Formalizing Apache Access Logs into Forensic Lucid

Phani Pramod Kruttiventi
p_krutti@encs.concordia.ca

Andy Pitcher
andy.pitcher@mail.concordia.ca

Pranesha Mardhala
p_mardh@encs.concordia.ca

Serguei A. Mokhov
mokhov@encs.concordia.ca

***Abstract*—Computer Forensic science deals with search for an evidence for reassembling the actions, which led a system from a secure state to a moment where an intrusion was detected. For this process, the main source for all the incidents happened is Log files. In this project we have to parse the raw logs we obtained from Apache to Forensic Lucid observations.** FORENSIC LUCID **is used to address the complexity that an investigator does to construct a finite-state machine (FSM) and model transitions. We propose a method in-order to do formalization of forensic analysis of the logs retrieved. Moreover, we have to examine the admissibility of forensic analysis and the verification issues. The translation of extracted logs into formal forensic lucid expressions are to be done for further analysis.**

***Index Terms*—Apache HTTP server, logging,** FORENSIC LUCID

I. INTRODUCTION

A. Motivation

The Apache server logging plugin and encode is an effort under a larger project of implementation and verification of FORENSIC LUCID encoders [1] for different popular server software as plug-ins or modules to provide functionality to the said servers to log their data directly in FORENSIC LUCID and/or write translation tools (scripts) to translate existing logs into FORENSIC LUCID, e.g., any two from Apache, Tomcat, Dovecot, Syslog, BIND [2], `iptables` [3], [4], `sshd`, JSON data, and others. Apache HTTP server is a popular web server software widely deployed; naturally its logs may contain a lot of relevant evidence in the case of a forensic investigation.

The encoder verification sub-project may involve Isabelle/HOL [5], [6] to show that any log or data structure translation done is faithful enough and no meaning loss or corruption occurs.

The idea is to present the evidence in a common representation for computable event reconstruction where the forensic knowledge can be represented and reasoned about in the same formal format of FORENSIC LUCID.

B. Overview

In Section II we provide a detailed background of our research into the relevant systems, components, and FORENSIC LUCID. In Section III we describe our approach to encoding of the Apache access logs along with some example results. In Section IV we conclude with current project state, its limitations, and future work.

II. BACKGROUND

A. What is Forensic Lucid?

LUCID is a family of programming languages which are built on intensional logic, which involves context and demand-driven parallel computational model [7]. The word intention explains about the dimensions for which we can state whether a statement is *true* or *false*. A program in LUCID contains expressions, which have many subexpressions needed to evaluate at certain context. Each statement in LUCID can be written as an equation defining the data flow. FORENSIC LUCID is a successor of the LUCID programming language, designed for specifying and encoding evidence for digital forensics investigation [8].

Forensic Science Elaborates regarding application of science to solve criminal investigations. Intentional Logic is considered for automating cyberforensics and bringing dimensions and context to programs. Intensional logic adds dimensions to logical expressions, thereby a non-intensional logic can be seen in all possible dimensions. FORENSIC LUCID is used to model the evidential statements by showing the observations and evidences as a higher-order context hierarchy. Execution of a FORENSIC LUCID program will expose the possibility of a proposed claim along with the events in the middle.

Some initial FORENSIC LUCID encoded examples for a MAC Spoofer Investigation appeared in [1], [9], [10], and some of which are publicly available at https://github.com/smokhov/atsm/tree/master/examples/flucid.

B. Apache HTTP Server

Apache [11] is an HTTP server and provides specific functionality to each modules. The main objective of Apache is to investigate an existing Apache WebServer, which carries multi tasking. Some of the advantages of Apache HTTP server are handles large amount of traffic and serves different types of content. Converting Apache logs into FORENSIC LUCID makes sense since it is the most used web server [12]. Server has refer to the processing unit (computer/hardware or to its software), instead of showing error message when the request was not processed. This means that a large amount of illegal websites/content investigation will be based on Apache web server. Third party modules are supported by Apache Server, later this modules are added as features in the new version of Apache 2. Open source applications are easily integrated to Apache. To sum up all Apache is more flexible, efficient and powerful.

1) Logs: A Log is a record, which contains all the details regarding an incident. They play a pivotal role in the part of an investigation. Logs are automatically produced along with timestamp regarding an event. For example in a web server, logs consists of all individual requests that were made from a website. Coming to logs collected from a server, the administrator can get data of how many visitors made requests, from which domain, how many number of requests made for each page and usage patterns (time of the day, week, month or year). The information in logs will give the investigator a detailed understanding of the events which caused an incident, to what extent the incident had its impact, which is important for mitigation [13]. Coming to our project, the logs retrieved from the dataset had these attributes: IP Address of the system, Timestamp (Date, Month, Year and Time), method used, HTTP path and code.

