Near Real Time Machine Driven Signature Detection, Generation and Collection

Edwin Ouma Ngwawe
School of Computing and Information Technology
Technical University of Kenya
Nairobi, Kenya
Email: edwin.ngwawe [AT] tukenya.ac.ke

Elisha Odira Abade
School of Computing and Informatics
University of Nairobi
Nairobi, Kenya

Abstract— **One way to perpetrate a Denial of Service (DoS) attack is to flood the network infrastructure with too much unnecessary data such as Internet worms. Internet worms can spread very fast and cause losses both in terms of lost business opportunities as well as human resources required to alleviate the caused damages. Ways of protecting against the Internet worms include the anomaly based and signature based systems. Signature based systems uses security signatures (patterns) that match particular known attacks while anomaly based systems rely on detecting anomalies with the background idea that abnormal activity is malicious. With the increasing internet speeds and growing amount and complexity of data across it, it is necessary to have correspondingly fast ways of analyzing network traffic in order to evaluate security scenarios in time. Also the existence zero-day attacks (attacks whose characteristics are still unknown) make relying on preconfigured signatures unreliable. This study sought to find how to develop an accurate, robust near real time machine driven Internet worm signature detection, generation and collection system using big data technologies. We set up Hadoop Ecosystem and analyze network traffic content using Hadoop Map Reduce programming Model. We were able to generate documented worm signatures. We also realize that adding the number of nodes to the Hadoop cluster first reduces the processing speeds due to overheads of load distribution up to some optimal point beyond which adding more nodes actually increases the overall speed of processing. The robustness of the processing facility is also improved due to the fact that HDFS replicates the files to be processed thus improving availability.**

Keywords- Signature real-time big-data polymorphic-worms, Hadoop, MapReduce

I. INTRODUCTION

Malware such as Internet worms spreads very fast and causes losses such as business disruptions as well as the human efforts necessary to alleviate the damages. Systems like Honeycomb [15], Autograph [13], EarlyBird [33], Polygraph [12], Hamsa [16] and Lisabeth [6] can monitor network traffic to identify new Internet worms and produce corresponding worm signatures in a small scale networks. Such systems use network traffic content analysis to generate signatures. The increasing speeds of the internet as per IEEE report [10] and the increasing number of devices being connected to the internet (Internet of Things) leads to generation of large volume of very high velocity and variety data. This presents the four Vs: volume, velocity, variety, and veracity, which characterize the big data [37].

While an ordinary desktop computer can generate signatures for reasonably less amount of data with low velocity and variety, it will start introducing delays as the data grows in volume, velocity, variety and at the same time increased degree of veracity is required, due to the limited computing resources.

While the term real-time is contextual, it can be defined as the span of time within which an intervention to an activity will be effective. Delays that is likely to be caused by single desktop computers analyzing big data for security purposes will hinder any possible intervention from being effective.

We therefore seek to experiment a way of using the existing algorithms but with different approach in order to analyze the big data in real-time and provide security signatures so it can monitor a larger network or even the Internet as per the current connectivity trends [30].

The remainder of this paper is organized as follows. Section II describes existing work, in section III, we present our proposed solution and present the evaluation of our system in section IV. We then present our conclusion in section V and then recommendation for future work in section VI.

II. EXISTING WORK

Intrusion detection systems can either be deployed at host level or at network level [12]. Systems like ACARM-ng [5], Taint check analysis [24], OSSEC [8], NIDES [1], eXpert-BSM [18], Fail2Ban [11], SAMHAIN [38], Sagan [29], Haystack [34] use traffic behavior at host level to evaluate security scenario. Host level behavior cannot be used to take care of attacks such as large-scale coordinated attacks [31]. Also single host based systems are not as robust [37].

Network based intrusion detection systems (NIDS) which include systems like Snort [32], and Bro [26,27] are standalone systems and will not scale well enough in the face of big data.

