1. INTRODUCTION

A virtualized infrastructure [11] consists of virtual machines (VM's) [17] that rely upon the software defined multi-instance resources of the hosting hardware. The ability to pool different computing resources as well as enable on-demand resource scaling has led to the widespread deployment of virtualized infrastructures as an important provisioning to cloud computing services. This has become an attractive target for cyber attackers to launch attacks for illegal access. Sophisticated attacks such as VENOM [6] (Virtualized Environment Neglected Operations Manipulation) have been performed which allow an attacker to break out of a guest VM's and access the underlying hypervisor. The vulnerability is serious because it pierces a key protection that many cloud service providers use to segregate one customer's data from another's. If attackers with access to one virtualized environment can escape to the underlying operating system, they could potentially access all other virtual environments. In the process, they would be undermining one of the fundamental guarantees of virtual machines. In addition, attacks such as HEARTBLEED [5] vulnerability was one of the most impactful. Heartbleed allows attackers to read sensitive memory from vulnerable servers, potentially including cryptographic keys, login credentials, and other private data. In addition to these attacks SHELLSHOCK which exploits the vulnerabilities within the operating system can also be used against the virtualized infrastructure to obtain login details of the guest VM's and perform attacks. CLOUDAV [2] antivirus software is one of the most widely used tools for detecting and stopping malicious and unwanted software.

Security analytics applies analytics on the various logs which are obtained at different points within the network to determine attack presence. By leveraging the huge amounts of logs generated by various security systems (e.g., intrusion detection systems (IDS), security information and event management (SIEM), etc.), applying big data analytics will be able to detect attacks which are not discovered through signature or rule-based detection methods.

To overcome these limitations, this study discusses a novel Big Data based Security Analytics (BDSA) [3] approach to protecting virtualized infrastructures

against advanced attacks. By making use of the network logs as well as the user application logs collected from the guest VM's which are stored in a Hadoop Distributed File System (HDFS), our BDSA approach first extracts attack features through graph-based event correlation, a MapReduce parser [14] based identification of potential attack paths and then ascertains attack presence through two-step machine learning methods, namely logistic regression and belief propagation.

2. LITERATURE SURVEY

[1] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," Communications of the ACM, vol. 51, no. 1, pp.107–113, 2008.

This survey says about MapReduce Parsers. MapReduce parsers are programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, the design a new abstraction that allows to express the simple computations that are trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Abstractions are inspired by the map and reduce primitives present in Lisp and many other functional languages. The computations involved applying a *map* operation to each logical record. The examples of this approach are

1. **Distributed Grep:** The map function emits a line if it matches a supplied pattern. The reduce function is an identity function that just copies the supplied intermediate data to the output.
2. **Reverse Web-Link Graph:** The map function outputs (target, source) pairs for each link to a target URL found in a page named source. The reduce function concatenates the list of all source URLs associated with a given target URL and emits the pair (target, list (source)).

Consider map task M and reduce tasks R_1 and R_2. Let $e(R_i)$ be the execution of R_i that committed (there is exactly one such execution). The weaker semantics arise because $e(R_1)$ may have read the output produced by one execution of M and $e(R_2)$ may have read the output produced by a different execution of M. The approach has many different implementations. The right implementation is depend on their environment. This study discussed about an implementation used at Google. In this example they used large clusters of commodity PC's connected

together in the suitable environment. The steps in the process are as follows. First step is that a cluster consists of hundreds or thousands of machines, and therefore machine failures are common. The second step is that storage is provided by inexpensive IDE disks attached directly to individual machines. The last step is that users submit jobs to a scheduling system. Each job consists of a set of tasks, and is mapped by the scheduler to a set of available machines within a cluster. The limitations of MapReduce parsers are

- If no response is received from a worker in a certain time then that worker is considered as failure.
- Any map task or reduce task on a failure worker then that process is set to an idle state and becomes an eligible for rescheduling.
- The completed map tasks are again executed because their output is stored on the local disks.

[2] J. Oberheide, E. Cooke, and F. Jahanian, "Cloudav: N-version antivirus in the network cloud." in USENIX Security Symposium, San Jose, California, USA, 2008, pp. 91–106.

Detecting malicious software is a complex problem. The vast, ever-increasing ecosystem of malicious software and tools presents a daunting challenge for network operators and IT administrators. Antivirus software is one of the most widely used tools for detecting and stopping malicious and unwanted software. However, the elevating sophistication of modern malicious software means that it is increasingly challenging for any single vendor to develop signatures for every new threat.

This study discusses about suggesting a new model for the detection functionality currently performed by host-based antivirus software. This shift is characterized by two key changes.

1. **Antivirus as a network service:** First, the detection capabilities currently provided by host-based antivirus software can be more efficiently and effectively provided as an *in-cloud network service*. Instead of running complex analysis software on every end host, the authors suggest that each end host run a lightweight process to detect new files, send them to a network

service for analysis, and then permit access or quarantine them based on a report returned by the network service.