The following are the various parameters, that were retrieved from the logs obtained.

1) IP Address: IP address is assigned specifically for every device, which uses Internet Protocol for communication in a Network. It serves like an identity for a device.
2) Identifier: Every log has an unique identifier. Since there are many sequences of observations, it is impossible to identify the logs. Unique code is generated with the combination of letters, numbers and symbols as an alpha- numeric code is called as identifier.
3) Timestamp: Timestamp is also known for POSIX TIME or UNIX Epoch time, is an encoded data for describing sequence of time in the logs, which shows time, date, month, and year. Sometimes it also shows the time for fraction of seconds in the event. It is used to track and sort the information in the computer based on the sequence of time. The format of timestamp is `DD/MM/YYYY: HH:MM:SS`.
4) Request: Apache server records all incoming requests. For example, if a client made a request to server for a specified website, logs store the information of request history. The access log format is configurable. The location and content of the access log are controlled by the custom log directive. In the logs we obtained, the method used for retrieving is “GET”.
5) HTTP Code: HTTP response status code indicates whether a specific HTTP request has been successfully completed. HTTP path variable will be returning only the path of the URL. These codes are issued by a server, in response to a request made by a client to the server device. Responses are grouped in five classes: they are informational response, successful responses, redirects, client errors, and server errors. These codes were maintained in an official registry of HTTP status codes by Internet Assigned Numbers Authority (IANA). Some of the HTTP status codes are given here:
   - Client errors:
     - 400 – Bad Response
     - 401 – Unauthorized
     - 402 – Payment
     - 403 – Forbidden
     - 404 – Not Found
     - 429 – Too Many Request
   - Server errors:
     - 500 – Internal Server Error
     - 501 – Not Implemented
     - 502 – Bad Gateway
     - 503 – Service Unavailable
6) Size Of The Object: This is denoted as `object_size`. It considers the size of the environments, that are associated with an object. This is used to allocate memory space an object occupies and helps us in understanding how objects are stored in the memory.
7) Referer: For each request to website, Apache will write an entry in the referer log file including the landing page and the referer strings. Referer is voluntarily sent in HTTP headers from the client/browser to the server, so the server logs can be used to determine things like how people are finding your site etc., the problem is created. Since HTTP header is sent by client, it cannot be trusted. A malicious client can forge this information.
8) User Agent: `user_agent` parameter serves like a client signature. This field logs the browser signature of the client, which accesses a website. It refers to determining whether items of possible evidence are unique so that they may be linked to a particular individual or event. Within this, the items must be evaluated and interpreted.

2) Apache Piped Log: Apache tool logs data directly into the log files. Both versions of Apache, i.e., Apache 1.x, 2.x have an option of enabling an option called “Piped Logging”. This option is powerful, since it has more flexibility of logging, without adding code to the main server. For writing logs to a pipe, we have to replace the filename to be converted with a pipe character “`|`”, followed by name of the executable which accepts logs on its input.

C. Modeling The Investigation

1) Event: Refers to determine the items in the computer as a possible evidence, which are unique, so that it may be linked to an individual. In a cybercrime investigation, there might be a sequence of events. It gives the exact view of the crime for the investigator. The finite set of events is identified as I.

2) State: State is defined as a collection of events. The combination of events and states are referred as transition function. These systems are mapped to the finite state machine. Observed state by investigator look like: $(1, u, t2)$ whereas 1 as data length, $t2$ as threats in slack, u as unrelated.

3) Observation Sequence: An Observation Sequence (os) consists of all observations in a chronological order [14]. A sample os will be like this:

```
os=(observation A, observation B, ...)
```

This set represents an eye witness story without any gaps in between. All the observations are laid one-after-the-other, so that the next observation begins after the previous observation gets completed. If there are any gaps in the sequence, they are shown as no-observations. This is represented as $\$ = (CT, 0, Infinitum)$. $Infinitum$ is an integer constant, which is greater than the length of any calculation. Observation Sequence is partitioned in such a way that the length of property run (pr) is negligibly equal to length of os. Subsequently, every observation of os will be explained by each element of pr [14].

4) Formalization of the Evidence: In an incident, we obtain a couple of evidences. Every evidence will have its own story. They will be obtained in different environment, timestamp. It is a tedious process to relate all the evidences one-after-the-other. So, it is our responsibility to reconstruct the event by combining the stories narrated by different evidences, to make sure that the description obtained is as accurate as possible. Formalization of evidence is depends on length of observation sequence (run).

