Paper

Graph-based Forensic Analysis of Web Honeypot

Hudan Studiawan, Supeno Djanali, and Baskoro Adi Pratomo

Department of Informatics, Institut Teknologi Sepuluh Nopember (ITS), Surabaya, Indonesia

Abstract—Honeypot still plays an important role in network security, especially in analyzing attack type and defining attacker patterns. Previous research has mainly focused on detecting attack pattern while categorization of type has not yet been-comprehensively discussed. Nowadays, the web application is the most common and popular way for users to gather information, but it also invites attackers to assault the system. Therefore, deployment of a web honeypot is important and its forensic analysis is urgently required. In this paper, authors propose attack type analysis from web honeypot log for forensic purposes. Every log is represented as a vertex in a graph. Then a custom agglomerative clustering to categorize attack type based on PHP-IDS rules is deployed. A visualization of large graphs is also provided since the actual logs contain tens of thousands of rows of records. The experimental results show that the proposed model can help forensic investigators examine a web honeypot log more precisely.

Keywords—access log, attack type, graph agglomerative clustering, visualization of large graphs.

1. Introduction

A honeypot is a system that lets attackers hack into it and records all activities so that their behavior can be observed [1]. There are several types of honeypots based on various services for example web, SSH, database, and network traffic or using the level of interaction or the responses given to the attacker, i.e. low, medium, and high [1]. The use of honeypots for forensic analysis was first developed in 2002 [2]. The researchers proposed a new forensic model and two architectures for honeypots, serial and parallel. In [3] the researcher tried to find the root cause of attack in a honeypot and the association rules were deployed to detect suspicious activity.

Forensic analysis can be conducted of both network traffic [4] and the compromised host [5] to get more detailed information from honeypot logs. More advanced techniques to review honeypot capabilities were introduced by using file system journaling [6]. It provided deeper analysis through file system abstraction to achieve meta data archiving. Forensic investigators are also able to generate attack statistics reports from honeypot deployment as observed in [7], [8]. Riebach *et al.* provided a case study of honeypot deployment to support forensic analysis and gave more attention to worm activities and classical multi-phase attack [9].

The investigation can be conducted in real time, usually called live forensic, as designed in [10] and the author focused on HTTP, FTP, POP3, and telnet protocols. Virtualization technology, cheaper than physical infrastructure could be used to deploy honeypots and supply real time forensics [11]. Attacker pattern was deeply investigated in [12] by clustering technique and time series analysis. The honeypot could be used for army of zombies detection [13]. They supplied an accurate observation of a botnet attack from honeypot trace. The other study of network forensics based on honeypot has been comprehensively described in [14]. They provided a survey, comparison, and future directions for this research area. Another work to provide honeypot for production mode of web application has been proposed by Pohl *et al.* since most honeypots only provide service in a non-productive environment [15]. The implementation of many honeypot platforms is also presented in [16]. However, these previous works still have not provided any interactive, collaborative, or real tools to model and visualize a web honeypot log.

Valli presented a visualization of honeypot data based on graph theory using Graphviz library and exploited AfterGlow for generating link graphs [17]. However, it could not provide interactive display for forensic investigators although could give large graph visualization. A recent study by Cabaj gives a graph visualization to assist in data analysis generated by honeypots [18]. The paper described visualization of the Honeypot Management System (HPMS) deployed at Institute of Computer Science, Warsaw University of Technology, but unfortunately this work only handled an attack to phpMyAdmin.

In this paper, a model for web honeypot logs based on graph theory is proposed. The authors will adopt the graph model to represent a log, and clustering technique will be deployed to analyze attacks for forensic purposes. The forensic investigation will be covered, which needs to know the origin and the type of penetration, e.g., cross site scripting or remote file inclusion. The visualization of proposed method will relax the one proposed by Cabaj [18].

The rest of this paper is organized as follows. Section 2 briefly describes the proposed log model based on graph theory, rule-based attack detection technique, and visualization of generated large graph. Section 3 explains experimental results and its analysis. Finally, conclusion and future works are discussed in Section 4.