2. **N-version protection:** Second, the identification of malicious and unwanted software should be determined by multiple, heterogeneous detection engines in parallel. Similar to the idea of N-version programming, the author's proposed the notion of N-version protection and suggests that malware detection systems should leverage the detection capabilities of multiple, heterogeneous detection engines to more effectively determine malicious and unwanted files.

To address the ever-growing sophistication and threat of modern malicious software, the authors proposed a new model for antivirus deployment by providing antivirus functionality as a network service using N-version protection. The benefits of this approach are better detection of malicious software, enhanced forensics capabilities, Retroscopic detection, improved deployability and management.

This novel paradigm provides significant advantages over traditional host-based antivirus including better detection of malicious software, enhanced forensics capabilities, retrospective detection, and improved deployability and management. Using a production implementation and real-world deployment of the CloudAV platform, the evaluation of the effectiveness of the proposed architecture and demonstrated how it provides significantly greater protection of end hosts against modern threats.

[3] T. Mahmood and U. Afzal, "Security analytics: big data analytics for cyber security: a review of trends, techniques and tools," in Information assurance (ncia), 2013 2nd national conference on. Rawalpindi, Pakistan: IEEE, 2013, pp. 129–134.

Critical Infrastructures (CIs), such as smart power grids, transport systems, and financial infrastructures, are more and more vulnerable to cyber threats, due to the adoption of commodity computing facilities. Despite the use of several monitoring tools, recent attacks have proven that current defensive mechanisms for CIs are not effective enough against most advanced threats.

Challenges and opportunities are discussed along three main research directions:

- Use of distinct and heterogeneous data sources,
- Monitoring with adaptive granularity, and
- Attack modeling and runtime combination of multiple data analysis techniques.

Over the past years attacker's community has developed smarter worms and root kits to achieve a variety of objectives, which range from credentials compromise to sabotage of physical devices. Cyber threats are targeting extremely diverse critical application domains including e-commerce systems, corporate networks, data center facilities and industrial systems. For example, on July 2010, the well-known Stuxnet cyber-attack was launched to damage gas centrifuges located at the Natanz fuel enrichment plant in Iran by modifying their speed very quickly and sharply. On August 2012, the Saudi oil giant Aramco was subjected to a large cyber attack1 that affected about 30000 workstations.

Field data represent a rich source of information for improving the security monitoring and protection of future critical infrastructures. Existing monitoring technologies already offer the possibility of collecting different types of data, such as performance, environmental and operational data. The idea of collecting these types of heterogeneous data, and analyze them through a combination os state-of-art attack modeling and data analysis techniques, is promising to improve the accuracy of detection and drive the adaptation of the monitoring itself. In addition, the availability of existing analysis approaches and open, configurable monitoring tools represents a good start for the viability of the proposed framework. However, the achievement of envisioned research objectives requires to face many open issues, such as:

- Need of strategies for changing the monitoring configuration at runtime on the basis of some predefined logic, without being forced to stop and restart the services in charge of gathering and analyzing data.
- Urgency of scalable solutions to combine the outputs generated by the different data analysis techniques, as the ones envisaged in the framework.

Further research is in order, involving the different views and know-how of the researcher's active in these fields.

The limitations of this approach are lack of publicly available data sets and ground truth, urgency of scalable solutions to combine the outputs generated by the different data analysis techniques. A possible solution could consist in leveraging the monitoring of the activities of such privileged accounts to pinpoint on going suspicious activity.

[4] T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda, "Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks," in Proceedings of the 29th Annual Computer Security Applications Conference. New Orleans, Lousiana, USA: ACM, 2013, pp. 199–208.

The use of data mining to mine attack patterns in network logs is used in Beehive security approach. Basically, Beehive detects attacks through correlating logs obtained from different points within the enterprise network. First it collects logs from different points (e.g., web server logs, user logs, IDS logs, etc.) within the enterprise network are collected over a two-week period. The logs are then parsed using known network configuration details to extract 15 features for each host based on the IP addresses of websites accessed, details of the application used, violation of network policy, and changes in network flow characteristics caused by the accesses. Next, the extracted features are represented as a vector, with dimension reduction carried out by principal component analysis (PCA), and finally clustering is undertaken on the feature vectors to determine attack presence.

Beehive is able to detect attacks from large amounts (approximately 1 TB) of log data. However, it is limited in providing prompt threat quarantine and elimination due to its post-factum nature. The BDSA approach has overcome this limitation by detecting attacks in real-time. Graph-based analysis is used in BotCloud to detect botnet attacks. It involves two steps. First Netflow traffic logs obtained over a 48 hour period are represented as a network graph. Then, MapReduce model together with PageRank algorithm is used to identify sub graphs consisting of common source/destination traffic flows. BotCloud does not support real-time threat mitigation/quarantine due to its post_factum length of data.

The approach involves three steps. First well-known botnet code is executed on the testbed and the network traffic over a 48 hour period is collected, and important packet features (e.g., time delay between packet transmissions, packet header length, etc.,) are extracted and stored in an Apache Hive database. With the extracted values, a MapReduce model is used to cluster common features.

