Two-Level Automated Approach for Defending Against Obfuscated Zero-Day Attacks

Ratinder Kaur(✉) and Maninder Singh

Computer Science and Engineering Department, Thapar University,
Patiala 147004, India
{ratinder.kaur,msingh}@thapar.edu

Abstract. A zero-day attack is one that exploits a vulnerability for which no patch is readily available and the developer or vendor may or may not be aware. They are very expensive and powerful attack tools to defend against. Since the vulnerability is not known in advance, there is no reliable way to guard against zero-day attacks before they happen. 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. This paper presents a novel combination of anomaly, behavior and signature based techniques for detecting such zero-day attacks. The proposed approach detects obfuscated zero-day attacks with two-level evaluation, generates a new signature automatically and updates other sensors by using push technology via global hotfix feature.

Keywords: Zero-day attacks · Unknown attacks · Obfuscation · Signature generation · Push technology

1 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 2013 [2] there is 42 % increase in targeted attacks in 2012, 31 % of all targeted attacks aimed at businesses and 14 zero-day vulnerabilities were discovered. Another security threat report by Sophos [20] 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 look terrible and threatening.

A zero-day attack occur during the vulnerability window that exists in the time between when vulnerability is first exploited and when software developers start to develop a counter to that threat. It is difficult to measure the length of the vulnerability window, as attackers do not announce when the vulnerability was first discovered. Even developers may not want to distribute data for commercial or security reasons or they may not know if the vulnerability is being exploited when they fix it. So the vulnerability may not be recorded as a zero-day attack. The vulnerability window however, can be of several years long. According to an empirical study [1,12], a typical zero-day attack may last for 312 days on average and, after vulnerabilities are disclosed publicly, the volume of attacks exploiting them increases by up to 5 orders of magnitude.

In this paper a two-level automated approach for detecting zero-day attacks is proposed. This paper is an extension of our previous work with more detailed and optimized methodology [21]. It 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 and at last generates new signatures automatically to update other IDS/IPS sensors via global hotfix. The contribution of this paper over our previous approach is three folds: (1) We have optimized our previous algorithms to efficiently extract zero-day attack candidate and to update other IDS/IPS sensors automatically. (2) We have observed an increase of 10 % in detection rate and 1 % decrease in false alarm rate. (3) We have evaluated our system with large datasets of real attacks from various malware repositories.

The remainder of the paper is organized as follows. In Sect. 2, related work is summarized. In Sect. 3, detailed working of the proposed technique is presented. Finally in Sect. 4, experimental evaluation is described with results and paper is concluded.

2 Related Work

To defend against zero-day attacks, the research community has proposed various techniques. These techniques are classified into: statistical-based, signature-based, behavior-based and other techniques [22].

2.1 Statistical-Based

– Supervised Learning [14] 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 Nave 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. This system proves to be better than similar signature-free techniques that detect polymorphic malware and unknown malware based on analysis of Windows APIs.

- Contextual Anomaly Detection [15,18] 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.
- Combined Supervised and Unsupervised Learning [17] 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. A tree-based feature transformation is also introduced to overcome data imperfection issues and to detect the malware classes effectively.

2.2 Signature-Based

- SweetBait [6] 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 [23] automatically generate signatures for polymorphic worms, Lisabeth uses invariant byte analysis of traffic content, as originally proposed in Polygraph [5] and refined by Hamsa [11]. 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. Lisbeth and Hamsa systems are equally sensitive to the suspicious flows pool size but Lisabeth is lesser sensible to innocuous flow pool size than Hamsa. Lisabeth has shown significant improvement over Polygraph and Hamsa in terms of efficiency and noise-tolerance.
- In Honeycyber [3] 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 [4].
- ZASMIN [19] a Zero-day Attack Signature Management Infrastructure is an early detection system for novel network attack detection. This system provides early detection function and validation of attack at the moment the attacks start to spread on the network. To detect unknown network attacks, the system adopted new 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 [7] 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.

2.3 Behavior-Based

– Network-Level Emulation [8,13] 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.
– SGNET [9] 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.