2. Proposed Method

2.1. Graph-based Agglomerative Cluster

The main evidence for forensic analysis is the raw access log containing every request to the web honeypot. This

artifact was provided by Honeynet Project [19]. Authors then define graph $G = (V, E)$ to represent all access logs, propose custom agglomerative clustering on the set of vertices V, and generate edges E during the clustering process. The similarity measurement for clustering is used based on the node's attributes value, i.e., attack type, attacker's IP address, origin city and country. This gives an exact similarity since these properties are standardized for all vertices. Therefore, there are six levels of hierarchy in the resulting cluster, i.e., one level for leaf vertex representing the raw request and five upper-levels of intermediate nodes, which represent cluster root of attack type, IP address, city, country, and top-level cluster of the main graph used for overall forensic analysis. Cluster C as a subgraph of G is defined as:

$$C = \{C_{l_i} \mid C \subset G, l = 0, 1, \ldots 5 \text{ and } i = 0, 1, \ldots, p\}, \quad (1)$$

where l is level of hierarchy and p is a set of the total number of clusters in specific level l as shown in Fig. 1 and every level is represented by a different color. Meanwhile, the node size also depicts level in graph – the deeper the level, the smaller its size.

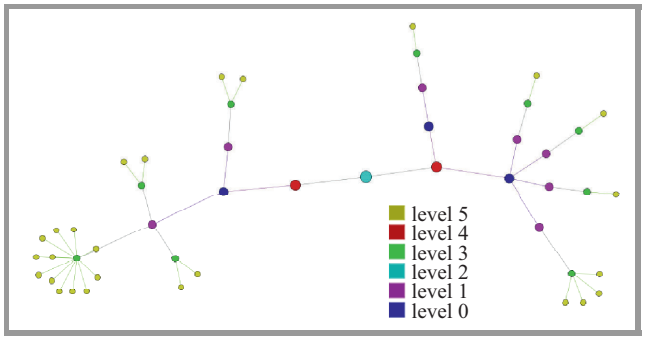

Fig. 1. Illustration of hierarchy level in proposed method. (See color pictures online at www.nit.eu/publications/journal-jtit)

In addition, by V_{l_j} the set of vertices of G in cluster level l is denoted:

$$V = \{v_{0_j}, v_{1_j}, \ldots, v_{l_m}\}, \quad (2)$$

where $j = 0, 1, \ldots, m$ and m is the total number of logs plus intermediate nodes created during clustering in level l. These vertices store every record and its attributes including id, attacker's remote address, time stamp, raw request, referrer, user agent, origin city and country, attack type and its description. Thus, some flags are also maintained as node's attributes in order to handle the clustering process, i.e., is_attackroot, is_iproot, is_cityroot, is_countryroot, and is_mainroot. These flags also distinguish whether or not a vertex is a root of a particular cluster.

Furthermore, there are six types of edges based on cluster level, E_l. These edges connect vertices to the intermediate node in each cluster and are defined as follows:

$$E = \{E_{0_k}, E_{1_k}, \ldots, E_{l_n}\} \quad (3)$$

where $k = 0, 1, \ldots, n$ and n is total number of cluster C_{l_i}. For each E_{l_k}, one can see an edge as a tuple consisting of a vertex and its connected cluster root:

$$E_{l_k} = \{(v_{0_j}, R_{0_i}), (v_{1_j}, R_{1_i}), \ldots, (v_{l_j}, R_{l_i})\}, \quad (4)$$

where variable i, j, and l have been described in previous equations.

Before creating edges E, all of the vertices based on IP address and attack type attributes are clustered. For each cluster, the procedure creates one vertex as an intermediate node acting as cluster root R. In other words, all vertices except v_m are an intermediate node. Formally,

$$R = V \setminus \{v_{x_j}\}, \; x = 0, 1, \ldots, m-1. \quad (5)$$

After that, every vertex in cluster C_{l_i} will have an edge to the root $R_{(l-1)_i}$ so that E_{l_k} is a set of edges in C_{l_i}. The first created R will act as R_{4_i}, root of cluster by attack type, and every R_4 has an edge to R_3, root of cluster by attacker's IP address. This step will provide E_4 and generates all C_{4_i} and its R_{4_i}. In this way, investigators can easily examine how many and what type of attacks were attempted from one IP address. The illustration of how to cluster vertices based on attack type and IP address are given in Figs. 2 and 3, respectively, where the vertices with the same attribute have the same color. In these figures, only 26 records consisting of two detected attacks (cross-site scripting and local file inclusion), seven IP addresses, three cities (Budapest, Seoul, and Osan), and two countries (Hungary and Korea) from the first file of the Honeynet Project