Finally a Random Forest classifier is trained on the clustered features using Apache Mahout. This detection scheme is limited in detecting sophisticated attacks which can adapt their communication behaviour to trick the system by mimicking normal communication flow. The BDSA approach has overcome this limitation by obtaining traffic and application logs in real-time, allowing for detection of attacks in real-time and an immediate response to attack presence. The BDSA approach is more insightful as it takes into account the changes both in the characteristics of the applications running within the guest VM's and in the network traffic flow.

[5] **N.Weaver, J.Amann, J. Beekman, M. Payer et al., "The matter of heartbleed," in Proceedings of the 2014 Conference on Internet Measurement Conference. Vancouver, BC, Canada: ACM, 2014, pp. 475–488.**

In March 2014, The Heartbleed vulnerability took the Internet by surprise in April 2014. The vulnerability, one of the most consequential since the advent of the commercial Internet, allowed attackers to remotely read protected memory from an estimated 24–55% of popular HTTPS sites. Researchers found a catastrophic vulnerability in OpenSSL, While OpenSSL has had several notable security issues during its 16 year history, the Heartbleed vulnerability was one of the most impactful. Heartbleed allows attackers to read sensitive memory from vulnerable servers, potentially including cryptographic keys, login credentials, and other private data. The bug is simple to understand and exploit.

OpenSSL is a popular open-source cryptographic library that implements the SSL and TLS protocols. It is widely used by server software to facilitate secure connections for web, email, VPN, and messaging services. The project has faced eight information leak vulnerabilities, four of which allowed attackers to retrieve

plaintext, and two of which exposed private keys. Two of the vulnerabilities arose due to protocol weaknesses; the remainder came from implementation errors.

The OpenSSL implementation of the Heartbeat Extension contained a vulnerability that allowed either end-point to read data following the payload message in its peer's memory by specifying a payload length larger than the amount of data in the Heartbeat Request message. Because the payload length field is two bytes, the peer responds with up to 216 bytes (~64 KB) of memory. The bug itself is simple: the peer trusts the attacker-specified length of an attacker-controlled message.

With the exception of a small handful, the most prominent websites patched within 24 hours. In many ways, this represents an impressive feat. However, the vulnerability scans from potential attackers within 22 hours, and it is likely that popular sites were targeted before the onset of large, indiscriminate scans. Several factors indicate that patching was delayed because the Heartbleed disclosure process unfolded in a hasty and poorly coordinated fashion. Several major operating system vendors were not notified in advance of public disclosure, ultimately leading to delayed user recovery.

The investigation of the attack landscape, found that no evidence of large scale attacks prior to the public disclosure, but vulnerability scans began within 22 hours. The post-disclosure attackers employing several distinct types of attacks from 692 sources, many coming from Amazon EC2 and Chinese ASes. The authors of this study conducted a mass notification of vulnerable hosts, finding a significant positive impact on the patching of hosts to which the notifications are sent, indicating that this type of notification helps reduce global vulnerability. Finally, the study drew upon the analyses to frame what went well and what went poorly in our community's response, providing perspectives on how the approach might respond more effectively to such events in the future.

2.1 Existing Techniques

Existing security approaches for protecting virtualized infrastructures generally include two types, namely malware detection and security analytics.

2.1.1 Malware detection

Malware detection [7] usually involves two steps, first, monitoring hooks are placed at different points within the virtualized infrastructure, and then a regularly-updated attack signature database is used to determine attack presence. Common techniques to detect malware are In-VM and outside-VM interworking approach.

2.1.1.1 In-VM and outside-VM interworking approach to malware detection

This approach consists of an in-VM agent running within the guest VM, and a remote scrutiny server monitoring the VM‟s behavior. The in-VM and outside-VM interworking approach is also used by **Cuckoo Droid,** to detect mobile malware presence on Android devices. Implemented on an emulated Android platform, Cuckoo Droid achieved a detection accuracy of 98.84 %.

2.1.1.2 Hypervisor-assisted malware detection

This scheme is designed to detect botnet activity within the guest VMs. The scheme installs a network sniffer on the hypervisor to monitor external traffic as well as inter-VM traffic. A hypervisor-assisted detection scheme [12] is proposed by using guest application and network flow characteristics.

2.1.2 Security analytics

Security analytics applies on the various logs which are obtained at different points within the network to determine attack presence. Security analytics aims to detect previously undiscovered threats by use of analytic techniques. Common techniques of security analytics include clustering and graph-based event correlation.

2.1.2.1 Clustering for security analytics

Clustering organizes data items in an unlabeled dataset into groups based on their feature similarities. For security analytics, clustering finds a pattern which generalizes the characteristics of data items, ensuring that it is well generalized to detect unknown attacks. Examples of cluster based classifiers include K-means

clustering and k-nearest neighbour's. Finally, an alert system determines attack presence by calculating the confidence levels of each event.

2.1.2.2 Graph-based event correlation

While clustering determines attack presence through grouping common attack characteristics, it is limited in establishing an accurate correlation which may exist between events. This makes it difficult to accurately identify the sequences of events. Graph-based event correlation overcomes this limitation by representing the events from the logs obtained as sequences in a graph. Graphs-based event correlation is presented in the security framework designed to detect attacks within critical infrastructures. The scheme collects events from different sources within the network, and generates a temporal graph model to derive different event correlations for threat detection.