2.4 Other Hybrid Techniques

– Hybrid Detection for Zero-day Polymorphic Shellcodes (HDPS) [10] 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. This method filters normal packets with accuracy and low overload. But this approach cannot block shellcodes in network packets and it is hard to obtain transition matrixes of Markov Model.
– Honeyfarm [16] 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 are used to track attacker activities while high interaction honeypots help in analyzing new attacks and vulnerabilities. The controller is responsible to redirect suspicious traffic to respective honeypots which are deployed in honeyfarm.

2.5 Limitations of Existing Techniques

The following limitations of recent studies have been the prime motivation for our research.

- Statistical-based detection techniques cannot be used for instant detection and protection in real time. They are dependent on static attack profiles and requires 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.

3 Proposed Technique

3.1 Architecture

An efficient and novel technique integrating the three main detection techniques (Anomaly, Behavior and Signature based) is proposed to minimize the impact of above identified challenges during zero-day attack detection. It does two-level evaluation to detect and confirm zero-day attack. Figure 1 shows the basic architecture of our proposed approach. It comprises of different components: Router, Port Mirroring Switch, Honeynet, Intrusion Detection and Prevention (IDS/IPS) Sensors, Zero-day Attack Detection (ZAD) System and Global IDS/IPS Hotfix Server. The router connects the entire setup to the Internet. Port mirroring switch passes network traffic simultaneously to both Honeynet and IDS/IPS sensors. Firstly, the network traffic is captured and filtered for known attacks. If the filtered traffic is found suspicious of containing some unknown attack it is evaluated for zero-day attack in the ZAD system and a new signature is generated and updated. Otherwise, if the traffic trace is found benign, whitelist in IDS/IPS sensors is updated.

Honeypots have been found to be effective against zero day threats therefore, Honeynet is used to identify the mechanism of a new attack and to collect evidence for attacker's activity. When a known attack hits Honeynet it is blocked and logged. When a new attack is encountered the network traffic associated with that attack is logged and is redirected to the high-interaction honeypots. The honeypots interact with the attacker and the entire communication is logged. The network logs and honeypot system interaction logs collectively addressed as "Honeynet Trace" or "Unknown Attack Trace" are kept for further analysis. At the same time the IDS/IPS sensor filters known attacks for the same traffic and stores rest of the filtered traffic in an online repository. Then the data

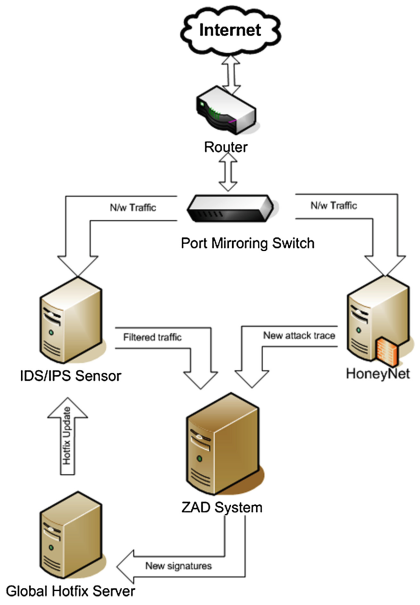

Fig. 1. Basic architecture of proposed approach

collected from both Honeynet and IDS/IPS sensor is compared and analyzed in ZAD. The ZAD system examines if similar unknown attack traces are found in IDS/IPS sensor's filtered traffic or not. If similar attack traces are found, then that is a candidate for zero-day attack undetected by an IDS/IPS sensor. Up to this level, this is assured that there is some malicious traffic which was missed by sensors. This could only happen when the IDS/IPS sensor does not have matching signature for the unknown malicious traffic in its database. After finding the candidate for zero-day attack it is necessary to do further analysis to confirm its malicious intent and to generate a new signature for it.