One more way of classifying intrusion detection systems is by use of the working mechanism, which can either be signature recognition or anomaly detection [39]. Signature

recognition systems such as the EarlyBird [33], HoneyComb [15] and Autograph [13], Paragraph [23] use network traffic content analysis to generate security signatures; however these systems do not capture polymorphic worms, these are worms that mutate at every instance to hide from detection. This was shown by a research that led to construction of Polygraph [22], however Polygraph was found to be so slow and was enhanced into Hamsa [19]. Hamsa was again found to be less resilient to targeted noise [28] by [6] and they enhanced Hamsa to Lisabeth [6]. Lisabeth analyzes network traffic to identify new Internet worms and generate the corresponding worm signatures with resilience to poisoning attacks. Lisabeth is currently central based and designed to run on a single computer. This limits the processing power, meaning that for it to process very large amount of complex data, it will have to be deployed on a very high end computer. The current computing trend is such that clusters of commodity computers are set up to provide higher computing powers than can be realized with any single high end computer [37].

Work has also been done to experiment and adopt big data infrastructure to improve processing speeds in the face of big data. These include the use of infrastructures such as DBStream [41], Hadoop Map Reduce and Spark [42]. DBStraeam uses PostgreSQL engine to provide a platform for large scale data analysis facility which can be used in network monitoring as discussed in [41], however, the dependency on PostgreSQL database engine makes parallelization impossible on DBStream platform and acts as a bottleneck in processing hence leading to longer processing time even or parallelizable tasks. Still there is a need to further extend the work to more flexible systems that adapt well for parallelization such as the Spark or the Hadoop Map Reduce. Spark [42] is a large-scale data processing platform which operates over Hadoop and supports in-memory operations hence in some cases a single Spark node can outperform a cluster of 10 Hadoop map reduce nodes. [42]. However, the in-memory operation is risky in the sense that the system memory state may be lost in event of a system crash due to power failure of hardware failure thus leading to loss of data. Hadoop Map Reduce therefore becomes a better option since it persists its data to the disk.

We adopt Lisabeth System and experimentally deploy it in Hadoop Map Reduce cluster environment.

III. PROPOSED SOLUTION

(i) Background to the proposed solution

We depend on the fact that all worms, whether polymorphic or not must have some invariant bytes for them to successfully achieve their goal.

Polymorphic Worm Structure

In a sample of polymorphic worm we can identify the following components [12]:

(i) Protocol framework. To infect new hosts and continue their spread, worms have to exploit a given vulnerability. This vulnerability, in many cases, is associated with a particular application code and execution path in this code. This execution path can be activated by few, or more often one, types of particular protocol request.

(ii) Exploit bytes. These bytes are used by the worm to exploit the vulnerability. They are necessary for the correct execution of the attack.

(iii) Worm body. These bytes contain instructions executed by the worm instances on new infected victims. In polymorphic worms these bytes can assume different values in each instance.

(iv) Polymorphic decryptor. The polymorphic decryptor decodes the worm body and starts its execution.

(v) Others bytes. These bytes do not affect the successfully execution of both the worm body and exploit bytes.

Content-based signature generation approaches rely upon the presence of invariant bytes in some of the identified components. Some of these components, for their nature, offer high chance of finding these invariant sequences which are useful for the signature generation purpose.

(ii) The proposed approach

Our approach relies on invariant bytes as discussed in section II (i) above. To allow a rapid spread of the worm, there will be considerably many flows in which all the invariant bytes occur. However, some of the invariant bytes should also appear in innocuous flows in order to prevent poisoning attacks [6, 21].

The fundamental principle is that given any suspicious traffic pool M and innocuous traffic pool N, the goal is to find a set S of signatures Si each of which covers many flows in M but not as many flows N. So the false positive FPsi must be low while the coverage, COVsi must be high.

$$FPsi = \frac{|Nsi|}{|N|} \qquad (1)$$

$$COVsi = \frac{|Msi|}{|M|} \qquad (2)$$

We use Hadoop MapReduce programming model to achieve this.

A. High Level Architecture

High level architecture of our prototype is very similar to the Lisabeth architecture, from which it is derived, only that for our system we tap traffic from several vantage points of inspections across the Internet. This is shown in Figure 1.