2.2 Limitations

- To detect previously undiscovered attacks, this is often not carried out in real-time and current implementations are intrinsically non scalable.
- Another limitation of existing security approaches is the centralized execution process.
- An in-VM agent detects and passes any suspicious file, which uses the signature database to determine if it is a malware.
- The dependence on a regularly-updated signature database makes it limited in detecting zero-day attacks.
- The completed map tasks are again executed because their output is stored on the local disks. But the reduce tasks are not re-executed because the output is stored in global file system.
- CloudAV antivirus software fails to detect significant percentage of malware.
- By increasing complexity of antivirus software and services have indirectly results in vulnerabilities.
- Difficulty of performing long-running tests on real-world systems.
- The Beehive [4] approach does not allow traffic and application logs in real-time to detect the attacks.
- The Beehive approach is limited in the detecting of the sophisticated attacks.

3. BDSA APPROACH

To overcome these limitations, Thu yein win, Huaglory and Mair discussed about an approach called Big Data based Security Analytics (BDSA) [9] approach to protecting virtualized infrastructures against advanced attacks. The basic idea of this approach is to detect real time malware and root kit attacks via holistic efficient use of all possible information obtained from the virtualized infrastructure, e.g., various network and user application logs. The design principles, which are integral in the development of the BDSA approach to protect virtualized infrastructures, can be elaborated as follows:

Principle 1-Unsupervised classification

The attack detection system should be able to classify potential attack presence based on the data collected from the virtualized infrastructure over time.

Principle 2-Holistic prediction

The attack detection system should be able to identify potential attacks by correlating events on the data collected from multiple sources in the virtualized infrastructure.

Principle 3-Real-time

The attack detection system should be able to ascertain attack presence as immediately as possible so as for the appropriate countermeasures to be taken immediately.

Principle 4-Efficiency

The attack detection system should be able to detect attack presence at a high computational efficiency, i.e., with as little performance overhead as possible.

Principle 5-Deployability

The attack detection system should be readily deployable in production environment with minimal change required to common production environments.

The diagram illustrates about the overall conceptual framework of the Big Data based Security Analytics (BDSA) approach.

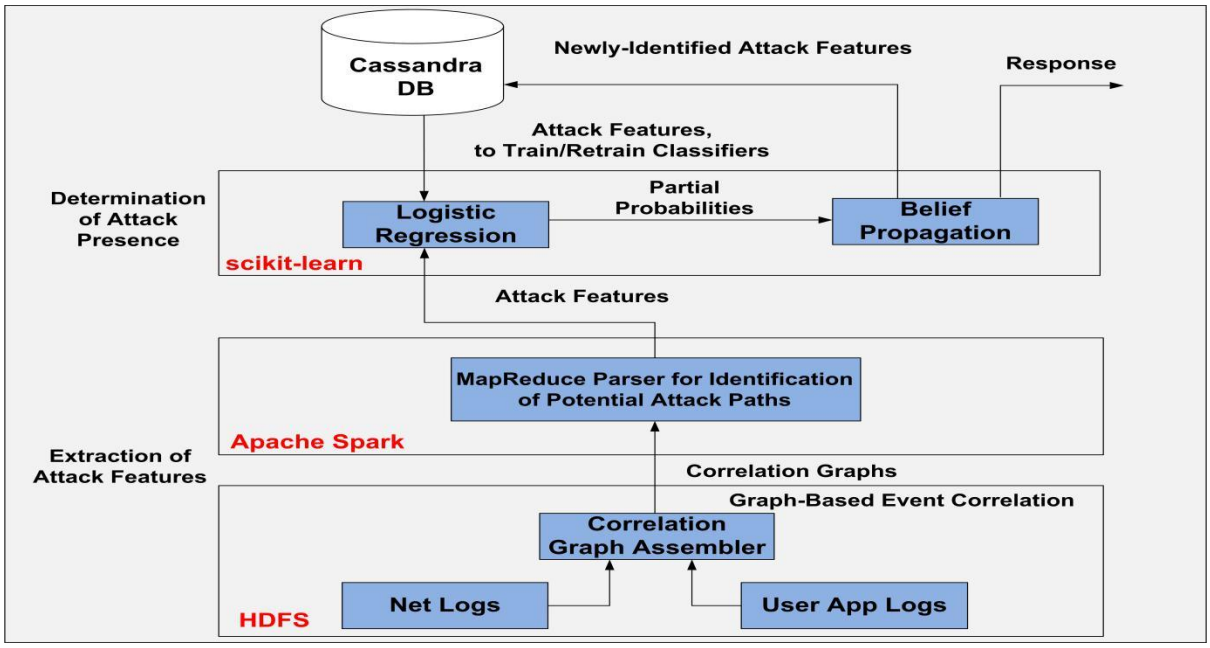

Fig.3.1: Conceptual framework of BDSA approach [Courtesy: Thu yein win]