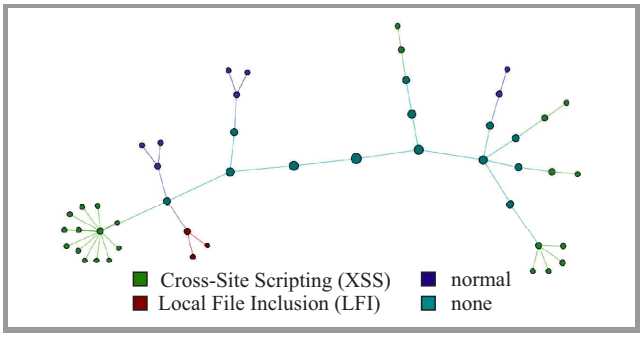

Fig. 2. Cluster by attack type.

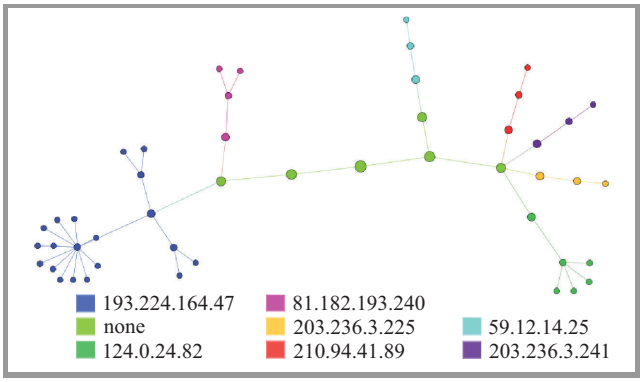

Fig. 3. Cluster by IP address.

dataset [19] in order to reduce the complexity of visualization are used.

The clustering continues to agglomerate existing group C_3 based on the origin city and then country of attacker. This geographic information is based on IP address and can be retrieved using the open GeoIP API module provided by MaxMind [20] and its Python bindings [21] for easy implementation. This procedure will produce cluster C_2, cluster by city, and C_1, cluster by country. The edges are drawn from every vertex in C_2 and C_1 to their respective intermediate root R so that we now have E_1 and E_2. Figure 4 provides clustering based on city data and the illustration of cluster by country can be easily inferred from Fig. 4. All illustrations in this section use the Yifan Hu graph layout [22] implemented in Gephi graph editor [23]. The more complex graph drawing will be explained in the next subsection.

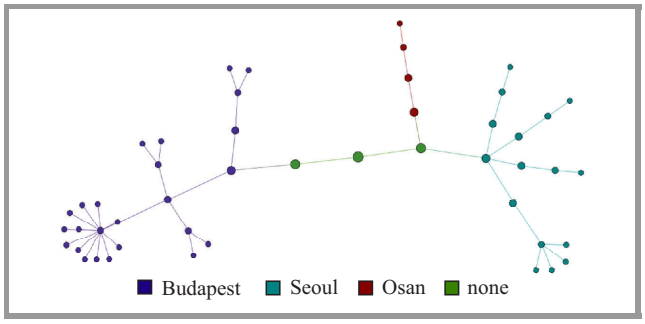

Fig. 4. Cluster by city.

In the last iteration, all separate clusters C_1 are connected to the main root R_0, resulting in both sets of edges E_0 and the cluster C_0. Through the clustering, each root flag is set to True according to processed level. This bottom-up approach provides a complete graph for forensic analysis and understanding the origin of attacks. The proposed method also provides a natural hierarchical structure to assist forensic investigators to understand attacker's behavior.