B. The Signature Generation Process

We crawl the internet for innocuous data using web crawler written in PHP. We use wireshark software to capture the network packets. We then create Hadoop mapReduce

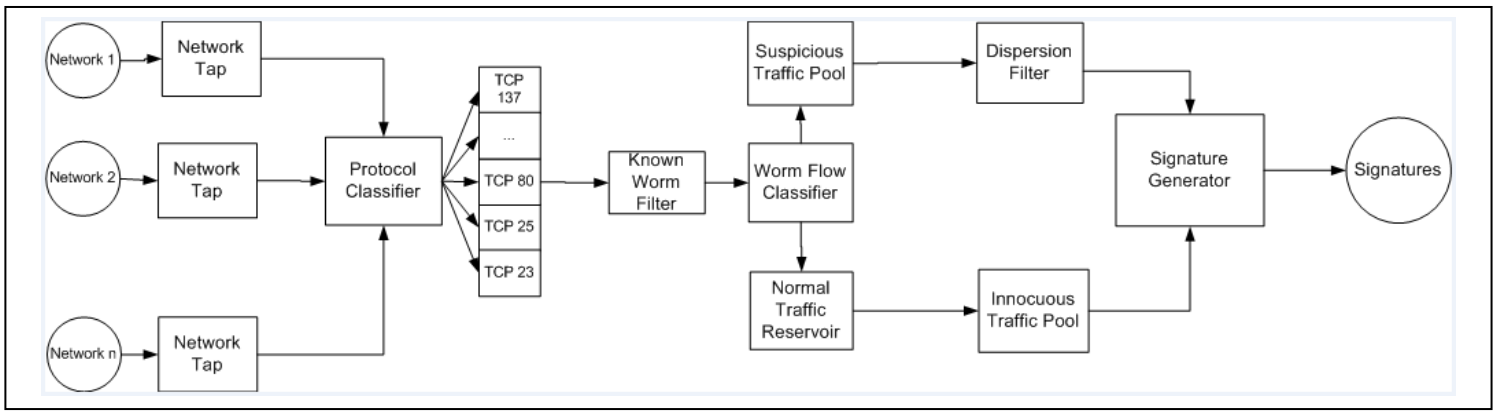

Figure 1. High Level Architecture

application [36,37] where the map section is written with the help of jnetpcap Java library [35] to preprocesses the data before feeding into the reduce section which now implements the Lisabeth algorithm (Figure 2). The output of the map phase involve the key being the source network and value being the payload (content) of the particular network. We then implement the reduce section to process this intermediary output based on the key value pairs to give the signatures specific to every network. We write the output in MySQL database and expose an API so security systems in corresponding networks can be programmed to automatically poll the signature database for new signatures for automatic self update.

The MapReduce

A MapReduce job is a unit of work that the client wants to be performed in a map reduce framework. It consists of the input data, the MapReduce program, and configuration information. Hadoop runs the job by dividing it into tasks, of which there are normally two types, map tasks and reduce tasks, sometimes involves the combiner stage but not mandatory and the combiner function needs not to be called for the mapReduce application to execute successfully. The map and reduce tasks are implemented as map and reduce methods in Hadoop MapReduce. It has the following general form:
Map task: (K1, V1) → list(K2, V2)

Reduce task: (K2, list(V2)) → list(K3, V3)

In general, the map input key and value types (K1 and V1) are different from the map output types (K2 and V2). However, the reduce input must have the same types as the map output, although the reduce output types may be different again (K3 and V3).

The HDFS System

HDFS is a file system designed for storing very large files with streaming data access, patterns, running on clusters of commodity hardware. HDFS has design features which make it optimal for processing of very large data sets [38]. The most important feature being the data file replication to ensure redundancy hence availability and also the fact that it provides a POSIX like interface hence which makes it easy to start working with.

