system save state.

This information about system states and the logs made can be effectively represented in the form of a timeline. The events in the log files can be simulated on the saved system states to show a replay of events that makes it simpler for the investigators to understand the sequence of developments during a suspected attack interval.

4.3 Information Collection from Different Logs

To adequately represent the developments on a system state, all significant events must be logged by the agent. These logs are then compressed before storage at the central store. The information that needs to be collected for the logs includes:

Network Traffic When an attack is made on a node, traffic from external agents to a node in the distributed system holds information about the attack. To capture information about the traffic from and to a node, tools such as libpcap or snort can be used. These tools also allow filtering based on protocol and IP addresses since all packets are not required to be stored. Packets of some protocols may have low chances of carrying malicious code.

Internal Communication Internal communication between nodes of the distributed system is generally logged by the middleware that specifies communication procedures for the system. These logs can be used for forensic purposes, in cases where a compromised node may communicate with other nodes of the system to try and spread the attack.

Running Processes At regular intervals, the agent makes a check on the running processes in order to log changes. The process-related information stored is elucidated in Sect. 4.2. This information is needed as part of the system state.

System Logs and Authentication Logs The logs that are kept by the system can be vital in the analysis and search of information about an attack that the system suffered. The event logs of each node have information about all operations carried out and events that occurred, among other useful information. These contain the details of the malicious operations carried out, except in cases where a cautious attacker erases relevant information from these logs.

Authentication logs contain information about the Authentication Provider and the time of login of all users on a system. It follows that remote login procedures used, such as SSH or Telnet, will also be logged in this file. Thus, the auth logs collected from each node can reveal information about login by an unfamiliar user or at an odd time.

Database Logs Any change made to a database existing on the node is stored in the database logs. These can hold information about back-doors installed in the database by the intruder. When the distributed system uses a database to store necessary information, it is necessary to ensure its security so as not to allow unauthorised changes to the information, or a loss of database entries.

4.4 Compression of Logs

Compression of the logs generated is a very important responsibility of the agent. Better compression ensures efficient storage of logs and better network utilisation as the data to be transferred to the central store would be smaller.

Table 1 compares various general purpose compression mechanisms. It is clear that gzip the fastest among the four and achieves a compression of 93.51 %. The other algorithms are considerably slower and offer only a marginal difference in size of the logs after compression.

Compression of logs is a well studied subject and several journals have been released about the same. In [1] Balakrishnan and Sahoo talk about a specialized lossless compression for large scale cluster logs. In the paper the authors talk about exploiting the redundancy in the log files to allow a large amount of compression. Compression is done in multiple stages to exploit redundancy in information at various stages and maximize the compression. The authors were able to achieve a 28.3 % better compression ratio and 43.4 % of improvement in compression time as compared to standalone compression utilities. However, the algorithm is very specific to the logs their systems generate.

In [2] Skibiski and Swacha talk about a custom compression scheme and five different variants of it. Using the faster variant, the log files were transformed to be 36.6 % shorter than the original files compressed with gzip. Using the slower variant, the log files were transformed to be 62 % shorter than the original files compressed with gzip, and 41 % shorter than the original files compressed with bzip2. The advantage of this scheme is that there is no pre defined format for the log and the compression will work for all generic logs.

We could also develop a custom compression method based on our logs to achieve a better compression. However, this is not a subject of this paper and more of a scope for future work. We will use a general purpose algorithm like gzip for our application.

Table 1 Comparison of popular compression algorithms. *Source* [6]

Compression algorithm	Compression ratio	Bytes/s
GZIP	93.51	0.5191
BZIP	96.52	0.2785
7-Zip	97.50	0.2002
LZXQ 0.4	95.84	0.3325

5 Central Storage

The central store is responsible for aggregating logs from various components and store it for further analysis. The central store should be highly secure as it stores all vital information regarding the behaviour of the distributed system. An attack on the central store could lead to loss of large amounts of information. Ideally, the central store should not be a part of the distributed system and must not be accessible from anywhere outside the organization's network. The only means of communication between the distributed system and the central store must be through the agents monitoring them. Confidentiality and integrity of the data sent over the network by the agents can be ensured by using a secure transfer mechanism such as SCP or SFTP. SCP and SFTP rely on security provided by SSH to transfer. SSH is very secure as long as we use non-trivial passwords or authentication is allowed only through SSH keys. Hence we will use this in our application. There are two ways we can go about storing the logs: a database like MongoDB [7, 8] to store logs, or in the file-system like normal files.

5.1 Using a Database—MongoDB

MongoDB is an open-source document database, and the leading No-SQL database. Written in C++, MongoDB offers features like:

- Full indexing—Index any attribute
- Replication support—Mirror across LAN's and WAN's
- Auto sharding—Horizontal scaling
- Querying—Feature rich query support
- Map/Reduce—Flexible aggregation and data processing

Using a database like MongoDB to store the logs would be easier for using the information in applications. MongoDB has adapters for various languages and hence it would be convenient to use while building the Traceback application. MongoDB is a full fledged database management system, this means the search and retrieval of documents is highly optimized and can provide a better performance than a traditional file-system without any such optimizations. MongoDB is fast, secure and also supports sharding which can be leveraged when the data set becomes very large. Having built-in support for replication allows us to maintain multiple replicas of the database. This would be useful in case of an attack on the central store.