3.2 Evaluating Zero-Day Attack

The candidate for zero-day attack may result in false positive so it's essential to evaluate it. The evaluation process is used to confirm the malicious intentions of the candidate by analyzing system anomalies in which it is executed. This evaluation is done by ZAD-Analyzer in the ZAD system. Figure 2 depicts the internal process flow of ZAD-Analyzer. More details on each component is discussed in the following sections.

Compare and Extract Unit (CEU): CEU takes input from both Honeynet and IDS/IPS sensors to compare and extract the zero-day attack candidate. For comparison, it uses Rabin-Karp algorithm for string matching [24]. Rabin-Karp algorithm is an easy solution for string matching with linear complexity.

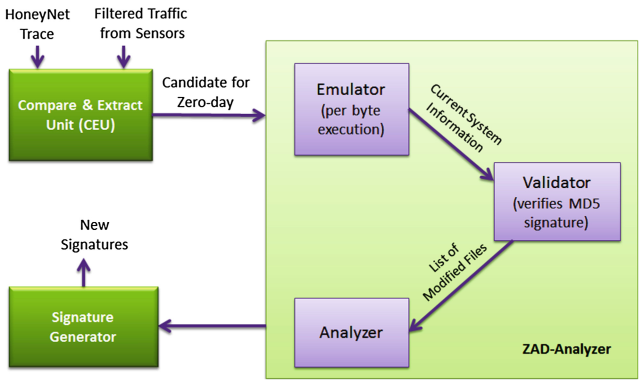
HoneyNet Trace
Filtered Traffic from Sensors
Compare & Extract Unit (CEU)
Candidate for Zero-day
Emulator (per byte execution)
Current System Information
Validator (verifies MD5 signature)
List of Modified Files
Analyzer
ZAD-Analyzer
New Signatures
Signature Generator

Fig. 2. ZAD-analyzer internal process flow [26]

Consider a new attack pattern captured by Honeynet, *Honeynet-Trace* (HT) of length m and filtered traffic stored by sensors, *Filtered-Traffic* (FT) of length n, where $m << n$. The general principle is that for every m byte of FT, the new hash value is calculated and this hash value is compared with the hash value of HT. For efficient string matching, the new hash value is computed by using the old hash value, the current byte of FT, and the byte of FT seen m byte before. The XOR operation $\oplus$ is a suitable function for this purpose. The hash value is calculated as:

$$hash \leftarrow hash \oplus FT[curpos] \oplus FT[curpos - m]$$

where $curpos$ is the current position within the FT, and m is the length of HT to be searched. Using packet bytes directly leads to false positives. Therefore, the current byte is used as an index into a table containing randomly generated 32-bit values. The XOR is then calculated using the derived 32-bit values. Just XORing the 32-bit values is not sufficient to reduce false positives. Thus, along with XOR, Shift operation is also applied to 32-bit values and old hash values. To find all occurrences of HT in FT an Algorithm 1 is implemented.

Emulator: Figure 3 depicts the working of an emulator. The zero-day attack candidate (attack trace) is input to an emulator for per byte execution. The emulator is the right choice for analyzing decrypted and obfuscated code. Any type of obfuscated code is allowed to execute in its original form. The idea here is to let the code decrypt itself in the memory and do harm to the emulated system. After execution, the interesting part is to log all the changes made to the file system and registry.

The emulator executes attack trace by successively reading its instructions and performing equivalent operations in the emulated environment. After execution of the attack trace the system anomalies are analyzed. The system anomalies help to determine a system's status (whether or not malicious code is present) by comparing the system status information to a standard. For this, the abstract

Algorithm 1. Compare and Extract Algorithm

```
1: procedure $main()$
2:     Initialize DB drivers
3:     Start secure communication with database
4:     for iterate over $FT$ do
5:         Fetch String $FT[1..n]$ = get...string
6:         for iterate over $HT$ do
7:             Fetch String $HT[1..m]$ = get...honeynet string
8:             invoke $RabinKarp(FT[1..n], HT[1..m])$
9:         end for
10:    end for
11: end procedure
12: procedure RabinKarp($stringFT[1..n], stringHT[1..m]$)
13:    for $i$ from 1 to $n - m + 1$ do
14:        if $hFT = hHT$ then
15:            if $FT[i..i + m - 1] = HT$ then
16:                return New attack pattern is found in the filtered traffic at: $i$
17:            end if
18:        end if
19:        $hFT := hash(hFT[i + 1..i + m])$
20:    end for
21:    return No similar attack traces are found.
22: end procedure
```