Input: Malicious flow set M and innocuous flow set N, FPMAX and COVMIN
Output: Generated signatures set S for worms in M

```
TM = PS = ST = S = [ ]
tokenList = M.getTokenList()
tokenList.sort() ;                    /* Sorted by descending token length */
foreach t ∈ tokenList do
    for i ← 1 to tokenList.maxOcc(t) do
        id = genNewId()
        if PS.checkIfSubT(t, i) then
            ST.append(PS.findSuperT(t, i), id)
        end
        else
            PS.append(genNewId(), id, tokenList.getFlowList(t, i))
        end
        TM.append(id, t, i)
    end
end
foreach e ∈ PS do
    if PS.calcCov(e, M) < COVMIN then
        PS.delete(e)
    end
end
signLen = 1
while PS.isNotEmpty() do
    signLen += 1
    foreach e ∈ PS do
        if calcFP(e, N) < FPMAX then
            S.append(newSign(e, TM, ST))
            PS.delete(e)
        end
    end
    foreach e ∈ PS do
        foreach f ∈ PS ∧ f.id > e.id do
            tmp = merge(e, f)
            if tmp.tokenNum() == signLen then
                if calcCov(tmp,M) ≥ COVMIN then
                    PS.append(genNewId(), tmp.id, tmp.flow)
                end
            end
        end
        PS.delete(e)
    end
end
return S
```

Figure 2. Lisabeth Algorithm

IV. SYSTEM EVALUATION

The research required two sets of data, namely the innocuous network traffic, which is the network traffic as observed during normal operation of a given network system and the malicious network traffic, which is the network traffic that is being suspected to be originating form an attacker.

Innocuous traffic was collected from HTTP traces of controlled web crawlers. Malicious traffic was generated from known worm samples in the lab. Two models of machine driven signature detection, generation and collection systems were then developed, one for centralized deployment in a standalone machine (the regular Lisabeth) and one for distributed deployment in Hadoop cluster which uses Hadoop MapReduce programming model.

We set up a cluster with forty virtual machines of specifications 10GB RAM, 1 processing cores and 100 GB Hard Disk Drive running on 3 HP ProLiant DL380 Gen9 Servers with 2TB Hardisk Drives each, 128GB RAM and two Intel® Xeon® E5-2600 16-core processors, running on Ubuntu server operating system, and with each virtual machine running 64-bit Centos 6.5 on top of virtualBox hypervisor.

We use public IPs to achieve stable connectivity between the VMs.

We deploy the local version on a virtual machine with 10GB RAM, 4 processing cores and 100 GB.

The specifications of these virtual computers were held constant across the study while the number of nodes from which the signatures were generated was varied from one test to the other in the cluster setup.

TABLE 1 RESULT

Number of Nodes	System Type	Time to accomplish processing (seconds)	Average CPU Load (%)	Average Memory Usage (%)
1	Local	15	67	81
1	Cluster	25	100	89
5	Cluster	23	78	68
10	Cluster	20	65	61
15	Cluster	18	62	54
20	Cluster	15	54	49
25	Cluster	12	47	46
30	Cluster	10	39	44
35	Cluster	8	35	42
40	Cluster	7	32	41

Timing was done by flagging the start and stop finish time in a file for every process.

We feed in 50GB of data and observe processing time, false positives ratio, false negatives, and how computing recourses are utilized during the processing. Results were recorded as shown in Table 1.

Near real time speeds achievement

According to oxford English dictionary, real time is defined as the exact time at which a particular activity takes place. Moore [20] shows that detection and containment must be initiated within minutes or seconds to prevent wide-spread infection in a 24 hour period. According to Center for Applied Internet Data Analysis [7], the Sapphire worm was able to spread so quickly doubling the number of affected machines in every eight seconds. We therefore consider a speed of less than eight seconds to be within real-time frame, and anything around that to be near real-time. From our results as shown in Figure 5, we can only realize these speeds with thirty-five nodes and above. As can be seen from the graph, below ten nodes, the cluster seems to take longer than local processing. This may be attributed to the overheads of load balancing in HDFS before the number of nodes reaches some optimal level.

