I. J. Computer Network and Information Security, **2015, 9, 19-31**
Published Online August 2015 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2015.09.03

A Hybrid Real-time Zero-day Attack Detection and Analysis System

Ratinder Kaur and Maninder Singh
Computer Science and Engineering Department, Thapar University, Patiala, 147004, India
Email: ratinder.kaur@thapar.edu, msingh@thapar.edu

***Abstract*—**A zero-day attack poses a serious threat to the Internet security as it exploits zero-day vulnerabilities in the computer systems. Attackers take advantage of the unknown nature of zero-day exploits and use them in conjunction with highly sophisticated and targeted attacks to achieve stealthiness with respect to standard intrusion detection techniques. Thus, it's difficult to defend against such attacks. Present research exhibits various issues and is not able to provide complete solution for the detection and analysis of zero-day attacks. This paper presents a novel hybrid system that integrates anomaly, behavior and signature based techniques for detecting and analyzing zero-day attacks in real-time. It has layered and modular design which helps to achieve high performance, flexibility and scalability. The system is implemented and evaluated against various standard metrics like True Positive Rate (TPR), False Positive Rate (FPR), F-Measure, Total Accuracy (ACC) and Receiver Operating Characteristic (ROC) curve. The result shows high detection rate with nearly zero false positives. Additionally, the proposed system is compared with Honeynet system.

***Index Terms*—**Zero-day Attacks, Unknown Attacks, Intrusion Detection, One-Class SVM, Malware Analysis, Network Security.

I. Introduction

Today the Internet has become a pervasive threat vector for various types of organizations. As new technologies are developed and adopted to meet changing business requirements, sneaky sources lie in wait to exploit vulnerabilities exposed. In recent years, zero-day attacks have been dominating the headlines for political and monetary gains. They are being used as essential success vectors in various sophisticated and targeted attacks like Aurora, Advanced Persistent Threat (APT), Stuxnet, Duqu and Flame. Also, the number of such attacks reported each year increases immensely. According to Symantec's Internet Security Threat Report of 2014 [1] there is 91% increase in targeted attacks campaigns in 2013, 62% increase in the number of security breaches and 23 zero-day vulnerabilities were discovered. Another security threat report by Sophos [2] reported that large tech companies like Apple, Facebook, Microsoft, Twitter and others were targeted with same zero-day Java vulnerability that attacks multiple customers. All such facts and figures represent a serious concern in today's network security. And the zero-day attacks are among the top security concerns that the modern enterprises face today. People talked about zero-day attacks few years back, but today every industry faces it. Another day, another breach and a company losses sensitive data.

To defend against zero-day attacks, the research community has proposed various techniques. These are divided into Statistical-based, Signature-based, Behavior-based and Hybrid techniques [3]. Most of the statistical-based techniques [4, 5, 6] are dependent on attack profiles build from historical data. Due to the static nature of attack profiles, the detection techniques are unable to adapt to the timely changes in the environment. For any change in the data pattern the system will require an updated profile with constant training. Setting the limit (or detection parameters) for judging new observations (new attacks) is a critical step in designing a statistical detection approach since it has a dramatic effect on the quality of the detection. If the threshold value is very narrow, it will frequently be exceeded resulting in a high rate of false positive alarms, and if it is very wide the limit will never be exceeded, resulting in many false negative alarms. At times, the detection parameters are either manually extracted or adjusted to detect new attacks. All these factors, limit the statistical detection approaches to work in offline mode. And hence, they cannot be used for instant detection and protection in real time.

The signature based detection techniques mainly focus on polymorphic worms. There are three types of signatures: content-based, semantic-based and vulnerability-based. The content-based signatures [7, 8, 9] capture the features specific to a worm implementation, thus might not be generic enough and can be evaded by other exploits. Furthermore, various attacks can evade the content-based signatures by misleading signature generation processes by using crafted packet injection into normal traffic. Semantic-based signatures [10] are computationally expensive to generate as compared to approaches based on substrings. Moreover, they cannot be implemented in existing IDS like Snort. Vulnerability-driven signatures [11] capture the characteristics of the vulnerability the worm exploits and are difficult to generate.

Behavior-based techniques [12], looks for the essential characteristics of worms which do not require the examination of payload byte patterns. They suffer from the fact that they cannot effectively capture the context in which the worm program interacts with the real victim machine and they are also prone to evasion [13]. Hybrid techniques [14, 15, 40] combines heuristics and different intrusion detection techniques like signature-based, anomaly-based, etc. to detect zero-day polymorphic worms.

A. *Motivation and Contributions*

The results of recent studies have been the prime motivation for this research. Various techniques were surveyed. The existing zero-day detection techniques do not raise the bar for the attackers, while their cost for the defender in terms of resources that need to be devoted to detection can be significant. Several research projects have addressed the problem of zero-day detection but unfortunately they exhibit one or more problems. As per literature survey there exist certain research proposals that have been promising, but they can also be easily defeated by using minor enhancements to the attack vectors. As attack tools are improving, reliance on minor improvements in the detection processes is insufficient. The current techniques have several drawbacks. Statistical-based detection techniques cannot be used for instant detection and protection in real time. They are dependent on static attack profiles and require manual adjustment of detection parameters. Signature-based techniques are widely used but, need improvement in generating good quality signatures. They suffer from one or more limitations of high false positives, false negatives, reduced sensitivity and specificity. Behavior-based techniques may detect a wide range of novel attacks but they are prone to evasion, computationally expensive and may not effectively capture the context in which the new attacks interact with the real victim machine. Other hybrid techniques combine heuristics and different intrusion detection techniques like signature-based, anomaly-based, etc. to detect zero-day attacks but they also suffer from high false positives, false negatives.

In this paper, a hybrid real-time system is presented which is a novel zero-day attack detection and analysis system. It tries to provide a single solution for the above stated issues and the main contributions of the research reported are:

- The proposed system has been designed and implemented to detect zero-day attacks. To the best of our knowledge this is the first hybrid approach that combines features of all three, anomaly, behavior and signature based detection techniques.
- The layered architecture has been designed which represents a modular and flexible approach that helps to improve system performance and scalability.
- A component-based analysis stub has also been designed to analyze malware. It integrates the advantages of static and dynamic and analysis.
- The proposed system has been tested with a dataset of malware collected from various online malware repositories and has achieved high accuracy with near zero-false positives.

The remainder of the paper is organized as follows. In Section II, related work is summarized. In Section III, the detailed working of the proposed system is presented. Finally in Section IV, describes the results and the paper is concluded in Section V.

II. Related Work

Supervised Learning [16] is a novel method of employing several data mining techniques to detect and classify zero-day malware based on the frequency of Windows API calls. A machine learning framework is developed using eight different classifiers, namely Naïve Bayes (NB) Algorithm, k-Nearest Neighbor (kNN) Algorithm, Sequential Minimal Optimization (SMO) Algorithm with 4 different kernels (SMO-Normalized PolyKernel, SMO-PolyKernel, SMO-Puk, and SMO-Radial Basis Function (RBF)), Backpropagation Neural Networks Algorithm, and J48 decision tree.

Contextual Anomaly Detection [17] is a contextual misuse and anomaly detection prototype to detect zero-day attacks. The contextual misuse detection utilizes similarity with attack context profiles, and the anomaly detection technique identifies new types of attacks using the One Class Nearest Neighbor (1-NN) algorithm. It uses information entropy and linear data transformation to generate feature-based and linear function-based attack profiles [18, 19] and systematically creates contextual relationships between known attacks to generate attack profiles that capture activities of zero-day attacks.