2.2. Attack Type Detection Using PHP-IDS Rules

The authors match every request log with the PHP-Intrusion Detection System (PHP-IDS) [24]. PHP-IDS rules currently contain 78 filters categorized to nine attack types: cross-site scripting (XSS), SQL injection (SQLi), cross-site request forgery (CSRF), denial of service (DoS), directory traversal (DT), spam, information disclosure (ID), remote file execution (RFE), and local file inclusion (LFI). Originally, these filters are utilized to check whether or not a request is suspicious in a PHP-based web application and it is installed and preconfigured inside the application.

However, instead of using PHP-IDS to detect attack in *ante mortem* conditions, the rules to assist forensic investigators are adopted, when examining the type of attack in *post mortem* fashion. PHP-IDS filters contain a sequence of regular expression (regex), which is developed and maintained periodically by the community. The examples of PHP-IDS filters are given in Table 1 [24].

Table 1
Example of PHP-IDS filters

No.	Filter and description	Tag
1	`(?:%u(?:ff|00|e\d)\w\w)|(?:(?:%(?:e\w|c[^3\W]|))(?:%\w\w)(?:%\w\w)?)` Detects halfwidth/fullwidth encoded unicode HTML breaking attempts	XSS
2	`(?:(?:[;]+|(<[?%](?:php)?)).*[^\w](?:echo|print|print_r|var_dump|[fp]open))|(?:;\s*rm\s+-\w+\s+)|(?:;.*{.*\$\w+\s*=)|(?:\$\w+\s*\[\]\s*=\s*)` Detects code injection attempts	RFE
3	`(?:%c0%ae\/)|(?:(?:\/|\\)(home|conf|usr|etc|proc|opt|s?bin|local|dev|tmp|kern|[br]oot|sys|system|windows|winnt|program|%[a-z_-]{3,}%)(?:\/|\\))|(?:(?:\/|\\)inetpub|localstart\.asp|boot\.ini)` Detects specific directory and path traversal	LFI

To implement the filters in test environment, an Apache Scalp, a Python implementation of PHP-IDS [25] updated using the newest PHP-IDS rules is deployed. Every filter can be attached to one or more attack types (tag) but only the first-found one using non-exhaustive mode in Apache Scalp is included. The procedure first parses the raw request, detects the HTTP method used, and then compares the request line to every regular expression in predefined filters. It returns the attack type for each attacker request to the web honeypot server.

2.3. Large Graph Visualization for Attack Type Analysis

The authors use Gephi [23] as graph editor and OpenOrd [26] as large graph layout since there are tens of thousands of logs to process and the existing typical layout can not well visualize large graph generated. Previous visualization by Cabaj [18] only displays a small portion of the attack where the vertices represent attacker, malware filename, and malware server. The display only showed a small part of the graph so the investigator may not see the entire attack attempts.

Presented work improves upon the one from [18] where the visualization is created as a tool to help forensic investigators inspect the attack type and the attacker location comprehensively using a large graph layout. An example of overall graph is shown in Fig. 5 where each color represents an attacks type: cross-site scripting (light green), remote file execution (purple), and local file inclusion (dark green) [19]. Normal means the request is not malicious and None is for an intermediate node for clustering. One can see in Fig. 5 that this visualization model will easily help the investigator to view, check, and analyze both normal requests and attempted attacks to the web honeypot as a whole. One can not clearly see the local file inclusion attack since it has only a very small number compared to overall records.