Resource Utilization

The graphs in Figures 6 and 7 shows that as number of nodes are added to the cluster, we generally realize calm down in the use of resources. At first the resource utilization seems to be higher than for a local setup. This can be attributed to the additional load due to the overheads as a result of load balancing in the HDFS system and implies that the cluster setup only becomes beneficial beyond some optimal number of nodes in the cluster, in this case about ten nodes, considering both CPU and memory usage hence as one sets up a cluster, he need to take care of this optimal point beyond which he will start realizing the benefits of a cluster.

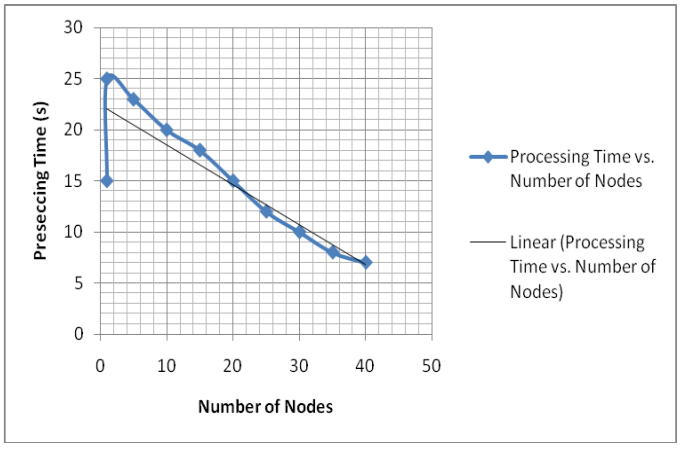

Figure 5. Processing Time vs. Number of nodes

As can be seen from the linear graph, load is inversely proportional to the number of processing nodes hence to ease memory and CPU capping of the processing, one needs to add processing nodes.

Accuracy
Accuracy is the degree with which a signature generation system generates correct signatures when there is an attack and generates no signatures when there is no attack. It is defined by false positives ratios and false negatives ratios as discussed below.

False Positives
False positive is the occurrence where by a signature generator generates a signature when there is no attack, i.e. it reports that there is signature (positive) when in actual sense there is no signature hence a false report. False positives are calculated by a ratio of the worm flows in the normal flows to the total normal flows. Therefore we can afford to increase the size of the normal flow in the cluster environments as long as we increase the number of processing nodes and still afford generation of signatures within a desired real time frame. This results in a lower false positive ratio in comparison to original Lisabeth [6], if all other factors are held constant.

False Negatives
False negative is the occurrence whereby a signature generator fails to generate signatures when actually it is supposed to generate some signature i.e. it reports that there is no (negative) signature when actually there is a signature to be generated hence a false report. Factoring in time, especially real-time scenario, ordinary system will generate delay and cause the polling mechanisms to miss the signature in time during signature collection which will be equivalent to a false negative hence our system will arguably have lower false negative in the face of big data, if all other factors are held constant.