Combined Supervised and Unsupervised Learning [20] technique is presented for zero-day malware detection. It employs machine learning based framework to detect malware using layer 3 and layer 4 network traffic features. It utilizes supervised classification to detect known malware and unsupervised learning to detect new malware and known variants.

Unsupervised Anomaly Detection System [21] is based on clustering and multiple one-class SVM to detect 0-day attacks and to improve the detection rate while maintaining a low false positive rate. It is able to construct intrusion detection models automatically without using labeled training data. In [22] the authors have optimized the values of parameters without predefining. This helps to construct models based on without tuning the parameters, and thus contributes to more practical operations in the real environment.

Integrated Anomaly and Misuse Detection [23] method hierarchically integrates a misuse detection model and an anomaly detection model in a decomposition structure. First, the C4.5 decision tree (DT) is used to create the misuse detection model that is used to decompose the normal training data into smaller subsets. Then, the one-class support vector machine (1-class SVM) is used to create an anomaly detection model in each decomposed region. Throughout the integration, the anomaly detection model indirectly uses the known attack information to enhance its ability when building profiles of normal behavior.

Data Mining based [24] method to detect unknown malware variants. The model is based on the frequency of the appearance of opcode sequences to detect and classify malware. It describes a weighting technique to mine the relevance of each opcode to malicious and benign executables and assess the frequency of each opcode sequence. It then constructs a vector representation of the executables to train machine-learning algorithms to detect unknown malware variants.

SweetBait [9] is a distributed system that is a combination of network intrusion detection and prevention techniques. It employs different types of honeypot sensors, both high-interaction and low-interaction to recognize and capture suspicious traffic. SweetBait automatically generates signatures for random IP address space scanning worms without any prior knowledge. And for the non-scanning worms, Argos is used to do the job. A novel aspect of this signature generation approach is that a forensics shellcode is inserted, replacing malevolent shellcode, to gather useful information about the attack process.

LISABETH [25] automatically generate signatures for polymorphic worms, Lisabeth uses invariant byte analysis of traffic content, as originally proposed in Polygraph [7] and refined by Hamsa [8] Lisabeth leverages on the hypothesis that every worm has its invariant set and that an attacker must insert in all worm samples all the invariants bytes.

In Honeycyber [26] a “Double-honeynet” is proposed as a new detection method to identify zero-day worms and to isolate the attack traffic from innocuous traffic. It uses unlimited Honeynet outbound connections to capture different payloads in every infection of the same worm. It uses Principal Component Analysis (PCA) to determine the most significant substrings that are shared between polymorphic worm instances to use them as signatures [27].

ZASMIN [28] a Zero-day Attack Signature Management Infrastructure is an early detection system for novel network attack detection. To detect unknown network attacks, the system adopted various technologies. To filter malicious traffic it uses dispersion of destination IP address, TCP connection trial count, TCP connection success count and stealth scan trial count. Attack validation is done by call function and instruction spectrum analysis. And it generates signatures using content analysis.

LESG [11] is a network-based automatic worm signature generator that generates length-based signatures for zero day polymorphic worms, which exploits buffer overflow vulnerabilities. The system generates vulnerability - driven signatures at network level without any host level analysis of worm execution or vulnerable programs.

Network-Level Emulation [12] is a heuristic detection method to scan network traffic streams for the presence of previously unknown polymorphic shellcode. Their approach relies on a NIDS-embedded CPU emulator that executes every potential instruction sequence in the inspected traffic, aiming to identify the execution behavior of polymorphic shellcode. The proposed approach is robust to obfuscation techniques like self-modifications and non-self-contained polymorphic shellcodes [29].

SGNET [30] is a distributed framework to collect rich information and download malware for zero-day attacks. It automatically generates approximations of the protocol behavior in form of Finite State Machines (FSMs). Whenever the network interaction falls outside the FSM knowledge (newly observed activity), SGNET takes advantage of a real host to continue the network interaction with the attacker. In that case, the honeypot acts as a proxy for the real host. This allows building samples of network conversation for the new activity that are then used to refine the current FSM knowledge.

ENDMal [31] is an anti-obfuscation, scalable and collaborative malware detection system. It consists of multiple monitors where each monitor takes charge of a network area and receives suspicious programs from end-host. Each monitor uses Iterative Sequence Alignment (ISA) method to defeat malware obfuscation and utilizes Handle dependences and Probabilistic Ordering Dependence (HPOD) technology to represent the program behaviors. All the monitors collaboratively identify the malicious program families by sharing HPOD-based behaviors via RENdezvous-based Sharing infrastructure (RENShare), based on Distributed Hash Tables (DHT).

Hybrid Detection for Zero-day Polymorphic Shellcodes (HDPS) [14] is a hybrid detection approach. It uses an elaborate approach to detect NOP Sleds to be robust against polymorphism, metamorphism and other obfuscations. It employs a heuristic method to detect return address, and achieves high efficiency by incorporating Markov Model to detect executable codes.

Honeyfarm [15] is a hybrid scheme that combines anomaly and signature detection with honeypots. This system takes advantage of existing detection approaches to develop an effective defense against Internet worms. The system works on three levels. At first level signature based detection is used to filter known worm attacks. At second level an anomaly detector is set up to detect any deviation from the normal behavior. In the last level honeypots are deployed to detect zero day attacks. Low interaction honeypots track attacker activities while high interaction honeypots analyze new attacks and vulnerabilities.

SBE [32] is a shellcode detection technique based on emulation and Support Vector Machine. It comprises of two stages: train and classification. In the train phrase, data (including both shellcode and benign data) is obtained and labeled first, then it is emulated and all features (loop, xor, GetPC) are recorded before trimming redundant features with PCA algorithm, and finally a predictive model is achieved after training procedure. In the classification phase, network traffic is emulated and classified by the SVM engine with the model acquired before to separate benign and malicious.

Analysis of Opcode Sequences [33] is an anomaly detection approach which can detect new malware. First, executable files are analyzed in order to extract operation code sequences and then n-gram models are employed to discover essential features from these sequences. The iterative SVM clustering and Support Vector Data Descriptions (SVDDs) are applied to analyze feature vectors obtained and to build a benign software behavior model. This model is then used to detect new malicious executable files.

Two-level automated analysis technique [40, 41] combines anomaly, behavior and signature based techniques for detecting such zero-day attacks. The proposed approach detects obfuscated zero-day attacks with two-level evaluation. At first level the system detects "unknown" by using Honeynet as an anomaly detector and at second level the system confirms "malicious" by analyzing behavior of unknown attack in an emulator. At last it generates new signatures automatically to update other IDS/IPS sensors via global hotfix update.