5.2 Using a File-System

For a smaller system with a relatively smaller amount of logs a file-system maybe a reasonable way to store these logs. File-system access is easier and can be used by

anyone without having knowledge of a particular database. Hence if manual, regular checks of logs are needed, then a file-system would suit this need better. A custom implementation for the specific application allows developers to have a finer control over its performance and storage. As the number of request for retrievals for logs is expected to be small (only in case of attack or suspected breach) using a database might not be necessary. As the size of log store grows we may have to consider a distributed file-system such as Hadoop to store the logs in a distributed manner.

MongoDB ensures integrity of data and also has built-in security mechanisms and fault tolerance. Additionally, it is easier to build applications to use MongoDB rather than a normal file-system. Due to the advantages of storing logs in MongoDB over a file-system we decided to go with using MongoDB to store logs for this particular application.

6 Traceback Tool

Traceback can be used to obtain a replay of events that occurred in the distributed system during some interval of time. It uses the information stored in the logs, along with the system states that were saved, to provide an analytical visualisation of events. The logs obtained from the store are decompressed and then used for replay purposes.

6.1 Timeline Format

To make the visualisation easy, the tool presents the system states and changes on a timeline. The system states are visualised as points on the timeline, and all events that change the state of the system can be represented as transitions on this state. The analyser can choose to view the state of the distributed system at a particular point of time on the timeline, and simulate the events that occurred from this instant to another chosen instant. Important instants of time can be bookmarked for easy reference, and any comments that follow from analysis can be added to the timeline to help future study.

6.2 Modules

Traceback also allows forensic investigators to write and include modules. Modules are well known attack vectors which can be used by other investigators to quickly search if a particular attack was used against the system. This will allow investigators to fly through their analysis and know exactly what was exploited on the

system. The modules are stored centrally on Traceback's repository which makes it easy for analysers to download and include them for their analysis.

6.3 Playback Mechanism

When the starting state is chosen and the replay is begun, all events occurring are shown to the analyser. All the information about events and system states is obtained from the logs at the central store. The simulation details packets being sent and received from a node in the distributed system to an external node, as well as packets interchanged between nodes within the system. Processes changing the states of each node are also shown. The simulation on all nodes in the system is shown together. On finding suspicious action in any node, the analysers can concentrate on—or "zoom" into—the behaviour of that node.

Filters can be used to observe specific events at a time. If only network interactions are to be analysed, or only events from a particular log (like the authorisation log) the appropriate filter is applied and the playback is observed.

7 Conclusion and Future Work

Avoiding attack is of first and foremost importance when we design a distributed system. But as the system grows and becomes more complicated, security vulnerabilities may creep in. Malicious users may exploit these flaws in the system to gain access to sensitive information or to cause damage. For such situations, we need a robust tool to analyse the extent of damage caused by the attacker, remove all back-doors and understand the actual vulnerability and the system state that allowed such an attack. This tool helps investigators as manually sifting through logs to find the source of attack is tedious and will lead to delay in patches. To summarise, this paper discusses the implementation of a forensic analysis tool in a distributed system to investigate the system post attack. It uses agents to collect information about the status of various components of the system and stores it centrally. This information is retrieved by the tool, decompressed and displayed to the investigator in a convenient manner so as to make the analysis fast and effective. We have also discussed about some of the issues which are faced during the design of such a tool and presented a few ways to overcome these issues.

Future work that can improve this tool can involve the following details. Sharding the central log and storing the primary log in a distributed manner is an option to resolve the issue of efficient storage on large distributed systems. The tool can be made more intuitive in the sense that it analyses the information provided and marks suspicious locations on the timeline for the analysers to scrutinise. The security of the central log store and the agent processes is also a matter that needs to be looked into thoroughly.

References

1. Balakrishnan, R., Sahoo, R.K.: Lossless compression for large scale cluster logs. In: 20th International Parallel and Distributed Processing Symposium (IPDPS 2006), IEEE (2006)
2. Skibiski, P., Swacha, J.: Fast and efficient log file compression. In: CEUR Workshop Proceedings of 11th East-European Conference on Advances in Databases and Information Systems (ADBIS 2007) (2007)
3. Ren, W., Jin, H.: Distributed agent-based real time network intrusion forensics system architecture design. In: 19th International Conference on Advanced Information Networking and Applications (AINA 2005), vol. 1, IEEE (2005)
4. Hunt, R., Slay, J.: Achieving critical infrastructure protection through the interaction of computer security and network forensics. In: 2010 Eighth Annual International Conference on Privacy Security and Trust (PST), IEEE (2010)
5. Capuzzi, G., Spalazzi, L., Pagliarecci, F.: IRSS: Incident response support system. In: International Symposium on Collaborative Technologies and Systems (CTS 2006), IEEE (2006)
6. Benchmarks for popular compression algorithms. http://www.maximumcompression.com/data/log.php
7. MongoDB Official Documentation. http://www.mongodb.org/
8. Using MongoDB to store logs. http://docs.mongodb.org/ecosystem/use-cases/storing-log-data/