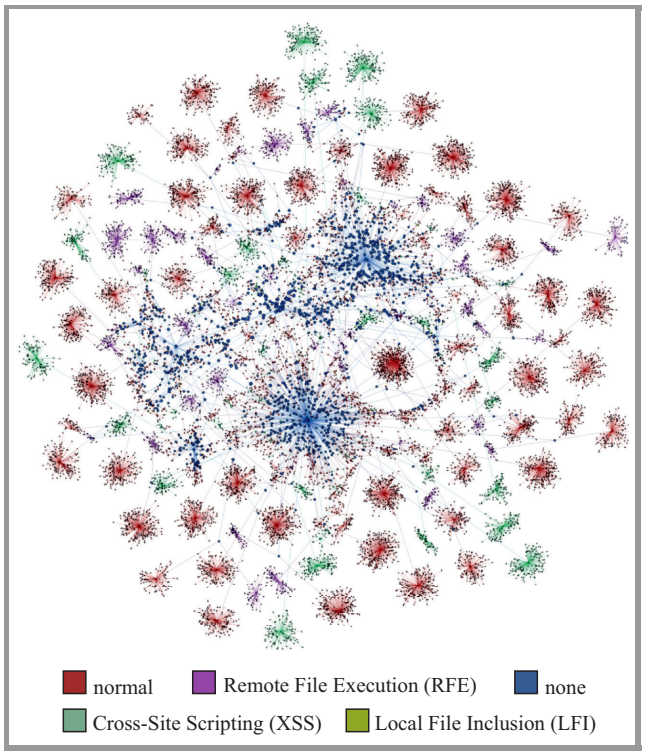

Fig. 5. Generated graph *G* from Honeynet Project dataset.

To enable the collaboration and interactive graph, the network graph from Gephi to Sigma.js was exported and hosted it a in Sigma.js server so that the investigator can access this visualization using a web interface. Sigma.js is a JavaScript library based on Node.js that is specially designed for interactive graph drawing [27]. The interactive mode means that the user is provided with a clickable vertex and cluster of vertices. This technique completes the analysis provided by large graph visualization with detailed examination of each request by displaying more detailed information when a node is clicked.

3. Experimental Results

The dataset used in this experiment is taken from the Honeynet Project [19], which is a real-life log from honeypot deployed in 2006. The complete dataset contains all logs in the /var/log directory from Linux Fedora operating system. As the authors focused on web honeypot, only the access_log file from /var/log/httpd has been taken into account. There are 32 access_log files consisting of 31 archives and one recent log containing 14,398 lines of raw requests, and we examined only the last one. Every line in the log file will be parsed and each entity becomes vertex's attributes as described in Subsection 2.1.

To manage and implement the graph, the Python-igraph [28] was used since it is fast, community-supported, and open source. It is also designed for efficiency, portability, and deployment-friendly. Apache Scalp [25] is employed to detect attack type in every request log.

The output from graph implementation using Python-igraph is a GraphML file, a XML-based format for graphs. This file is processed using Gephi to add color and provide clear and precise layout. The processed graph is then exported to Sigma.js code using Sigma.js exporter plugin in Gephi [29] and the resulting network is configured in Sigma.js server. This action will enable collaborative and interactive visualization between forensic investigators using web-based interface.

The graph *G* produced from Subsection 2.1 (Fig. 5), which is exported to Sigma.js server, is depicted in Fig. 6. When an investigator clicks a vertex, this tool will show detailed information about all attributes. It can be zoomed and slid smoothly to view the node accurately as shown in Fig. 7.

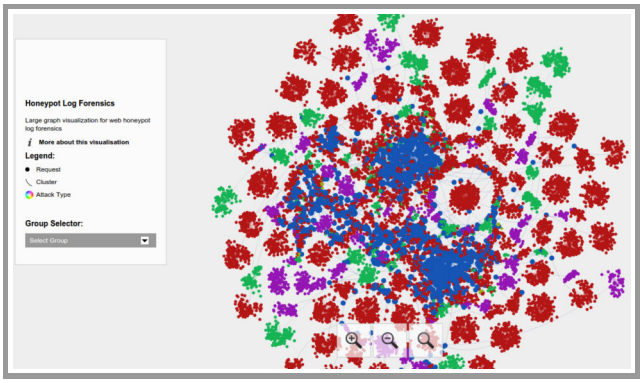

Fig. 6. Large graph visualization in Sigma.js web interface.

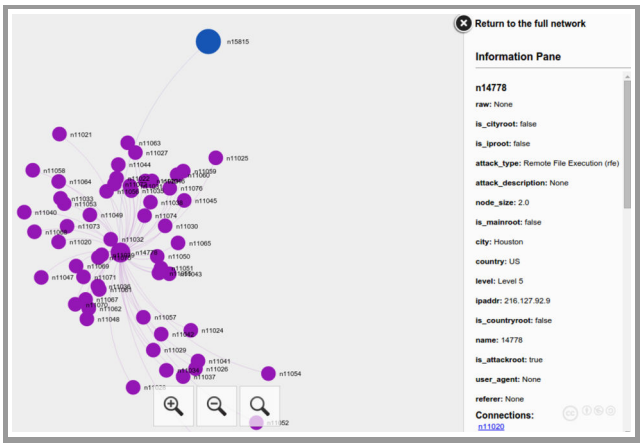

Fig. 7. The display when investigator clicks the root of attack type cluster.