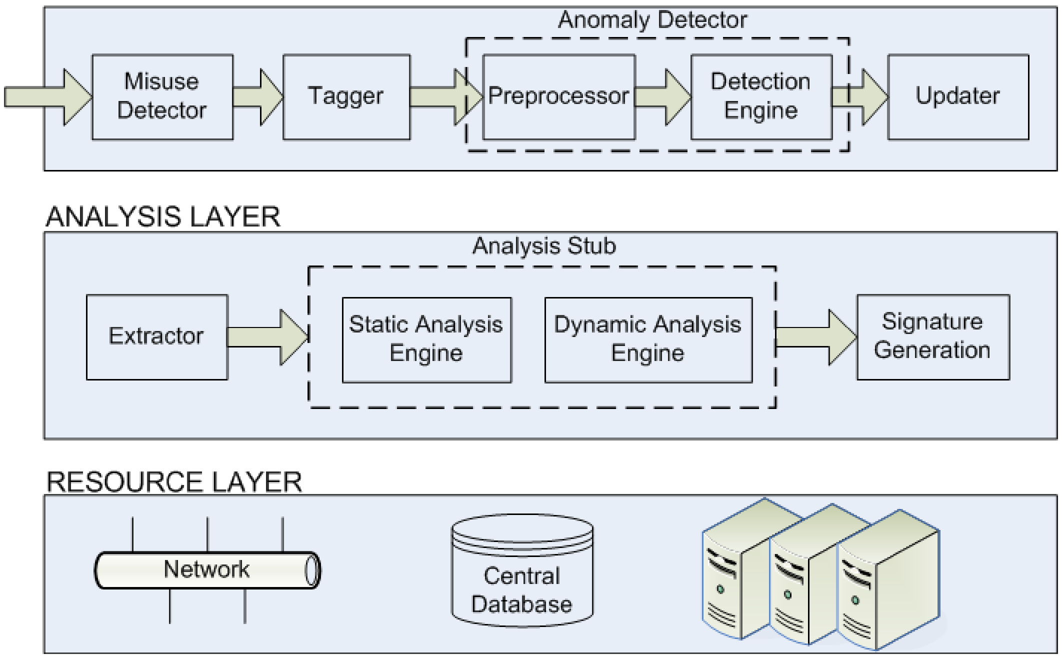

Fig.1. Layered Architecture of Proposed System.

III. Proposed System

Fig. 1 provides an overall layered architecture of the proposed system. It has three layers namely; Detection Layer, Analysis Layer and Resource Layer. The detection layer is responsible for detecting unknown attack. The analysis layer is required to analyze the behavior of captured binary. The resource layer provides hardware resources like network, database and processing servers which helps in execution of components in above two layers. All these layers work in parallel to improve overall performance of the system.

A. Detection Layer

The detection layer is the first layer of defense that detect unknown attacks. It constitutes of the following components:

Misuse Detector: A misuse detector basically models abnormal behavior. It has a well-defined set of malicious behaviors in terms of rules. Misuse detection systems are used to filter all known attacks as they are highly accurate in their decisions and have excellent throughput. In the proposed system, Snort [34] has been used as a misuse detector. Snort is the most popular open source network intrusion prevention and detection system (IDS/IPS). It avoids known intrusions through signature matching. Snort analyzes the packets that arrive to the network interface, match their characteristics with those contained in the rules stored in its rule base. If a specific packet matches the premises of any rule, this rule is executed and a specific action is generated to give notification of the fact. Here the snort drops all the known attack packets and passes filtered traffic for further processing. To drop known attacks snort is used in inline mode. All the "alert" rule actions of well-known attacks were changed to "drop" by a script. All the filtered traffic is then stored in a central database.

Tagger: After filtering known attacks, all the remaining traffic is tainted and passed through an anomaly detector. As the anomaly detectors have either score or label based output techniques, therefore tainting is done to track the network packets which deviate from the normal profile. This way the unknown network packets are identified for further analysis. Traffic tainting is done by a component called "Tagger". It monitors all filtered traffic, tags it and sends it to the preprocessor. The tagger creates a new identifier based on 16-bit hash of a packet. The tag value and label for the filtered packet is stored in a table <*Tag*, *Label*> for later use. The value for *Tag* is calculated for the 6-tuple *{arrival_time, src_ip, dst_ip, src_port, dst_port, and protocol}* by using a fast and effective method of "XOR and shift". The *Label* field is updated later with the result of detection engine as, *<1: anomaly, 0: benign>*.

Anomaly Detector: A misuse detector is unable to respond against unknown attacks. So, to overcome this shortcoming, anomaly detector is used in the next step. Anomaly detector model legitimate network traffic in order to obtain potential deviations from the normal profile. Each deviation that is found significant enough is considered for further analysis. Also establishing a "good" network profile makes it easier to spot previously unknown bad behavior. Therefore, modified Snort-AD is deployed as an anomaly detector. The challenge with using Snort-AD is that the probability of detecting new attacks is low. So there is a need to improve this by modifying preprocessor of Snort-AD in order to increase the probability of detecting new or unknown anomalies. The modified pre-processor receives the filtered tagged packets from the tagger and processes them. It extracts features, identifies most relevant parts of network traffic and normalizes data before sending to the detection engine. For constructing a candidate set of traffic features, total 17 significant and essential features were extracted and stored in a log file. These features identify relevant network traffic characteristics that may be part of a zero-day attack. The extracted network traffic features are listed in Table 1.

Table 1. Extracted Features

Features	Description
Duration	the length (number of seconds) of the connection
Protocol_type	type of the protocol, e.g. tcp, udp, etc.
Service	the connection's service type, e.g., http, telnet, etc
Source bytes	the number of data bytes sent by the source IP address
Destination bytes	the number of data bytes sent by the destination IP address
Count	the number of connections whose source IP address and destination IP address are the same to those of the current connection in the past two seconds.
Same_srv_rate	percentage of connections to the same service in Count feature
Serror_rate	percentage of connections that have SYN errors in Count feature
Srv_serror_rate	percentage of connections that have SYN errors in Srv_count(the number of connections whose service type is the same to that of the current connection in the past two seconds) feature
Dst_host_count	among the past 100 connections whose destination IP address is the same to that of the current connection, the number of connections whose source IP address is also the same to that of the current connection.
Dst_host_srv_count	among the past 100 connections whose destination IP address is the same to that of the current connection, the number of connections whose service type is also the same to that of the current connection
Dst_host_same_src_port_rate	percentage of connections whose source port is the same to that of the current connection in Dst_host_count feature
Dst_host_serror_rate	percentage of connections that have SYN errors in Dst_host _count feature
Dst_host_srv_serror_rate	percentage of connections that SYN errors in Dst_host_ srv_count feature
Flag	the state of the connection at the time the summary was written (which is usually when the connection terminated)
Pkt_count_legitimate_ports	among the past 100 connections whose destination port is same to the port in the legitimate ports list
Pkt_count_unexpected_ports	among the past 100 connections whose destination port is same to the port in unexpected ports list, especially on ports known to be backdoor ports

Detection Engine: It receives the parsed packets from the preprocessor and then compares them with existing good traffic profile and uses machine learning to detect unknown observations. For collecting good traffic a subnet of safe machines in the network have been identified which does not generate or generates less malicious content like network admin's system, analysts' system, HoD's (Head of Department) system and other trusted faculty's or researcher's system. These systems are hardened and all possible security mechanisms are applied. These systems have defined security privileges and policies and does not participate in any malicious activity. A trust value has been assigned to these machines based upon the past experience. This trust value ranges from 1 to 10, with 1 as a compromised machine generating malicious traffic and 10 as fully hardened with no security loopholes. For e.g., the network admin's computer system has a 9 trust value and the analyst's computer system has trust value of 8. All the traffic generated by this subnet is stored in a central database as "known-good" traffic. An approximate of 50 GB raw network traffic is collected from this trusted subnet in Thapar University. This data is then used to train the machine learning algorithm implemented in the detection engine. The preprocessor extracts similar 17 statistical features from the trusted traffic to construct a good traffic profile. The detection engine then applies machine learning on two types of data, i.e., known-good traffic (from trusted subnet) and filtered traffic (from Snort), to detect zero-day attack. Fig.2 represents the creation of good profile.