5) Evidential Statement: An evidential statement is a collection of observation sequences. Each observation sequence has its own version of the story. All the sequences put together will be listed here, although ordering is not given much importance. An explanation of Evidential statement is a sequence of partitioned runs, such that all elements are partitioning of the same run. It should be a story, which details all the versions of the story. If this set doesn't have any explanations, it is said to be inconsistent.

6) Testing Investigative Hypotheses: The motive is to determine which statement in the hypothesis is most likely to be true.

1) This is done to show that, the claims made against the event are true. The person who is investigating should show the evidences in the form of an explanation, which agree with the claim.
2) Formulating the claim in observation sequence and try to find explanation that agrees with evidence and claim or show that there are no explanations proving that the claims are false.
3) In-order to verify one's claim, we test the final evidential statement for finding an explanation for it.
4) In order to verify the program conversion, in a dynamically manner, by using the piped log option given by `mod_log_config`, we assume that contributing to Apache software foundation is required, since the code will be verified before being embedded to Apache install. This is the case of the program `rotatelogs` that uses the piped log option, in order to rotate the logs without having to stop/start the webserver.

This is relevant in a court case, where the the tools used to provide the digital evidence, has to be verified, to provide integrity.

III. Methodology

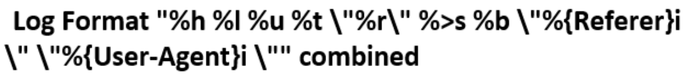
A. Conversion Of Access Logs Into Forensic Lucid Observations

converter_apache2_Flucid.c: Converting statically and dynamically access logs into FORENSIC LUCID Observations. `Converter_apache2_Flucid` is a program implemented in the C language, which can convert a generated Apache access log, statically, by reading a passed log file as argument, and format each access records (lines), into a FORENSIC LUCID observation. This main functionality can be used dynamically when combined to Apache `mod_log_config`, where the access logs will be formatted in FORENSIC LUCID observation, by using the piped log.

C language has been used to achieve the program implementation since it is powerful and fast. It runs close to the CPU, so any log conversion becomes quicker. The other advantage is that any POSIX based operating system is built to run this language's compiled binary natively, which provide more source code portability, for our converter, to be directly attached to Apache, which also relies on C.

B. How It Works?

Statically: By providing the `-F` options along with the Logfile. The program relies on the following Apache access log format, with the default attributes generated by `mod_conf_log` and described above, that comes with as standard configuration (in our case `httpd` version 2.4.6):



```
Log Format "%h %l %u %t \"%r\" %>s %b \"%{Referer}i
\" \"%{User-Agent}i \"" combined
```

Fig. 1. Apache Access Log Format

The format in Figure 1 will generate the following type of logs, which will be written into access_logfile.

Raw Format is exemplified in Listing 1.

```
172.16.16.5 -- [05/Mar/2018:20:06:02 -0500] "POST /user/
     passwordforgotten.php HTTP/1.1" 200 5174 "http://
     example.com/user/passwordforgotten.php" "Mozilla/4.0
     (compatible; MSIE 5.5; Windows NT)"
```

Listing 1. Forensic Lucid Format Example Output

converter_Apache2_Flucid.c.

In order to treat each attribute implements a structure variable (struct `ApacheLog`) containing each attribute as int or string variable, two functions are used. They are `display_flucid_obs()` and `main()`.

`main()`:

This is the main function that takes a logfile as argument passed to the program.

For each line:

1) Apply the regex [15], as shown in Figure 2
2) Assign each regex's chunk of data to the given and ordered attribute (struct variables)
3) Call `display_flucid_obs()` with the filled structure `ApacheLog`
4) This will display (stdout) the attributes and format them into an observation (line number)

```
%s %s %s [%s -%d] \"%s %s %[HTTP/1.0-1]\" %d %d
%s %[^\t\n]
```

Fig. 2. Regex

C. Forensic Lucid Format

Our program then processes the `ApacheLog` struct to produce an equivalent FORENSIC LUCID output, as exemplified in Listing 2. This particular encoding structure is inspired from the MAC Spoofer Analyzer case encoding in [1].

```
observation access_o_1 = ([src−ip:”172.16.16.5” , access−
      date:”05/Mar/2018:20:06:02”
  timezone:”0500” , http−identd:”−” , http−userid:”−” ,
      http−method:”POST”
  http−path:”/user/passwordforgotten.php” , http−protocol:”
      HTTP/1.1” , http−code:”200”
  object−size:”5174” ,http−referer:”http://example.com/user
      /passwordforgotten.php”
  user−agent:”Mozilla/4.0 (compatible; MSIE 5.5; Windows NT
      )”] 1 , 0 , 1 . 0 ,”05/Mar/2018:20:06:02”
```