There are three attack types detected in this dataset: 14.66% cross-site scripting (XSS), remote file execution (RFE) with 11.64%, local file inclusion (LFI) with 0.13%, and the normal request account of 65.83%. Table 2 depicts the examples of detected requests based on every PHP-IDS filter shown in Table 1. In the first example, the attacker tried to run a script while in the second one he executed a remote script that was previously downloaded using wget command. The last example shows that the attacker attempts to run an unauthorized file in local directory.

Table 2
Example of detected malicious requests

No.	Raw request	Attack
1	193.224.164.47 - - [27/Feb/2006:03:01:23 -0500] "GET /scripts/..%%35c../winnt/ system32/cmd.exe?/c+dir HTTP/1.0" 400 297 "-" "-"	XSS
2	62.175.253.180 - - [22/Jan/2006:08:28:17 -0500] "GET /awstats/awstats.pl?configdir=| echo;echo%20YYY;cd%20%2ftmp%3bwget%20209 %2e136%2e48%2e69%2fmirela%3bchmod%20%2bx %20mirela%3b%2e%2fmirela;echo%20YYY;echo| HTTP/1.1" 404 296 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;)"	RFE
3	218.26.222.13 - - [07/Feb/2006:01:36:28 -0500] "GET /c/winnt/system32/cmd.exe?/ c+dir HTTP/1.0" 404 293 "-" "-"	LFI

Table 3
Top five countries and their cities for each attack type

No.	Attack type	Country	City	Count
1	Cross-Site Scripting (XSS)	n/a	n/a	1497
		US	Amana	138
		US	Livingston	120
		DE	Berlin	108
		US	Pleasanton	84
2	Remote File Execution (RFE)	n/a	n/a	753
		CN	Beijing	245
		US	Dallas	99
		FR	Bischheim	74
		US	Saint Louis	60
3	Local File Inclusion (LFI)	n/a	n/a	4
		CN	Harbin	2
		CN	Jinan	2
		HU	Budapest	2
		US	Schaumburg	2
List of countries: US – United States, DE – Germany, CN – China, FR – France, HU – Hungary.				

Table 4
Top ten attacker's IP addresses

No.	IP address	Attack type	Count
1	64.6.73.199	Cross-Site Scripting	138
2	81.114.87.11	Cross-Site Scripting	138
3	80.55.248.206	Cross-Site Scripting	132
4	200.99.135.130	Cross-Site Scripting	120
5	209.137.246.36	Cross-Site Scripting	120
6	211.99.203.228	Remote File Execution	120
7	64.214.80.6	Cross-Site Scripting	108
8	82.127.23.55	Cross-Site Scripting	108
9	85.214.20.161	Cross-Site Scripting	108
10	82.177.96.6	Remote File Execution	107

Table 3 shows the top five countries and their respective cities for each attack type while Table 4 lists the top 10 attacker's IP addresses. One can see that there are very high "n/a" values in Table 3 since the free and open source version of GeoIP API is used [20] and the provided data are not as complete as in the paid one. As stated in Table 3, the most frequently detected IP address (64.6.73.199) attempted an XSS attack as shown in Table 4. The remote file inclusion attacks are also included in the top ten IP addresses while there is no local file inclusion since it only has a very small number of attempts from the whole dataset.

4. Conclusions and Future Works

In this paper, a graph-based forensic analysis have been implemented to examine an access log from a web honeypot. The proposed method employs an agglomerative clustering to group every record and to model them as an undirected graph. The clustering has some levels based on node's attributes, i.e., attack type, attacker IP address, and attacker's origin city and country. Every log is checked for its maliciousness, then visualized using a large graph layout, and then accessed using a web browser. These procedures will help forensic investigators to examine logs to work interactively and collaboratively with others.