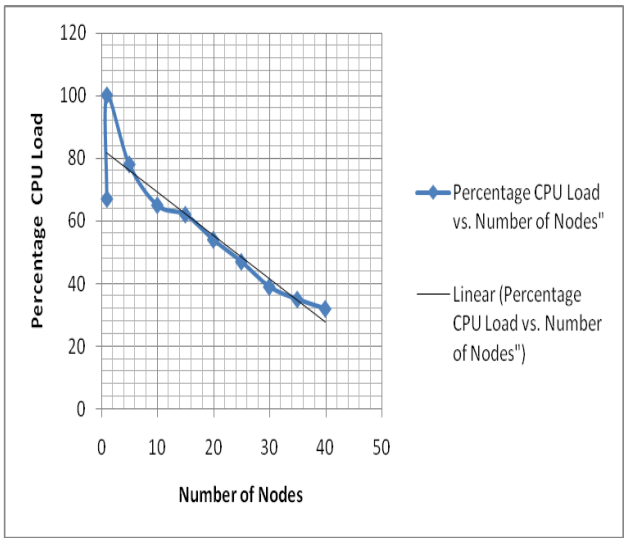

Figure 7. Percentage CPU Load vs. Number of nodes

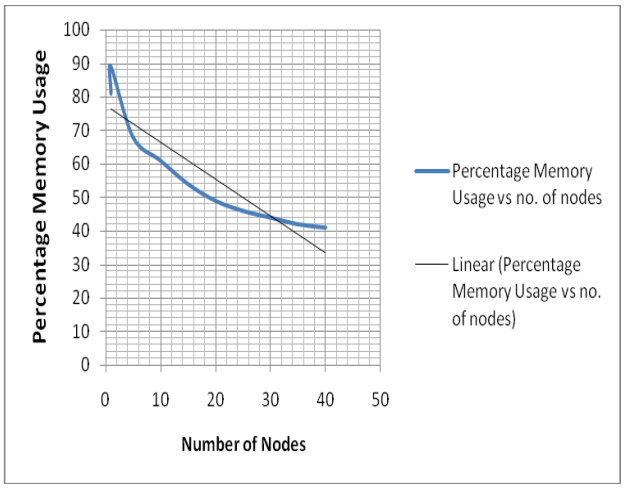

Figure 7. Percentage Memory Usage vs. Number of nodes

System Robustness
Robustness is the ability of a computer system to cope with errors during execution. In our case, we realize that our system is able to work even if one node fails even though at reduced overall performance. This means that signature generation process can continue to take place as opposed to a centralized system whereby when the only node running the entire system breaks down, the whole system stops the operation thus delicate. Also the data replication in HDFS system allows for improvement of data availability as opposed to the standalone system thus making it more robust.

V. CONCLUSSION

In this research, we have presented an reactive distributed accurate and robust near real time machine driven signature detection, generation and collection system for zero-day polymorphic worms which can be used to implement a central processing facility to inspect multiple points of a network or Internet over big data infrastructure to improve network security, especially to curb the denial of service attacks through network flooding. According to our experiments the prototype achieves significant improvements with respect to speed of processing, accuracy, and robustness as compared to Lisabeth [6], from which it is derived as the number of nodes in a cluster deployment increases after some optimal number.

VI. RECOMMENDATIONS FOR FUTURE WORKS

Yes we were able to achieve near real time speed in the context of worm spreading which requires around 8 seconds to contain, however we realize that it may require too many nodes to set up a facility that attains milliseconds time frame for distributed monitoring which is the real time frame for other applications such as missile control systems, we realize that since map reduce processes data from the disk, it might not be tenable through this direction. We are therefore intending to develop a system restore mechanism for Spark setup which can restore the memory state in event of system crash which is the spark weakness as single node of spark can

outperform 30 nodes of mapreduce. There is also a room for deploying DBStream in an infrastructure that can provided more parallelization than PostgreSQL database engine. This can also improve performance as a single node of DBstream can outperform 10 nodes of a Spark cluster or arguably 300 Hadoop MapReduce nodes cluster, therefore we are going on with the work to test the outcome when these two other setups are used. We are also considering using machine learning algorithms in the context of anomaly detection to improve the security on top of mere signature based methods since in some cases an artificial intelligence algorithm can take less data to evaluate a security scenario as opposed to structured programming and hence worth exploring. Also the use of machine learning algorithms and artificial intelligence for job scheduling in big data processing is within the plans for our future works.

VII. REFERENCES

[1] Anderson, D., Frivold, T. & Valdes, A., 1995. *Next-generation Intrusion Detection Expert System (NIDES), A summary*. [Online] Available at: http://www.csl.sri.com/papers/4sri/4sri.pdf [Accessed 10 July 2016].

[2] Anne-Fleur, K. & Pete, M., 2016. *Suricata*. [Online] Available at: https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Suricata [Accessed 10 July 2016].

[3] Anon., 2013. *The 2013 IBM Security Intelligence Index*. [Online] Available at: http://www.935.ibm.com/services/us/en/security/infographic/cybersecurityindex.html [Accessed 12 July 2016].

[4] Anon., 2015. *Prelude Specifications*. [Online] Available at: https://www.prelude-siem.org/projects/prelude/wiki/PreludeSpecifications [Accessed 10 July 2016].