Support Vector Machines (SVM) [35, 36] is one of the most developed machine learning techniques. They have been succeeded in many applied applications due to their good theoretical properties in generalization and convergence as well as due to their excellent performance in some hard problems. Let there be the only data of one class and the aim is to test new data and to find out whether it is alike or not like the training data then, the best method is to use 1-class SVM [37]. It is easy to gather training data for normal situations but a collection of all possible abnormal scenarios is difficult, or just impossible. To deal with such problem in detection of zero-day attacks, 1-class SVM is used. By just providing the normal traffic data, an algorithm will create a representational model of this data. If the new encountered traffic data is too different (based upon some measurement), from the model, it will be labeled as out-of-class.

Given the unlabeled l data points, $\{x_1, \ldots, x_2\}$ where $x_i \in R^n$; 1-class SVM maps the data points x_i into the feature space by using some non-linear mapping $\phi(x_i)$, and finds a hypersphere which contains most of the data points in the feature space. Fig. 3 shows the formal illustration of the hypersphere model. It is formulated with the center c and the radius $R > 0$ in the feature space, of which the volume R^2 is minimized. The data points that lies outside the hypersphere are regarded as anomalies.

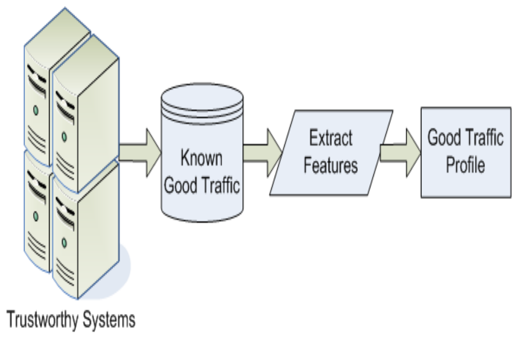

Fig.2. Creation of Good Profile.

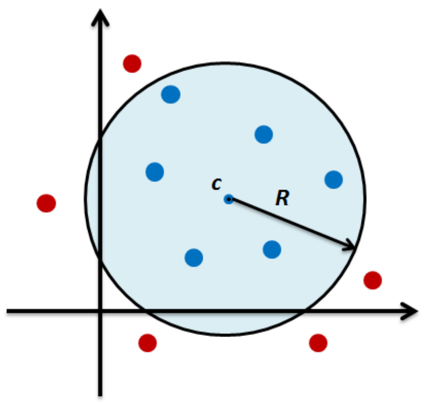

Fig.3. Hypersphere Model.

Mathematically the problem of fitting a hypersphere around the data is formalized as:

$$\min_{R\in\Re,\xi\in\Re^l,c\in\mathcal{F}} \; R^2 + \frac{1}{vl}\sum_{i=1}^{l}\xi_i,$$

subject to: $\|\phi(x_i - c)\| \le R^2 + \xi_i,$

$$\xi_i \ge 0, i = 1, \cdots, l. \qquad (1)$$

To prevent SVM from over-fitting with noisy data, the non-negative slack variables ξ_i are introduced to allow some data points to lie on the "wrong" side of the hypersphere. Also, the parameter $v \in [0, 1]$ determines the tradeoff between the radius of the hypersphere and the number of the data points that belong to the hypersphere. When v is small, more data is put into the hypersphere. When v is larger, its size decreases. Since the center c belongs to the possibly high-dimensional feature space, it is difficult to solve the fundamental equation (1) directly. Instead, it is possible to solve the fundamental problem by its dual form with kernel functions, $k(x,y)$ as in (2).

$$\max_{\alpha\in\Re^l} \sum_{i,j=1}^{l} \alpha_i\alpha_j k(x_i, x_j) - \sum_{i=1}^{l} \alpha_i k(x_i, x_j)$$

subject to: $\sum_{i=1}^{l} \alpha_i = 1,$

$$0 \le \alpha_i \le \frac{1}{vl}, i = 1, \cdots, l. \qquad (2)$$

After finding a hypersphere data can be classified as either normal or attack. In this classification, the following decision function, whether point x in the testing data is normal (i.e., inside of the hypersphere), is used as given in (3).

$$f(x) = sgn\left(R^2 - \sum_{i,j=1}^{l} \alpha_i\alpha_j k(x_i, x_j) + 2\sum_i \alpha_i k(x_i, x) - k(x, x)\right) \qquad (3)$$

The points with $f(x) = -1$ are considered to be anomalies because this means that they exist outside of the hypersphere. Otherwise they are considered to be normal, because they are members of the hypersphere. LibSVM SDK has been used to implement 1-class SVM.

Updater: This component serves as the output-plugin for anomaly detector and pushes the result to a central database. Let P_i be one filtered packet. The tagger gives it a tag, T_i. After feature extraction and comparison with "good" profile, the detection engine output results as $<T_i, 1>$ meaning, that the packet with tag T_i is anomalous. The same information is updated in "Tagger" table and the Label entry changes to *1*. The corresponding packet P_i with tag T_i is saved on the file system with same name as the tag value. This packet is then processed by the second layer for further investigation.

Algorithm 1. Explains the working of detection layer.

```
Algorithm 1 Detection Layer

1: procedure Detection()
2:     for Pcur in P do
3:         Snort(Pcur);
4:         TagPacket();
5:         NormalizeData();
6:         DetectionEngine();
7:     end for
8: end procedure
9: function FilteredPacket Snort(currentPacket)
10:     if (pkt_content.matches(snort_rules)) then
11:         drop(currentPacket);
12:         return filtered_Pkt;
13:     end if
14: end function
15: function TagPacket()
16:     Calculate Tag = HASH(arrival_time, src_ip, dst_ip, src_port, dst_port, protocol);
17:     UpdateDatabase(Tag);
18:     return Tagged_Pkt;
19: end function
20: function String NormalizeData()
21:     for Tagged_Pkt from 1 to N do
22:         PARSED PKT='Use snort preprocessor module for parsing packet'.
23:         Filter features defined in Table 1 from PARSED_PKT.
24:         Arrange the features in decimal format understandable by SVM.
25:     end for
26:     return string;
27: end function
28: function DetectionEngine(String normalizeData, boolean isTrain)
29:     if (isTrain) then
30:         for Trust_Value > threshold do
31:             Capture good data from trustworthy systems.
32:             Extract features and update the SVM database to act as profiler.
33:         end for
34:     else
35:         boolean unknown = Compare the normalizedData with data as per SVM profile
36:         if unknown then
37:             resultSet = Form the INSERT statement.
38:             UpdateDatabase();
39:         end if
40:     end if
41: end function
42: function UpdateDatabase(String resultSet)
43:     INSERT resultset.
44:     Save packet pkt with same Tag value at /usr/home/packets.
45: end function
```

B. Analysis Layer

This layer is responsible for analyzing malicious behavior of a zero-day malware. This layer comprises of following components:

Extractor: The extractor on receiving suspicious packet P_i, parses it and extracts the malicious $binary_i$ for further analysis. The head of the packet has an initial header type, here in this case it is Ethernet. The structure of each header is known to identify the location of fields containing the current header length and the next header type. Ethernet header contains information about next header type i.e. IP header and length of current Ethernet frame is 14. These two values are used to locate the position of IP header in the packet. This process is applied to subsequent headers and is repeated till all headers are processed. In the end, the application protocol stream is processed to strip the last header and to extract the data. This data is then saved as a binary file and is sent to next component for detailed examination.