In the future, authors plan to include more access logs archived in /var/log/httpd. This strategy will enable the investigator to comprehensively analyze through the last period of log rotation (the most common time frame is one month). The Sigma.js web interface will be improved to view a specific range of time, although it still displays a large graph generated from thousands of logs. In addition, the proposed method can be extended to become live forensic analysis of a web honeypot or typical web application by reading and parsing the access log periodically. This approach provides administrators real-time monitoring and reports if there are any attack attempts to their system. In relation to the types of offensive activity, the system will be enhanced with OWASP ModSecurity Rules [30] which contains more complete rules since it is actually a web application firewall but that can be utilized in a forensic manner. To increase the reliability of the proposed technique, authors also plan to implement a graph database such as Neo4j or Titan (natively distributed one) to make analyzed logs become persistent and able to be queried any time.

References

[1] L. Spitzner, Honeypots: *Tracking Hackers*. Boston: Addison-Wesley Longman Publish., 2002.

[2] A. Yasinsac and Y. Manzano, "Honeytraps: A network forensic tool", in *Proc. World Multiconf. System., Cybernet. & Informat. SCI 2002*, Orlando, FL, USA, 2002.

[3] F. Pouget and M. Dacier, "Honeypot-based forensics", in *Proc. AusCERT Asia Pacific Inform. Technol. Secur. Conf. AusCERT2004*, Gold Coast, Australia, 2004.

[4] F. Raynal, Y. Berthier, P. Biondi, and D. Kaminsky, "Honeypot forensics, part I: Analyzing the network", *IEEE Secur. Privacy*, vol. 2, no. 4, pp. 72–78, 2004.

[5] F. Raynal, Y. Berthier, P. Biondi, and D. Kaminsky, "Honeypot forensics, part II: Analyzing the compromised host", *IEEE Secur. Privacy*, vol. 2, no. 5, pp. 77–80, 2004.

[6] K. D. Fairbanks, C. P. Lee, Y. H. Xia, and H. L. Owen, "Timekeeper: a metadata archiving method for honeypot forensics", in *Proc. Inform. Assurance & Secur. Worksh. IAW'07*, West Point, NY, USA pp. 114–118, 2007.

[7] P. T. Chen, C. S. Laih, F. Pouget, and M. Dacier, "Comparative survey of local honeypot sensors to assist network forensics", in *Proc. 1st Int. Worksh. System. Approach. to Digit. Forensic Engin. SADFE'05*, Taipei, Taiwan, 2005, pp. 120–132.

[8] V. Maheswari and P. E. Sankaranarayanan, "Honeypots: Deployment and data forensic analysis", in *Proc. Int. Conf. Computational Intellig. & Multimed. Appl.*, Sivakasi, Tamil Nadu, 2007, vol. 4, pp. 129–131.

[9] S. Riebach, E. P. Rathgeb, and B. Toedtmann, "Efficient deployment of honeynets for statistical and forensic analysis of attacks from the Internet", in *NETWORKING 2005. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems*, R. Boutaba, K. Almeroth, R. Puigjaner, S. Shen, and J. P. Black, Eds., *LNCS*, vol. 3462, pp. 756–767. Springer, 2005.

[10] W. Ren and H. Jin, "Honeynet based distributed adaptive network forensics and active real time investigation", in *Proc. 20th ACM Symp. Appl. Comput. SAC 2005*, Santa Fe, NM, USA, 2005, pp. 302–303.

[11] A. Capalik, "Next-generation honeynet technology with real-time forensics for U.S. defense", in *Proc. IEEE Milit. Commun. Conf. MILCOM 2007*, Orlando, FL, USA, 2007, pp. 1–7.

[12] O. Thonnard and M. Dacier, "A framework for attack patterns' discovery in honeynet data", *Digit. Investig.*, vol. 5 (Supplement), pp. S128–S139, 2008.

[13] V. H. Pham and M. Dacier, "Honeypot trace forensics: The observation viewpoint matters", *Future Gener. Comp. Syst.*, vol. 27, no. 5, pp. 539–546, 2011.

[14] Q. Nasir and Z. A. Al-Mousa, "Honeypots aiding network forensics: Challenges and notions", *J. Commun.*, vol. 8, no. 11, pp. 700–707, 2013.