3.1 Determination of Attack Presence

The stored features are loaded from the Cassandra database to train the logistic regression classifiers. The determination of attack presence consists of two phases, i.e.,

- Training and retraining of **logistic regression** classifiers
- Attack classification using **belief propagation**

3.1.1 Logistic Regression

Logistic regression provides a quick means of as certaining whether a given test data projects to one of the two pre-defined classes, as well as supporting the quick training of a classifier given a training set, (X ~Y), [8] which denotes a series of features versus classes. Logistic regression calculates the probability P of attack at which a feature x projects to one of the two pre-defined classes using the logit function as below.

$$P(y=c/x)=1/(1+e^{-\varepsilon}) \text{ -----} \rightarrow \text{ eq(1)}$$

3.1.2 Belief Propagation

Belief propagation [13] takes into account the conditional probabilities in order to calculate the belief of attack presence within the virtualized environment. This allows a holistic approach for attack detection.

The port and application logistic regression classifiers(i.e., LRport and LRapp, respectively) which are trained using scikit-learn produce as outputs their respective conditional probabilities which calculate the probabilities of each feature belonging to each of the two pre-defined classes (i.e., Attack and Benign) which are of the form as below.

$$P^{port\_change}(Attack/\chi_{port\_change}) = [P^{port\_change}_{Attak}, P^{port\_change}_{Benign}]$$

$$P^{unknown\_exect}(Attack/\chi_{unknown\_exect}) = [P^{unknown\_exect}_{Attack}, P^{unknown\_exect}_{Benign}]$$

$$P^{in\_connect}(Attack/\chi^{in\_connect}_{port}) = [P^{in\_connect}_{Attack}, P^{in\_connect}_{Benign}]$$

$$P^{out\_connect}(Attack/\chi_{out\_connect}port) = [P^{out\_connect}_{Attack}, P^{out\_connect}_{Benign}]$$

After the marginal probabilities are represented as CPTs, the belief (BEL_{Attack}) of the attack nodes state is then calculated using message-passing.

$$BEL_{Attack} = P^{port\_change}(Attack/\chi_{port\_change}) * P^{unknown\_exect}(Attack/\chi_{unknown\_exect}) * P^{in\_connect}(Attack/\chi^{in\_connect}_{port}) * P^{out\_connect}(Attack/\chi^{out\_connect}_{port}) \text{---} \rightarrow \text{eq(2)}$$

3.2 Extraction of Attack Feature

Extraction of attack features through graph-based event correlation and MapReduce parser [10] based identification of potential attack paths.

3.2.1 Graph-Based Event Correlation

In the Extraction of Attack Features phase, first, it carries out Graph-Based Event Correlation. Periodically collected from the guest VM's, network and user application logs are stored in the HDFS. The IP addresses of the guest VM's ware used to obtain the memory process lists on the VM's. **TShark** is used to obtain the network logs containing the traffic flows of the guest VMs. The network logs contain

connection entries describing the guest VM's internal as well as external network connections, namely the source and destination IP addresses(i.e., IP_{source} and $IP_{destination}$) as well as the port numbers(i.e., $Port_{source}$ and $Port_{destination}$) used. Each entry in the network logs is of the format as below.

$$net\_log:= \{IP_{source}, Port_{source}, IP_{destination}, Port_{destination}\} \text{ ------} \rightarrow \text{ eq (3)}$$

The user application logs, on the other hand, contain process entries detailing the applications running within the guest VM's and the port numbers [15] on which the applications are listening for connections. Each entry in the application logs is of the format as below,

$$app\_log:=\{IP_{guest}, APP_{guest}, UserID_{app}, Port_{app}\} \text{ -----} \rightarrow \text{ eq(4)}$$

Where IPguest refers to the guest VM's IP address, App_{guest} refers to the application running on the VM, UserIDApp refers to the user ID to which the user application is running, and $Port_{App}$ refers to the port opened by the application.

The formed correlation graph, consisting of multiple paths is then stored in the HDFS on the HPC node as a new entry, called correlated log, of the format as below,

$$Correlated\_log:=\{IP_{source},Port_{source},IP_{destination},Port_{destination},App_{guest},UserID_{app}\}$$

$$\text{----} \rightarrow \text{eq(5)}$$

3.2.2 MapReduce Parser for Identification of potential attack paths

Identification of potential attack paths is carried out by parsing the correlation graph with a MapReduce model [1]. MapReduce is a distributed programming model which consists of two processes namely Map and Reduce.

In the Map process, key-value pairs of the form (k_i,v_i) are sorted from the correlation graph, where k_i denotes the monitored traffic flow while v_i is the count of occurrence of the traffic flow in the graph.

In the Reduce process the intermediate key-value pairs generated from the Map process, and unify all those key-value pairs. This generates a set of new-variant key-value pairs (k'_i,v'_i), where k_i' represents the unified path for a distinctive source IP and port, while v_i' is the total occurrence counts within the graph.

3.3 Information flows in BDSA approach

It shows how the information flows in BDSA approach.