Listing 2. Forensic Lucid Format Example Output

A sample invocation of our tool is shown below with the first two observations logged as illustrated in Listing 3.

```
converter_FLucid git:(master)  \
  ./converter_apache2_FLucid \
    ../access_log-20180305 \
    | head -2

FLucid observations: 6896
```

```
observation access_o_1 = ([src−ip:”127.0.0.1” , access−
      date:”05/Mar/2018:19:22:52” , timezone:”0500” , http−
      identd:”−” , http−userid:”−” , http−method:”HEAD” ,
      http−path:”/” , http−protocol:”HTTP/1.1” , http−code
      :”200” , object−size:”0” , http−referer: , user−agent
      :] 1 , 0 , 1 . 0 ,”05/Mar/2018:19:22:52”

observation access_o_2 = ([src−ip:”127.0.0.1” , access−
      date:”05/Mar/2018:19:22:57” , timezone:”0500” , http−
      identd:”−” , http−userid:”−” , http−method:”HEAD” ,
      http−path:”/htdocs/” , http−protocol:”HTTP/1.1” ,
      http−code:”404” , object−size:”0” , http−referer: ,
      user−agent:] 1 , 0 , 1 . 0 ,”05/Mar/2018:19:22:57”
```

Listing 3. Forensic Lucid Format Example Output 2

A collection of such observations from each server will constitute an observation sequence – a formal story witnessed by the web server, that, possibly combined with any other evidence, may be used as a part of the larger case with other evidence and reasoning functions.

IV. CONCLUSION

We have implemented the initial FORENSIC LUCID facility for the Apache web server, but have a number of limitations to address:

- Recognize the types and the orders of the format string (combined) passed within the Apache configuration (`mod_log_config`), to have an accurate parsing–it could change from an Apache version/configuration to another:
- Finalize the piped logging ability, to have the FORENSIC LUCID observations generated on the fly (same type as the `rotatelogs` program) and choose the destination path, along with the logfile generation frequency (how many request should be done to rotate the observations logfile).

A. Future work

Majority of the future work will focus on addressing the mentioned limitations as well as improving flexibility and testing scalability of the approach. More concrete items are outlined below:

- The process of conversion between JSON to FORENSIC LUCID should be verified through Isabelle/HOL [5], [17]. The research team invested time and resources to learn this tool and implement a verification process and this process is under investigation. For future work, the conversion will be validated and verified by Isabelle/HOL.
- Build a library of formalized components and cases as a dataset for research and investigative community and release it as open-source.
- Dynamically (to be implemented): To log into directly into FORENSIC LUCID using piped log feature by passing `-D` option, as below:
  This implementation, is based on the same program but will assume that the passed log file is going through the standard input `STDIN`, so anything that gets logged with Apache, will be written to /var/log/httpd/access_Flucid_log.

V. ADDITIONAL REFERENCES

(2) https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

(3) https://blog.restcase.com/rest-api-error-codes-101/

(5) https://whatis.techtarget.com/definition/log-log-file

(6) http://docs.roxen.com/pike/7.0/tutorial/strings/sscanf.xml

(8) https://users.encs.concordia.ca/~g66104/- lecture notes

(10) https://www.liquidweb.com/kb/how-and-why-enabling-apaches-pip

(11) http://httpd.apache.org/docs/2.2/logs.html#piped

(12) https://www.utica.edu/academic/institutes/ecii/publications/articles/A0B70121-FD6C-3DBA-0EA5C3E93CC575FA.pdf

(13) Sumalatha, M.R. Title-Data collection and audit logs of digital forensics in cloud, URL:https://ieeexplore.ieee.org/document/7569587/

(14) Jorge Herrerias. Title-A Log Correlation Model to Support the Evidence Search Process in a Forensic Investigation, URL:https://ieeexplore.ieee.org/document/4155348/

(15) Sean Thorpe. Title-Towards a Forensic-Based Service Oriented Architecture Framework for Auditing of Cloud Logs, URL:https://ieeexplore.ieee.org/document/6655678/

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\"" combined
```

Fig. 3. Format string for change and accurace parsing