[15] C. Pohl, A. Zugenmaier, M. Meier, and H. Hof, "B.Hive: A zero configuration forms honeypot for productive web applications", in *ICT Systems Security and Privacy Protection*, *IFIP Advances in Information and Communication Technology*, vol. 455, pp. 267–280. Springer, 2015.

[16] K. Cabaj, and P. Gawkowski, "Systemy HoneyPot w praktyce (HoneyPot systems in practice)", *Przegląd Elektrotechniczny*, vol. 91, no. 2, pp. 63–67, 2015 (in Polish) (doi: 10.15199/48.2015.OZ.16).

[17] C. Valli, "Visualisation of honeypot data using Graphviz and Afterglow", *J. Digit. Forensic, Secur. & Law*, vol. 4, no. 2, pp. 27–38, 2011.

[18] K. Cabaj, "Visualization as support for web honeypot data analysis", *Inform. Systems in Manag.*, vol. 4, no. 1, pp. 14–25, 2015.

[19] A. Chuvakin, "Free honeynet log data for research", The Honeynet Project [Online]. Available: https://www.honeynet.org/node/456

[20] MaxMind, "GeoIP Legacy C API" [Online]. Available: https://github.com/maxmind/geoip-api-c

[21] Applied Security, "Pure Python API for Maxmind's binary GeoIP databases" [Online]. Available: https://github.com/appliedsec/pygeoip

[22] Y. Hu, "Efficient, high-quality force-directed graph drawing", *Mathematica J.*, vol. 10, no. 1, pp. 37–71, 2005.

[23] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: An open source software for exploring and manipulating networks", in *Proc. 3rd Int. AAAI Conf. on Weblogs & Soc. Media*, San Jose, CA, USA, 2009, pp. 361–362.

[24] PHP-IDS, "PHP-Intrusion Detection System" [Online]. Available: https://github.com/PHPIDS/PHPIDS

[25] R. Gaucher, "Apache Scalp: Apache log analyzer for security" [Online]. Available: https://code.google.com/p/apache-scalp/

[26] S. Martin, W. M. Brown, R. Klavans, and K. W. Boyack, "OpenOrd: An open-source toolbox for large graph layout", *Proc. of SPIE, Visualization and Data Analysis*, San Francisco, CA, USA, 2011, vol. 7868, pp. 786806–786806, 2011.

[27] Sigma.js, "JavaScript library dedicated to graph drawing" [Online]. Available: http://sigmajs.org/

[28] G. Csardi and T. Nepusz, "The igraph software package for complex network research", *InterJournal Complex Systems*, vol. 1695(5), pp. 1–9, 2006.

[29] S. Hale, "Sigmajs exporter – Gephi marketplace" [Online]. Available: https://marketplace.gephi.org/plugin/sigmajs-exporter/

[30] SpiderLabs, OWASP ModSecurity Core Rule Set (CRS) [Online]. Available: https://github.com/SpiderLabs/owasp-modsecurity-crs

Hudan Studiawan earned both a B.Sc. and M.Sc. degree from Department of Informatics, Institut Teknologi Sepuluh Nopember, Indonesia. He is now working as a lecturer at his alma mater university. His research areas are digital forensics, network security, and cluster computing.

E-mail: hudan@if.its.ac.id
Department of Informatics
Institut Teknologi Sepuluh Nopember (ITS)
Surabaya, Indonesia

Supeno Djanali is a Professor of Network Architecture and Design in Department of Informatics, Institut Teknologi Sepuluh Nopember, Indonesia. He studied at University of Wisconsin-Madison, USA, for both his master's and Ph.D. degrees. His research areas are primarily network security and mobile computing.

E-mail: supeno@its.ac.id
Department of Informatics
Institut Teknologi Sepuluh Nopember (ITS)
Surabaya, Indonesia

Baskoro Adi Pratomo is a lecturer in Department of Informatics, Institut Teknologi Sepuluh Nopember, Indonesia. He finished his B.Sc. and M.Sc. degree from the same university as he works at the moment. His main research interest is network security.

E-mail: baskoro@if.its.ac.id
Department of Informatics
Institut Teknologi Sepuluh Nopember (ITS)
Surabaya, Indonesia