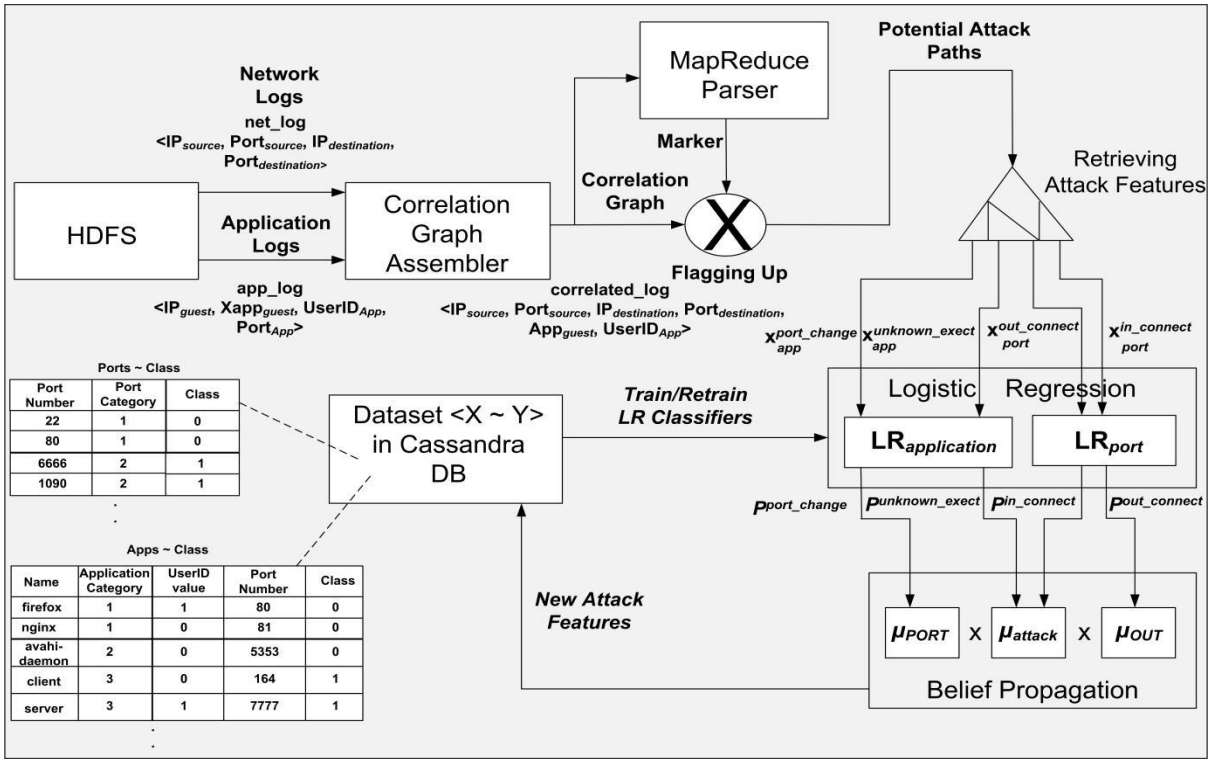

Fig.3.2: Flow of information in BDSA approach [Courtesy: Huaglory]

The execution of the BDSA approach begins by loading all well-known malicious as well as benign port numbers from the distributed Cassandra database. Both of these port types are then used to train a classifier using logistic regression. This allows the approach to determine on-the-fly the probability of an unknown port being malicious, before passing it to the belief propagation framework for final aggregation. A trained logistic classifier is used to determine if any of the attributes are malicious or benign, before passing their respective probabilities to the belief propagation process for final probability aggregation. Belief propagation process takes attack's conditional probabilities with respect to individual attributes to calculate the belief of attack presence, taking into account each conditional probability values to ensure that the value obtained is not influenced only by any conditional probability alone.

3.4 Algorithms

Here in this approach there are 2 algorithms [16]. Those are:

1) Belief propagation.

2) Security Analysis.

3.4.1 Belief propagation

Algorithm 1 provides the pseudocode of the belief propagation algorithm which is used in the BDSA approach.

Algorithm 1 Belief propagation for BDSA

Begin

1: **Initialize:** Create the Bayesian network of attack features using factor graphs.

2: Set the factor graphs F_{exe}, F_{in}, F_{out}, and F_{port} with the placeholder CPT's.

3: **while** True **do**

3.1: Update the factor graphs F_{exe}, F_{in}, F_{out}, and F_{port} with the respective conditional probabilities P_{Attack} and P_{Benign}.

3.2: Calculate $\mu_{exe->Attack}$, $\mu_{in->Attack}$, $\mu_{out->Attack}$, and $\mu_{port->Attack}$

3.3: For unknown_exect and in_connect, calculate F_{attack}.

3.4: Calculate BEL_{Attack} by using the Eq.(1)

3.5: **if** BEL_{Attack}< lower belief **then**

3.5.1: Alarm "attack presence".

3.5.1.1: Update the tables in Cassandra DB with newly identified attack features.

4: End do

End

3.4.2 Security Analysis

Algorithm 2 provides the pseudocode of the Security Analysis which is used in the BDSA approach.

Algorithm 2 Security analytics in BDSA

Begin

1: **Initialize:** Obtain benign and malicious parameters of the attack features from Cassandra DB.