Analysis Stub: Today, there is really no automated way to understand completely the malicious behavior of a zero-day malware. There is no single best approach for malware analysis so it demands to combine static and dynamic analysis techniques. The analysis stub in the proposed system combines various static and dynamic analysis functionalities in a component based architecture, where any functionality can be replaced in the future. Different analysis features integrated into the system to behave as a single unit. The integrated functionalities work together automatically to provide detailed insight about the malware behavior. The captured binary is fed to analysis stub for automated static and dynamic analysis. And the analysis result of all the procedures is stored in a central database.

Static Analysis Engine (SAE): It combines static malware analysis functionalities to describe various static features of the captured binary like anti-virus scanning, obfuscation, structure, uniqueness, and strings. SAE is completely modular and this makes it flexible and extensible. With the preliminary static analysis it is possible to extract valuable information that will shape the profile of the malware.

- Antivirus Scanning: Before analyzing the prospective malware, the first step is to run it through multiple antivirus programs that may have already identified it. For this, VirusTotal Public APIs [38] are used to upload the binary for scanning by multiple antivirus engines. VirusTotal provides free checking for viruses and more than 50 different antivirus products and scan engines. If the binary is identified by VirusTotal, then SAE fetches existing analysis results and uploads them to a central database and exits. If the binary is not detected by VirusTotal then, further analysis is done to capture its behavior.
- Obfuscation: Malwares often use obfuscation techniques to evade detection systems. One such popular obfuscation technique is packing. To detect the type of packer employed, a script is written to access a text file containing packer signatures.
- Structure: Any binary executable file includes a header to describe its structure like, the base address/size of code section, data section, list of functions imported, exported, etc. This functionality thus, returns the information on file header, sections and import/export tables.

- Uniqueness: For uniqueness hashes are computed on the captured binary. The main purpose of using this feature is to verify that whether the captured binary is unique or not. The hash is generated and checked in a central database where known hashes of existing binaries are stored. Thus, it avoids duplicated malware samples.
- Strings: The legitimate programs always include many embedded strings but an obfuscated or packed malicious program contains very few strings. So, if Strings return few embedded strings (either make sense or not) then the tested binary is likely to be malicious. This functionality extracts ASCII and Unicode strings in binary file.

Dynamic Analysis Engine (DAE): After describing static properties the binary is passed to DAE for dynamic analysis since static analysis is not foolproof. DAE focuses on behavioral analysis, by executing and monitoring the binary. This helps to understand the nature and the purpose of the malicious binary and reveals which files are read or accessed and which operations has been carried out. DAE comprises of an emulator running 32-bit Windows Operating System. The emulator has an integrated functionality running in analysis component to track running processes, network statistics, system calls, and file system changes. Following are the integrated functionalities in the analysis component:

- Running Processes: When the captured binary is executed, the active processes are monitored in emulator to identify loaded DLLs for individual processes and open handles.
- Network Statistics: It is important to keep a check on network connections and this functionality provides information about active connections established by the running malware. It also records network traffic for malicious communication attempts, such as DNS resolution requests, bot traffic, or downloads.
- System Calls: The system calls provide useful information about a process behavior. So, to intercept and record the system calls which are called by a process and the signals which are received by a process, this feature is used. It monitors interactions between processes and the OS kernel, which include system calls, signal deliveries, and changes of process state.
- File System: This functionality monitors real-time file system and registry activity. It returns list of added, deleted and modified files and registry keys.

Signature Generation: After analysis, new signature for malicious binary is generated in ClamAV format. The easiest way to create ClamAV signature is to use file hash checksums. To create MD5 signature the *--md5* option of the ClamAV command-line sigtool (signature and database management tool) is used and all the signatures are stored in the central database. A small script is written to perform this task automatically. The generated signature has three fields. They are MD5 checksum of the binary, size of the binary and type of binary. The size and type of binary are determined during analysis. The generated signature is then uploaded in a central database for future access. The format for the ClamAV signature thus generated is as:

Signature = <HashString:FileSize:MalwareType>.

Algorithm 2. Explains the working analysis layer.

Algorithm 2 Analysis Layer

```
procedure Analysis()
    List binaries = extractBinary();
    for binary in binaries do
        StaticAnalysis(binary);
        DynamicAnalysis(binary);
    end for
end procedure
function List ExtractBinary()
    Read pkt from /usr/home/packets.
    hdr = initialType
    pos = 0
    while hdr != DONE do
        len = GetHeaderLen(pkt, hdr, pos)
        hdr = GetNextHeaderType(pkt, hdr, pos)
        pos = pos + len
    end while
    Extract MessageBody of the application protocol.
    Save MessageBody into a binary file.
return Packet_Binary;
end function
function StaticAnalysis(binary)
    invoke uploadVirusTotal(binary);
    repeat
        response = getVirusTotalResponse();
    until response==null
    if (detectionRatio != 0) then
        Upload VirusTotal result in database.
        BREAK;
    else Continue;
    end if
    packer = obfuscation(binary);
    Upload packer information
    response= structure();
    Upload response
    hash= calculate_hash(binary);
    if (hash = hash(existingBinary)) then
        BREAK;
    else Upload information in database
    end if
    response[] = strings(binary);
    Upload list of embedded strings in database
end function
function DynamicAnalysis(binary)
    analysis_result = Emulator(binary);
    Upload database.
end function
function SignatureGeneration(binary)
    hashString = Calculate Md5sum on binary
    Size = Determine filesize
    Type = MalwareType
    Signature =< hashString : Size : Type >
end function
```

IV. Implementation and Results

To evaluate the proposed system a prototype was implemented using the Oracle Java6 SDK, Python language, Eclipse IDE, MySql database. Various off-the-shelf solutions have been employed wherever possible in an attempt to combine static analysis and dynamic analysis functionalities. For that many existing utilities were modified and incorporated in the system to work as a single unit. The implementation setup of proposed system comprises of router, IDS/IPS sensor, Ethernet switch, Intranet machines and system components like the detection system, analysis system, emulator machine and central database as depicted in Fig. 4. The detection server receive Internet traffic directly from the router, detects unknown attack and passes results to a central database. From central database the analysis system fetches the current results and accordingly extracts binary for further analysis in QEMU emulator.