[5] Bartłomiej, B., Bartosz, S., Mariusz, U. & Wojciech, W., 2012. ACARM-ng: Next Generation Correlation Framework. In Bartłomiej, B., Bartosz, S., Mariusz, U. & Wojciech, W. *Building a National Distributed e-Infrastructure–PL-Grid*. Berlin: Springer. pp.114-27.

[6] Cavallaro, L., Lanzi, A., Mayer, L. & Monga, M., 2008. *LISABETH: Automated Content-Based Signature Generator for Zero-day Polymorphic Worms*. Milan: University of Milan University of Milan.

[7] Center for Applied Internet Data Analysis (CAIDA), 2003. *The Spread of the Sapphire/Slammer Worm*. [Online] Available at: HYPERLINK "http://www.caida.org/publications/papers/2003/sapphire/" http://www.caida.org/publications/papers/2003/sapphire/ [Accessed 14 June 2014].

[8] Daniel, B., 2015. *Log Analysis using OSSEC*. [Online] Available at: " http://ossec.net/ossec-docs/auscert-2007-dcid.pdf [Accessed 10 July 2016].

[9] Ditcheva, B. & Fowler, L., 2005. *Signature-based Intrusion Detection*. [Online] University of North Carolina Available at http://www.cs.unc.edu/~jeffay/courses/nidsS05/slides/6-Sig-based-Detection.pdf [Accessed 29 June 2014].

[10] IEEE, 2014. *Bandwidth Trends on the Internet… A Cable Data Vendor's Perspective*. [Online] Available at: http://www.ieee802.org/3/ad_hoc/bwa/public/sep11/cloonan_01a_0911.pdf [Accessed 30 November 2014].

[11] Jaquier, C., 2013. *Fail2Ban User Manual*. [Online] Available at: http://www.fail2ban.org/wiki/index.php/MANUAL_0_8 [Accessed 10 July 2016].

[12] Khan, M.A., 2016. A survey of security issues for cloud computing. *Journal of Network and Computer Applications*.

[13] Kim, H.-A. & Karp, B., 2004. Autograph: Toward Automated, Distributed Worm Signature Detection. In *USENIX Security Conference*. Carlifonia, 2004.

[14] Kolesnikov, A. & Lee, W., 2004. *Advanced Polymorphic Worms: Evading IDS by Blending in with Normal Traffic*. Technical. Georgia : Georgia Tech College of Computing Georgia Tech College of Computing.

[15] Kreibich, C. & Crowcroft, J., 2003. Honeycomb - Creating Intrusion Detection Signatures Using Honeypots. In *Second Workshop on Hot Topics in Networks (Hotnets II)*. Boston , 2003. Boston Publishers.

[16] Kruegel, C., 2005. Polymorphic Worm Detection Using Structural Information of Executables. In *International Symposium on Recent Advances in Intrusion Detection (RAID)*. Seatle, 2005. WA.

[17] Kruegel, C., Mutz, W., Robertson, F. & Valeur, F., 2003. Bayesian event classification for intrusion detection. In *19th Annual Computer Security Applications Conference*. Las Vegas, 2003.

[18] Lindqvist, U. & Porras, P., 2001. eXpert-BSM: A Host-based Intrusion Detection Solution for Sun Solaris*. In *Proceedings of the 17th Annual Computer Security Applications Conference*. New Orleans, Louisiana, 2001. IEEE Computer Society.

[19] Li, Z., 2006. Hamsa: Fast Signature Generation for Zero-day Polymorphic Worms with Provable Attack Resilience. In *IEEE Symposium on Security and Privacy*. California, 2006. Oakland.

[20] Moore, D., Shannon, C., Voelker, G.M. & Savage, S., 2003. *Internet Quarantine: Requirements for Containing Self-Propagating Code*. [Online] Available at: http://cseweb.ucsd.edu/~savage/papers/Infocom03.pdf [Accessed 14 June 2014].

[21] Nazario, J., 2004. Defense and Detection Strategies against Internet Worms. Artech House.

[22] Newsome, J., Karp, B. & Song, D., 2005. Polygraph: Automatically Generating Signatures for Polymorphic Worms. In *IEEE Symposium on Security and Privacy*. carlifonia, 2005. Carnegie Mellon University.