```
CustomLog "|/usr/sbin/converter\_apache2\_FLucid -D" combined
```

Fig. 4. Dynamic custom logging

(19) Hyungseok Kim. Title-Network Forensic Evidence Generation and Verification Scheme (NFEGVS), URL: https://dl.acm.org/citation.cfm?id=2819443

(20) Brian D. Carrier. Title-An Event-Based Digital Forensic Investigation Framework, URL: http://www.digital-evidence.org/papers/dfrws_event.pdf

(21) Hyunji Chung. Title-Digital forensic investigation of cloud storage services, URL: https://www.sciencedirect.com/science/article/pii/S1742287612000400

REFERENCES

[1] S. A. Mokhov, M. J. Assels, J. Paquet, and M. Debbabi, "Automating MAC spoofer evidence gathering and encoding for investigations," in *Proceedings of The 7th International Symposium on Foundations & Practice of Security (FPS'14)*, ser. LNCS 8930, F. Cuppens *et al.*, Eds. Springer, Nov. 2014, pp. 168–183, full paper.

[2] P. Albitz and C. Liu, *DNS and BIND*, 3rd ed. O'Reilly, 1998, ISBN: 1-56592-512-2.

[3] G. N. Purdy, *Linux iptables: Pocket Reference*. O'Reilly, 2004, ISBN: 978-0-596-00569-6.

[4] M. Rash, *Linux Firwalls: Attack Detection and Response with iptables, psad, and fwsnort*, 3rd ed. San Francisco: No Starch Press, Inc., 2007, ISBN: 978-1-59327-141-1.

[5] L. C. Paulson, T. Nipkow, and M. Wenzel, "Isabelle: A generic proof assistant," [online], University of Cambridge and Technical University of Munich, 2007–2015, http://isabelle.in.tum.de/, last viewed October 2015.

[6] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer-Verlag, Nov. 2007, vol. 2283, http://www.in.tum.de/∼nipkow/LNCS2283/, last viewed: December 2007.

[7] S. A. Mokhov, "Intensional cyberforensics," Ph.D. dissertation, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, Sep. 2013, online at http://arxiv.org/abs/1312.0466.

[8] S. A. Mokhov and J. Paquet, "Formally specifying and proving operational aspects of Forensic Lucid in Isabelle," Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada, Tech. Rep. 2008-1-Ait Mohamed, Aug. 2008, in Theorem Proving in Higher Order Logics (TPHOLs2008): Emerging Trends Proceedings. Online at: http://users.encs.concordia.ca/∼tphols08/TPHOLs2008/ET/76-98.pdf and http://arxiv.org/abs/0904.3789.

[9] S. A. Mokhov, "Encoding forensic multimedia evidence from MARF applications as Forensic Lucid expressions," in *Novel Algorithms and Techniques in Telecommunications and Networking, proceedings of CISSE'08*, T. Sobh, K. Elleithy, and A. Mahmood, Eds. University of Bridgeport, CT, USA: Springer, Dec. 2008, pp. 413–416, printed in January 2010.

[10] S. A. Mokhov, J. Paquet, and M. Debbabi, "Towards automatic deduction and event reconstruction using Forensic Lucid and probabilities to encode the IDS evidence," in *Proceedings of Recent Advances in Intrusion Detection RAID'10*, ser. Lecture Notes in Computer Science (LNCS), S. Jha, R. Sommer, and C. Kreibich, Eds., vol. 6307. Springer Berlin Heidelberg, Sep. 2010, Poster, pp. 508–509.

[11] Apache Foundation, "Apache HTTP Server Project," [online], 1997–2018, https://httpd.apache.org/.

[12] Netcraft, "January 2017 web server survey," [online], Jan. 2017, https://news.netcraft.com/archives/2017/01/12/january-2017-web-server-survey.html.

[13] A. R. Arasteh, M. Debbabi, A. Sakha, and M. Saleh, "Analyzing multiple logs for forensic evidence," *Digital Investigation Journal*, vol. 4, no. 1, pp. 82–91, Sep. 2007.

[14] P. Gladyshev, "Formalising event reconstruction in digital investigations," Ph.D. dissertation, Department of Computer Science, University College Dublin, Aug. 2004, online at http://www.formalforensics.org/publications/thesis/index.html.

[15] RJK, "Regexp syntax summary," http://www.greenend.org.uk/rjk/2002/06/regexp.html, last viewed May 2008, Jun. 2002.

[16] O. A. Mohamed, C. A. M. noz, and S. Tahar, Eds., *Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008*, ser. LNCS, vol. 5170. Springer, 2008.

[17] M. Wenzel, L. C. Paulson, and T. Nipkow, "The Isabelle framework," in *TPHOLs*, ser. LNCS, O. A. Mohamed, C. A. M. noz, and S. Tahar, Eds., vol. 5170. Springer, 2008, pp. 33–38.

[18] C. Charland, M. Dörfelt, J. Echelman, A. Koblin, M. Song, S. A. Mokhov, and P. Grogono, "Demo hour," *Interactions*, vol. 21, no. 4, pp. 8–11, Jul. 2014.