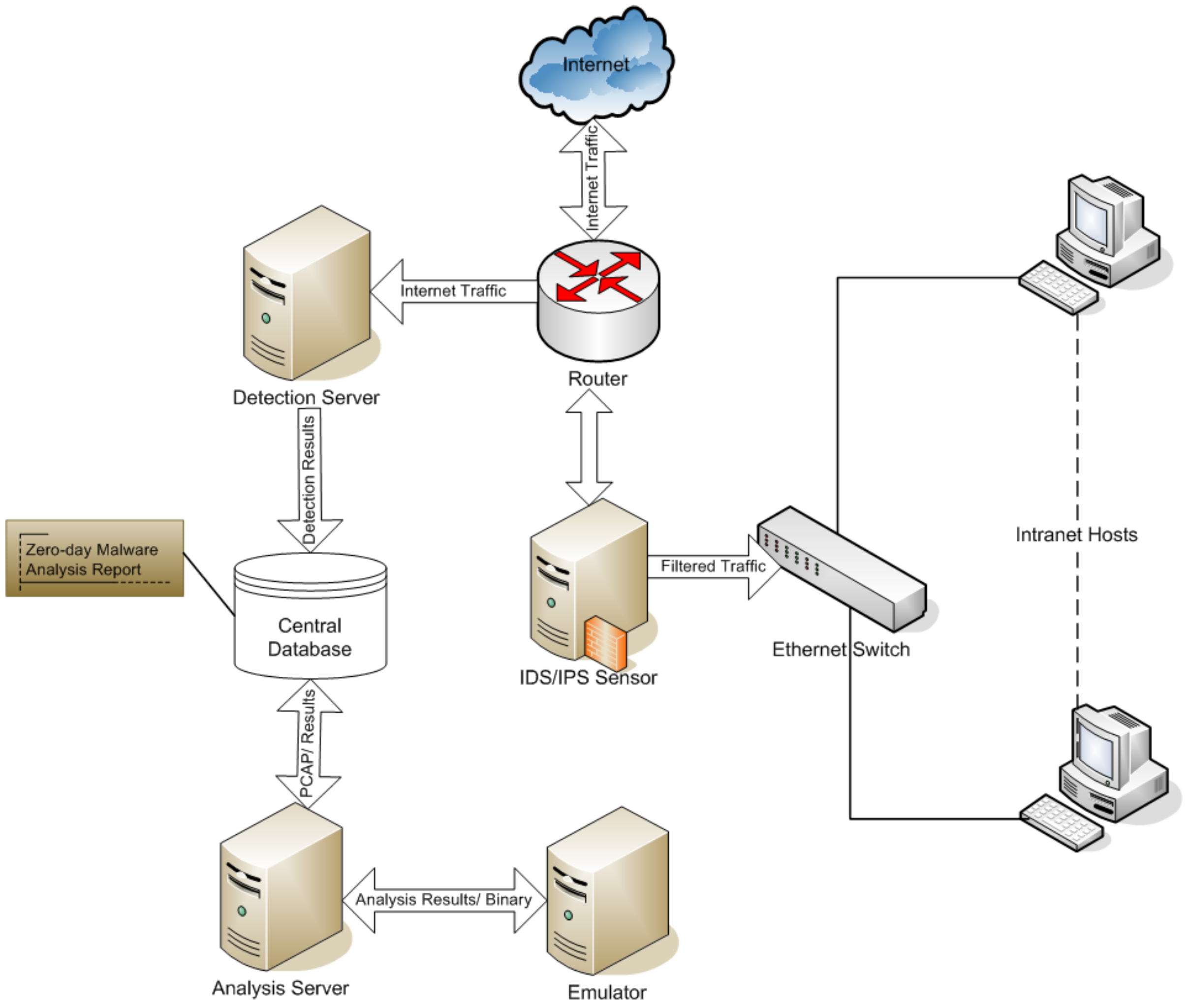

Fig.4. Experimental Setup.

Following standard metrics were used to evaluate the performance of our system: True Positive Rate (TPR), False Positive Rate (FPR), F-Measure, Total Accuracy (ACC) and Receiver Operating Characteristic (ROC) curve. TPR is the percentage of correctly identified malicious code. FPR is the percentage of wrongly identified benign code. F-measure is a measure of a test's accuracy by combining recall (same as TPR) and precision scores into a single measure of performance. ACC is the percentage of absolutely correctly identified code, either positive or negative, divided by the entire number of instances. In ROC curve the TPR rate is plotted in function of the FPR for different points. The ROC curve shows a trade-off between true positive and false positive.

The experimental dataset comprises of 5000 samples in total consisting of 4000 malware samples (both obfuscated & non-obfuscated) and 1000 benign samples. The dataset with obfuscated and unknown malware have been collected from various sources like Honeynet project, VX heavens [39] and other online malware repositories. The benign data samples include application software, system software, legitimate executables, documents and many other user applications. These benign samples were collected from various trusted systems in the production network. The distribution of benign samples are represented in Table 2.

Table 2. Distribution of Benign Samples

Benign Sample Type	No. of Samples
Application Software	300
System Software	250
User Applications	200
Legitimate Executables	150
Legitimate Documents	100

Table 3. Distribution of Zero-day Malware Samples

Malware Type	No. of Samples	Non-Obfuscated	Obfuscated
Virus	1200	500	700
Worm	1000	400	600
Rootkit	100	50	50
Backdoor	600	250	350
Exploit	200	50	150
Trojan	900	350	550

The distribution of malware samples both non-obfuscated and obfuscated are shown in Table 3. Common malware types like viruses, network worms and trojans constitutes the major samples in the dataset. Other types included backdoors, buffer overflow exploits and rootkits. Some of the polymorphic exploits have been generated using Metasploit Framework by applying various encoding engines like Shikata ga nai, XOR encoder and Jmp/Call Additive.

To compute the accuracy of the proposed system both benign and malicious packets were directed to the system set up. These packets were included in tcpdump file for Snort to read. The packets unknown to the system were identified by the detection layer with few escapes. The recorded values of TPR, FPR, Precision, Recall, F-Measure, ACC and ROC for non-obfuscated zero-day malware are presented in Table 4.

Table 4. Detection Accuracy for Zero-day Non-Obfuscated Malware

Malware Type	Non-Obfuscated						
	TPR	FPR	Precision	Recall	F-Measure	ACC	ROC
Virus	0.937	0.04	0.961	0.937	0.95	0.951	0.965
Worm	0.966	0.05	0.954	0.966	0.93	0.944	0.934
Rootkit	0.984	0.023	0.99	0.984	0.986	0.971	0.981
Backdoor	0.973	0.026	0.98	0.973	0.982	0.976	0.975
Exploit	0.984	0.032	0.968	0.984	0.983	0.985	0.982
Trojan	0.902	0.033	0.959	0.902	0.935	0.956	0.935

Same standard intrusion detection metrics were recorded for obfuscated zero-day malware are shown in Table 5. The overall results were very promising achieving the best detection rate of nearly 98% with 0.02 false positive rate and in the worst case, detection rate was 89% with 0.03 false positive rate. Thus, the proposed system has achieved high accuracy with few (near to zero) false positives.

Table 5. Detection Accuracy for Zero-day Obfuscated Malware

Malware Type	Obfuscated						
	TPR	FPR	Precision	Recall	F-Measure	ACC	ROC
Virus	0.918	0.056	0.946	0.918	0.933	0.93	0.931
Worm	0.94	0.081	0.924	0.94	0.932	0.931	0.927
Rootkit	0.966	0.033	0.97	0.966	0.961	0.964	0.956
Backdoor	0.959	0.061	0.933	0.959	0.945	0.96	0.967
Exploit	0.982	0.301	0.82	0.982	0.894	0.785	0.748
Trojan	0.898	0.035	0.96	0.898	0.928	0.945	0.935

A. Comparison with Honeynet

In this section features of proposed system are compared with Honeynet system in Table 6. A Honeynet is a network setup that invites attackers to compromise the system (honeypots) and do harm in a controlled and isolated environment, while their activities are monitored and studied to increase network security. Honeynet has been found effective against zero day attacks. It identifies the mechanism of a new attack and collects evidence for attacker's activities, which is later analyzed by a human expert. This analysis is done by first preparing a toolkit comprised of (but not limited to) physical or virtual systems, behavioral analysis tools, code analysis tools and online analysis tools. All such tools are run separately with human intervention. This takes time sometimes weeks or months and requires high expertise to report a zero-day attack behavior. To address these issues the proposed system provides a single automated solution combining static and dynamic malware analysis. On the other hand, Honeynet is not a detection system, it only traps and monitors unknown attack activities. The detection layer of proposed system detects zero-day attacks against good traffic profile build from trustworthy systems. The following comparison shows that the proposed system is more efficient in delivering a complete solution to zero-day attack detection and analysis.