2: Train classifiers for monitored features using Logistic Regression.

3: **while** True **do**

3.1: Collect network and user application logs from guest VM's.

3.2: Filter network log entries using the guest VM's IP addresses.

3.3: Form correlated log.

3.4: Use correlated log to form a correlation graph G.

3.5: Input G into MapReduce parser to identify potential attack paths {attack_paths} which is a sub-set of all graph paths.

3.6: **for** each attack_path in {attack_paths} **do**

3.6.1: i<-0.

3.6.2: **for** each monitored feature $t_{feature}$ in attack_path **do**

3.6.2.1: Calculate P_{port_change}, $P_{unknown\_exect}$, $P_{in\_connect}$, and $P_{out\_connect}$.

3.6.2.2: Pass P_{port_change}, $P_{unknown\_exect}$, $P_{in\_connect}$, and $P_{out\_connect}$ into Step4 of algorithm 1.

3.6.3: End do

3.7: End do

4: End

End

3.5 Advantages of BDSA Approach

- This approach is also implemented by using python environment.
- For experiments, this approach is also implemented on HPC server nodes running Ubuntu OS.

- The ability of the BDSA approach to detect different malware attacks is evaluated by executing the two userspace malware programs as well as the two kernel-level root-kits on the guest VMs.

3.5.1 Comparison between Livewire and BDSA approach:

Livewire is similar approach to threat detection in using external monitoring of guest VM behavior. Livewire periodically polls the guest VM's behaviour through executing remote commands such as obtaining the hardware level information to infer the guest VM's behavior. Due to the similarity in the regard to the BDSA approach in monitoring threat in guest VMs, Livewire is used for the comparative evaluation. The BDSA approach provides a fastest detection time compared to Livewire threat detection approach. Based on Performance comparison between Livewire and BDSA.

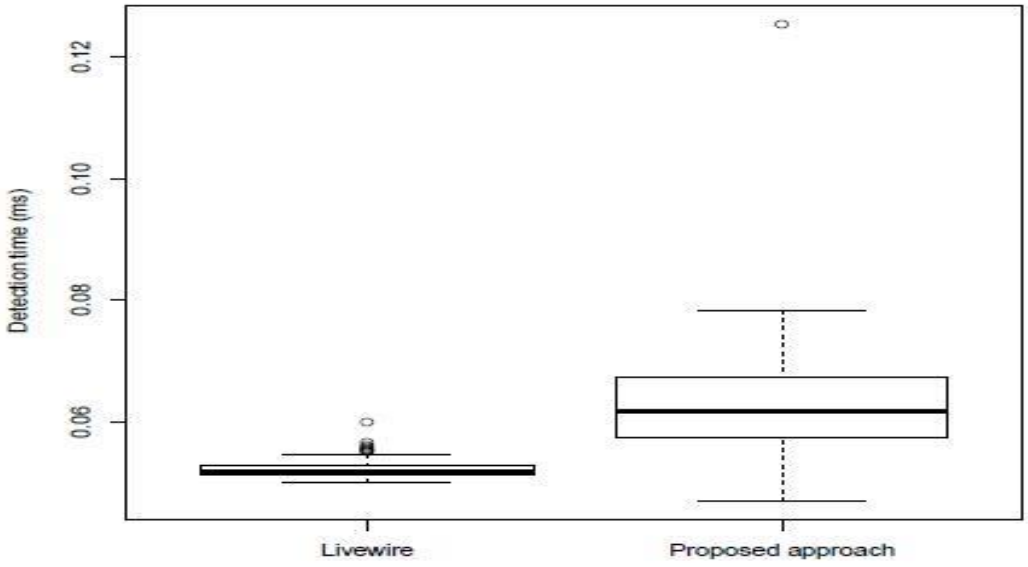

Fig.3.3: Performance evaluation between Livewire and BDSA approach [Courtesy: Mair]

3.6 Comparison of security approaches

This table 3.3 discusses about comparison of security approaches i.e., characteristics, strengths, limitations in Beehive, BotCloud, Botnet approaches with BDSA approach.

Table 3.3 comparison between security approaches [Courtesy: 17]

Approach	**Working characteristics**	**Strengths**	**Limitations**	**Improvement By BDSA approach**
Beehive [4]	Use Of PCA and clustering for attack detection, clustering-based correlation.	Identifying previously unknown attacks.	Post-factum threat Detection	Real time Threat detection
BotCloud	Use of page rank algorithm for C&C botnet detection.	Identifying botnets And their connections.	Limited monitoring Scope	Wide Range scope(network And user logs And details)
Largescale Botnet Detection	Use of common packet characteristics clustering-based correlation.	Detecting well Known botnet attacks.	Limited against hidden attacks.	Detecting Hidden attacks.
BDSA Approach	External monitoring of guest VM behaviour. Graph-based correlation.	Detecting both botnets and in-VM malware.	Occasional latency Increases due to SSH server reset by guest VMs	Real-time Threat detection and Also detecting Hidden attacks.