[23] Newsome, J., Karp, B. & Song, D., 2006. Paragraph: Thwarting Signature Learning by Training Maliciously. In *Ninth International Symposium on Recent Advances in Intrusion Detection (RAID 2006)*. Hamburg, 2006. Germany Pub.

[24] Newsome, J. & Song, D., 2005. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. In *12th Annual Network and Distributed System Security Symposium*. Milan, 2005. University of Milan.

[25] OSSEC Project Team, 2015. *OSSEC Architecture (How it Works)*. [Online] Available at: http://ossec.github.io/docs/manual/ossec-architecture.html?page_id=169 [Accessed 10 July 2016].

[26] Paxson, V., 1998. Bro: A System for Detecting Network Intruders in Real-Time. In *7th USENIX Security Symposium*. San Antonio, 1998. Texas Publishers.

[27] Paxson, V., 1999. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks*, pp.2435–63.

[28] Perdisci, R., 2006. Misleading Worm Signature Generators Using Deliberate Noise Injection. In *2006 IEEE Symposium on Security and Privacy*. Washington DC, 2006.

[29] Quadrant Information Security, 2015. *The Sagan Log Analysis Engine*. [Online] Available at: https://quadrantsec.com/sagan_log_analysis_engine/ [Accessed 10 July 2016].

[30] Rajan, S., Ginkel, W. & Sundaresan, N., 2013. Expanded Top Ten Big Data Security and Privacy Challenges. *Cloud Security Alliance*.

[31] Riquet, D., Grimaud, G. & Hauspie, M., 2012. Large-scale coordinated attacks: Impact on the cloud security. In *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. Lille, 2012.

[32] Roesch, M., 1999. Snort - Lightweight Intrusion Detection for Networks. In *In LISA '99 : Proceedings of 13th USENIX conference on System administration*. Berkeley, 1999. USENIX Association.

[33] Singh, S., Estan, C. & Varghese, G., 2003. *The EarlyBird System for Real-time Detection of Unknown Worms*. San Diego: University of California University of California.

[34] Smaha, S.E., 1991. Haystack: An intrusion detection system. In *IEEE Fourth Aerospace Computer Security Applications*. Orlando, FL, 1991.

[35] Sly Technologies, 2016. *jNetPcap OpenSource DPI SDK*. [Online] Available at: http://jnetpcap.com/ [Accessed 30 April 2016].

[36] Tutorials Point (I) Pvt. Ltd, 2016. *MapReduce Tutorial*. [Online] Available at: https://www.tutorialspoint.com/map_reduce/map_reduce_tutorial.pdf [Accessed 30 April 2016].

[37] White, T., 2012. Hadoop: The Definitive Guide. In M. Loukides & M. Blanchette, eds. *Hadoop: The Definitive Guide*. 3rd ed. California: O'Reilly Media, Inc..

[38] Wichmann, R., 2006. *The SAMHAIN file integrity / host-based intrusion detection system*. [Online] Available at: HYPERLINK "http://www.la-samhna.de/samhain" http://www.la-samhna.de/samhain [Accessed 10 July 2016].

[39] Zhang, W., Yang, Q. and Geng, Y. (2015) 'A Survey of Anomaly Detection Methods in Networks' Jinan, China.

[40] Zou, C.C., Gao, L., Gong, W. & Towsley, D., 2003. Monitoring and early warning for internet worms. In *10th ACM conference on Computer and communications security*. Washington D.C., 2003. ACM Press.

[41] Baer, A., Casas, P., DÕAlconzo, A., Fiadino, P., Golab, L., Mellia, M. and Schikuta, E. (2016) 'DBStream*: A Holistic Approach to Large-Scale Network Traffic Monitoring and Analysis'*, Computer Networks, April

[42] Chen, Z., Xu, G., Mahalingam, V., Ge, L., Nguyen, J., Yu, W. and Lu, C. (2015) 'A Cloud Computing Based Network Monitoring and Threat Detection System for Critical Infrastructures', *Big Data Research*, November