Table 6. Comparison with Honeynet System

Techniques → Features ↓	Honeynet System	Proposed System
Known Attack Detection	Snort in honeywall log and report known attacks	Snort in inline mode and VirusTotal is used to keep check on known attacks
Zero-day Attack Detection	The unknown traffic is redirected to honeypots to monitor interactions between the attacker and honeypot	Utilized machine learning algorithm, 1-class SVM to detect unknown attacks that deviate from the good network traffic profile
Obfuscation Detection	The obfuscated binary is allowed to run on honeypot with Sebek to track commands	Detect obfuscation in SAE and later the binary is allowed to run on a real host.
Attack Analysis	Analysis is only done manually	Automated analysis: static, dynamic.
Signature Generation	No	Yes in ClamAV format
Response Time	Manual analysis takes time to analyze the behavior of malicious binary	Layered architecture does detection and analysis in parallel. Further, SAE and DAE provides detailed and useful information for manual analysis (if required). Hence reducing response time.

V. Conclusions and Future Work

In this paper a hybrid real-time zero-day attack detection and analysis system is discussed. The proposed system is a combination of anomaly-based detection, behavior-based detection and signature-based detection techniques. The proposed system addresses the research problems with existing approaches in zero-day attack detection and analysis and tries to provide a complete solution to the whole problem. It does so by a layered designed where each layer is dedicated to a single functionality and works in parallel to improve performance. The system employs 1-class SVM as an anomaly detection technique in detection layer to detect zero-day attacks that diverts from the good traffic profile. The analysis layer in the system captures both static and dynamic behavior of malicious binaries captured in the detection layer. The analysis stub combines both static and dynamic malware analysis functionalities to work as a single unit in a component based architecture where any feature can be replaced in the future. The SAE provides basic information to profile the malicious binary and DAE captures run-time behavior of a malicious binary by executing it in an emulator. The system also generates signatures in ClamAV format.

The proposed system was evaluated by various standard metrics. In experiments it was shown that the system provides the best detection rate of nearly 98% with 0.02 false positives. Furthermore, comparison with Honeynet system depicts that the proposed system will minimize response time to a great extend in zero-day attack detection and analysis. In the future work it is planned to: (1) Achieve scalability and improve throughput of the system by detecting and analyzing multiple zero-day binaries at a time. (2) Address defensive measures against anti-analysis techniques like anti-emulation and to explore multiple execution paths for malware analysis. (3) Generate better and more detailed signatures for zero-day obfuscated binaries in Snort format.

Acknowledgment

The authors would like to thank Tata Consultancy Services (TCS) for their support to this research work. The authors are highly obliged to the Computer Science and Engineering Department of Thapar University, Patiala for rendering their incessant help in providing best infrastructure and work-environment.

References

[1] Symantec, “Internet Security Threat Report,” *Security Response Publications*, vol. 19, April 2014. http://www.symantec.com/content/en/us/enterprise/other_resources/bistr_main_report_v19_21291018.en-us.pdf.

[2] Sophos, “Security Threat Report: Smarter, Shadier, Stealthier Malware” *Sophos Publications*, 2014.

[3] R. Kaur and M. Singh, “A Survey on Zero-Day Polymorphic Worm Detection Techniques”, in *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1520-1549, March 2014.

[4] W. C. Sun and Y. M. Chen, “A Rough Set Approach for Automatic Key Attributes Identification of Zero-day Polymorphic Worms”, in *Expert Systems with Applications: An International Journal*, vol. 36, no. 3, pp. 4672-4679, April 2009.

[5] S. Almotairi, A. Clark and G. Mohay and J. Zimmermann, “A Technique for Detecting New Attacks in Low-Interaction Honeypot Traffic”, *Proc. of the IEEE 4th International Conference on Internet Monitoring and Protection*, Washington DC, USA, 2009, pp. 7-13.

[6] J. Song, H. Takakura and Y. Kwon, “A Generalized Feature Extraction Scheme to Detect 0-day Attacks via IDS Alerts”, *Proc. of the IEEE International Symposium on Applications and the Internet*, Washington, DC, USA, 2008, pp. 55-61.

[7] J. Newsome, B. Karp and D. Song, “Polygraph: Automatically Generating Signatures for Polymorphic Worms”, *Proc. of the IEEE Symposium on Security and Privacy (S&P’05)*, Oakland, CA, 2005, pp. 226-241.

[8] Z. Li, M. Sanghi, Y. Chen, M.Y. Kao and B. Chavez, “Hamsa: Fast Signature Generation for Zero-day Polymorphic Worms with Provable Attack Resilience”, *Proc. of the IEEE Symposium on Security and Privacy (S&P’06)*, Berkeley/Oakland, CA, 2006, pp. 15-47.

[9] G. Portokalidis and H. Bos, “SweetBait: Zero-hour Worm Detection and Containment using Low-and High-Interaction Honeypots”, in *Computer Networks: The International Journal of Computer and*

Telecommunications Networking, vol. 51, no. 5, pp. 1256-1274, April 2007.

[10] C. Kruegel, E. Kirda, D. Mutz, W. Robertson and G. Vigna, "Polymorphic Worm Detection using Structural Information of Executables", *Proc. of the LNCS Springer 8th International Symposium on Recent Advances in Intrusion Detection (RAID'05)*, Seattle, 2005, pp. 207-227.

[11] L. Wang, Z. Li, Y. Chen, Z. Fu and X. Li, "Thwarting Zero-day Polymorphic Worms with Network-level Length-based Signature Generation", in *IEEE/ACM Transactions on Networking (TON)*, vol. 18, no. 1, pp. 53-66, February 2010.

[12] M. Polychronakis, K. G. Anagnostakis and E. P. Markatos, "Network-level Polymorphic Shellcode Detection using Emulation", in *Journal in Computer Virology*, vol. 2, no. 4, pp. 257-274, July 2006.

[13] A. Abbasi, J. Wetzels, W. Bokslag, E. Zambon and S. Etalle, "On Emulation-Based Network Intrusion Detection Systems", *Proc. of the LNCS, Springer 17th International Symposium on Research in Attacks, Intrusions and Defenses (RAID'14)*, Gothenburg, Sweden, 2014, pp. 384-404.

[14] C. Ting, Z. Xiaosong and L. Zhi, "A Hybrid Detection Approach for Zero-day Polymorphic Shellcodes", *Proc. of the IEEE International Conference on E-Business and Information System Security*, Wuhan, 2009, pp. 1-5.

[15] P. Jain and A. Sardana, "Defending against Internet Worms using Honeyfarm", *Proc. CUBE International Information Technology Conference (CUBE'12)*, Pune, India, 2012, pp. 795-800.

[16] M. Alazab, S. Venkatraman, P. Watters and M. Alazab, "Zero-day Malware Detection based on Supervised Learning Algorithms of API call Signatures", *Proc. 9th Australasian Data Mining Conference (AusDM'11)*, Ballarat, Australia, 2011, pp. 171-182.