4. CONCLUSION

A novel Big Data based Security Analytics (BDSA) approach is used to protect virtualized infrastructures in cloud computing against advanced attacks. The BDSA approach constitutes a three phase framework for detecting advanced attacks in real-time. First, the guest VM's network logs as well as user application logs are periodically collected from the guest VM's and are stored in the HDFS. Then, attack features are extracted through correlation graph and MapReduce parser. Finally, two-step machine learning approach is utilized to ascertain attack presence. Logistic regression is applied to calculate attack's conditional probabilities with respect to individual attributes. Furthermore, belief propagation is applied to calculate the overall belief of an attack presence. From the second phase to the third, the extraction of attack features is further strengthened towards the determination of attack presence by the two-step machine learning approach. The use of logistic regression enables the fast calculation of attack's conditional probabilities. More importantly, logistic regression also enables the retraining of the individual logistic regression classifiers using the new attack features as they are obtained from attack detection.

The effectiveness of the BDSA approach is evaluated by testing it against well-known malware and root kit attacks. In all cases, it has been shown that BDSA approach is able to detect them while maintaining a consistent performance overhead with increasing number of guest VM's at an average detection time of approximately 0.06 ms. Tested against Livewire, BDSA approach incurs less performance overhead in attack detection through monitoring the guest VM's behaviour.

REFERENCES

[1]. J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," Communications of the ACM, vol. 51, no. 1, pp.107–113, 2008.

[2]. J. Oberheide, E. Cooke, and F. Jahanian, "Cloudav: N-version antivirus in the network cloud." in USENIX Security Symposium, San Jose, California, USA, 2008, pp. 91–106.

[3]. T. Mahmood and U. Afzal, "Security analytics: big data analytics for cyber security: a review of trends, techniques and tools," in Information assurance (ncia), 2013 2nd national conference on. Rawalpindi, Pakistan: IEEE, 2013, pp. 129–134.

[4]. T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda, "Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks," in Proceedings of the 29th Annual Computer Security Applications Conference. New Orleans, Lousiana, USA: ACM, 2013, pp. 199–208.

[5]. N.Weaver, J.Amann, J. Beekman, M. Payer et al., "The matter of heartbleed," in Proceedings of the 2014 Conference on Internet Measurement Conference. Vancouver, BC, Canada: ACM, 2014, pp. 475–488.

[6]. D. Fisher, "'venom' flaw in virtualization software could lead to vm escapes, data theft," https://threatpost.com/venomflaw-in-virtualization-software-could-lead-to-vm-escapes-datatheft

[7]. X. Wang, Y. Yang, and Y. Zeng, "Accurate mobile malware detection and classification in the cloud," SpringerPlus, vol. 4, no. 1, pp. 1–23, 2015.

[8]. C.-T. Lu, A. P. Boedihardjo, and P. Manalwar, "Exploiting efficient data mining techniques to enhance intrusion detection systems," in Information Reuse and Integration, Conf, 2005. IRI-2005 IEEE International Conference on. Las Vegas, Nevada, USA: IEEE, 2005, pp. 512– 517.

[9]. K. Singh, S. C. Guntuku, A. Thakur, and C. Hota, "Big data analytics framework for peer-to-peer botnet detection using randomforests," Information Sciences, vol. 278, pp. 488–497, 2014.

[10]. J. Francois, S. Wang, W. Bronzi, R. State, and T. Engel, “Botcloud: detecting botnets using mapreduce,” in Information Forensics and Security (WIFS), 2011 IEEE International Workshop on. Foz do Iguacu, Brazil: IEEE, 2011, pp. 1–6.

[11]. L. Aniello, A. Bondavalli, A. Ceccarelli, C. Ciccotelli, M. Cinque, F. Frattini, A. Guzzo, A. Pecchia, A. Pugliese, L. Querzoni et al., “Big data in critical infrastructures security monitoring: Challenges and opportunities,” arXiv preprint arXiv:1405.0325, 2014.

[12]. D. Kirat, G. Vigna, and C. Kruegel, “Barecloud: bare-metal analysis-based evasive malware detection,” in 23rd USENIX Security Symposium (USENIX Security 14), San Diego, California, USA, 2014, pp. 287–301.

[13]. L. Invernizzi, S. Miskovic, R. Torres, S. Saha, S. Lee, M. Mellia, C. Kruegel, and G. Vigna,“Nazca: Detecting malware distribution in large-scale networks,” in Proceedings of the Network and Distributed System Security Symposium (NDSS), San Diego, California, USA, 2014, pp. 23–26.

[14]. J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.

[15]. IANA, “Service name and transport protocol port number registry,” 2015, accessed: 2015-09-30.

[16]. F. R. Kschischang, B. J. Frey, and H.-A.Loeliger, “Factor graphs and the sum-product algorithm,” Information Theory, IEEE Transactions on, vol. 47, no. 2, pp. 498–519, 2001.

[17]. T. Garfinkel, M. Rosenblum et al., “A virtual machine introspection based architecture for intrusion detection,” in Proceedings of the Network and Distributed System Security Symposium (NDSS), vol. 3, San Diego, California, USA, 2003, pp. 191–206.