[17] A. AlEroud and G. Karabatis, "A Contextual Anomaly Detection Approach to Discover Zero-Day Attacks", *Proc. IEEE International Conference on Cyber Security (CYBERSECURITY'12)*, Washington, DC, 2012, pp. 40-45.

[18] A. AlEroud and G. Karabatis, "Detecting Zero-Day Attacks Using Contextual Relations", *Proc. of the LNBIP, Springer 9th International Conference on Knowledge Management in Organizations (KMO'14)*, Santiago, Chile, 2014, pp. 373-385.

[19] A. AlEroud and G. Karabatis, "Toward Zero-Day Attack Identification Using Linear Data Transformation Techniques", *Proc. 7th IEEE International Conference on Software Security and Reliability (SERE'13)*, Gaithersburg, MD, 2013, pp. 159-168.

[20] P. M. Comar, L. Liu, S. Saha, P. N. Tan and A. Nucci, "Combining supervised and unsupervised learning for zero-day malware detection", *Proc. of the IEEE INFOCOM'13*, Turin, 2013, pp. 2022–2030.

[21] J. Song, H. Takakura, Y. Okabe and Y. Kwon, "Unsupervised Anomaly Detection Based on Clustering and Multiple One-class SVM", in *IEICE Transactions on Communications*, vol. E92-B, no. 6, pp.1981–1990, June 2009.

[22] J. Song, H. Takakura, Y. Okabe and K. Nakao, "Toward a More Practical Unsupervised Anomaly Detection System", in *Information Sciences*, vol. 231, pp. 4-14, May 2013.

[23] G. Kim, S. Lee and S. Kim, "A novel hybrid intrusion detection method integrating anomaly detection with misuse detection", in *Expert Systems with Applications*, vol. 41, no. 4, pp. 1690–1700, March 2014.

[24] I. Santos, F. Brezo, X. Ugarte-Pedrero and P. G. Bringas, "Opcode sequences as representation of executables for data-mining-based unknown malware detection", in *Information Sciences*, vol. 231, pp. 64–82, May 2013.

[25] L. Cavallaro, A. Lanzi, L. Mayer and M. Monga, "LISABETH: Automated Content-based Signature Generator for Zero-day Polymorphic Worms", *Proc. of the ACM 4th International Workshop on Software Engineering for Secure Systems*, Leipzig, German, 2008, pp. 41-48.

[26] M. M. Z. E. Mohammed, H. A. Chan and N. Ventura, "Honeycyber: Automated Signature Generation for Zero-day Polymorphic Worms", *Proc. of the IEEE Military Communications Conference (MILCOM' 2008)*, San Diego, CA, 2008, pp. 1-6.

[27] M. M. Z. E. Mohammed, H. A. Chan, N. Ventura, M. Hashim, I. Amin and E. Bashier, "Detection of Zero-day Polymorphic Worms using Principal Component Analysis", *Proc. of the IEEE 6th International Conference on Networking and Services*, Cancun, 2010, pp. 277-281.

[28] I. Kim, D. Kim, B. Kim, Y. Choi, S. Yoon, J. Oh and J. Jang, "A Case Study of Unknown Attack Detection against Zero-day Worm in the Honeynet Environment", *Proc. of the IEEE 11th International Conference on Advanced Communication Technology (ICACT' 2009)*, Phoenix Park, 2009, pp. 1715-1720.

[29] M. Polychronakis, K. G. Anagnostakis and E. P. Markatos, "Emulation-based Detection of Non-self-contained Polymorphic Shellcode", *Proc. of the LNCS Springer 10th International Conference on Recent Advances in Intrusion Detection (RAID'07)*, Gold Goast, Australia, 2007, pp. 87-106.

[30] C. Leita and M. Dacier, *SGNET: A Distributed Infrastructure to Handle Zero-day Exploits*, Technical Report EURECOM+2164, EURECOM institute, France, 2007.

[31] H. Lu, X. Wang, B. Zhao, F. Wang and J. Su, "ENDMal: An anti-obfuscation and collaborative malware detection system using syscall sequences", in *Mathematical and Computer Modelling*, vol. 58, no. 5, pp. 1140–1154, September 2013.

[32] Y. Hou, J.W. Zhuge, D. Xin and W. Feng, "SBE - A Precise Shellcode Detection Engine Based on Emulation and Support Vector Machine", *Proc. of the LNCS, Springer 10th International Conference on Information Security Practice and Experience (ISPEC'14)*, Fuzhou, China, 2014, pp. 159-171.

[33] M. Zolotukhin and T. Hamalainen, "Detection of zero-day malware based on the analysis of opcode sequences", *Proc. of the IEEE 11th International Conference on Consumer Communications and Networking Conference (CCNC'14)*, Las Vegas, Nevada, USA, 2014, pp. 386-391.

[34] M. Roesch, "Snort lightweight intrusion detection for networks", *Proc. of the 13th Systems Administration Conference USENIX LISA'99*, Seattle, Washington, USA, 1999, pp. 229–238.

[35] V. Vapnik, *The nature of statistical learning theory*, Springer Verlag, 1999.

[36] V. Vapnik, *Statistical Learning Theory*, Wiley-Interscience, 1998.

[37] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. Smola and R. Williamson, "Estimating the support of a high-dimensional distribution", in *Neural Computation*, vol. 13, no. 7, pp. 1443-1471, 2001.

[38] VirusTotal, *Public API v2.0*, VirusTotal Community, https://www.virustotal.com/en/documentation/public-api/.

[39] VX Heavens, *VX Heavens Site*, http://vxheaven.org/.

[40] R. Kaur and M. Singh, "Automatic Evaluation and

Signature Generation Technique for Thwarting Zero-Day Attacks", in *Recent Trends in Computer Networks and Distributed Systems Security*, CCIS, vol. 420, pp. 298-309, March 2014.

[41] R. Kaur and M. Singh, "Two-Level Automated Approach for Defending Against Obfuscated Zero-Day Attacks", in *Risks and Security of Internet and Systems*, LNCS, vol. 8924, pp 164-179, April 2015.

Authors' Profiles

Ratinder Kaur is a PhD scholar at Thapar University carrying out her research in the field of Network Security. She holds strong academic record. She received her Bachelor's Degree from Punjab Technical University and holds a Master's Degree, with honors in Software Engineering from Thapar University. She showcases strong inclination towards Computer Security field which is evident from her master thesis on Operating System fingerprinting, for which she won TCS (Tata Consultancy Services) Best Student Project Award, and now exploring Zero-day attack frontiers. Email: ratinder.kaur@thapar.edu.

Maninder Singh received his Bachelor's Degree from Pune University in 1994, and holds a Master's Degree, with honors in Software Engineering from Thapar Institute of Engineering & Technology, as well as a Doctoral Degree specialization in Network Security from Thapar University. He is currently working as Associate Professor in Computer Science and Engineering Department at Thapar University. Dr. Singh is on the Roll-of-honour at EC-Council USA, being certified as Ethical Hacker (C|EH), Security Analyst (ECSA) and Licensed Penetration Tester (LPT). Dr. Singh has successfully completed many consultancy projects for renowned national bank(s). His research interests include network security and grid computing, and he is a torchbearer for the open source community. He can be reached at, msingh@thapar.edu.

How to cite this paper: Ratinder Kaur, Maninder Singh,"A Hybrid Real-time Zero-day Attack Detection and Analysis System", IJCNIS, vol.7, no.9, pp.19-31, 2015.DOI: 10.5815/ijcnis.2015.09.03