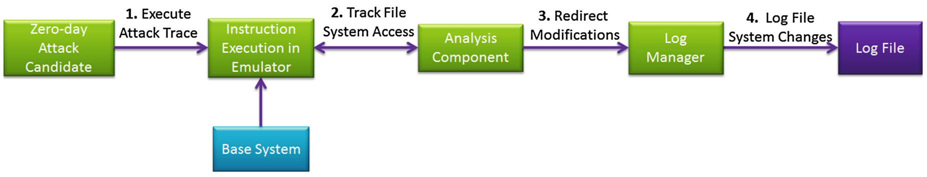
Zero-day Attack Candidate
1. Execute Attack Trace
Instruction Execution in Emulator
2. Track File System Access
Analysis Component
3. Redirect Modifications
Log Manager
4. Log File System Changes
Log File
Base System

Fig. 3. Working of emulator

method of analyzing system anomalies is used that is validating checksums of critical files. The most critical files include the registry files (in Window's) and the file system (in both Window's and Unix). The file system is a vital storage component and any anomalous executions intended to damage it will likely be detected by monitoring the changes that attempt to alter or damage the file system. Our proposed approach may not be able to detect attacks that alters only runtime memory, while the majority of attacks which do result in changes to the file system will leave a proof of an malicious event. Thus, our analysis is based on the fact that it is not possible to compromise a system without altering a system file. A malicious code can only do one of three things: add, remove or modify files. It can remove system logs. It can add tools such sniffers or viruses for later use. And most important it can change the system in numerous ways like new accounts, modified passwords, tweaked registries, trojaned files etc. During execution of attack trace, the analysis component analyzes file system access in a stealthy manner and redirects all the modifications transparently to a log manager. The basic idea behind this is to keep the original base system image clean so that no reboot is required after every malicious code execution. The log manager generates a log file containing information about the file system changes during the execution of attack trace. This log file is then given to validator for further verification.

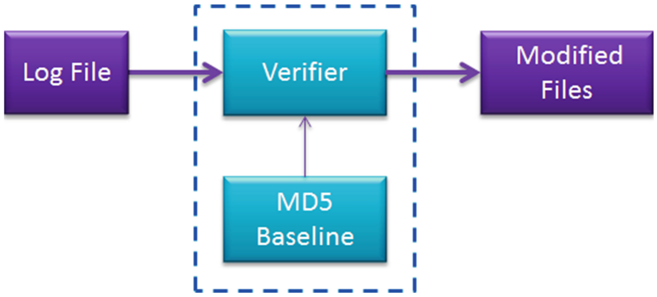

Fig. 4. Validator

Validator: In Fig. 4 the validator compares a log file to a MD5-baseline and results in list of modified files. The validator maintains a MD5-checksum database of the original base system files. After execution of an attack trace the file system gets corrupted and a log file containing file system changes is generated which is then sent to the validator. A critical component known as verifier, accurately recalculates MD5-checksum on logged files and compares them with the MD5-baseline. The MD5 algorithm takes an input of arbitrary length and produces a fixed-length fingerprint, hash, or checksum. As it is computationally infeasible to produce same fingerprint therefore, the MD5-checksum provides a mechanism to verify data integrity. So, when the data within a file is changed, its hash will also change. Such changes to the file system can be categorized into four cases as in Table 1. After comparing with the baseline a "List of Modified Files" is created for the analyzer.

Table 1. Enumeration of possible cases

File exists in:		Interpreted action
Log file	MD5-database	
X		Created
	X	Deleted
X	X'	Altered/Updated
X	X	Read/Accessed

Analyzer: The analyzer receives a "List of Modified Files" from the Validator. It then crosschecks the "List of Modified Files" with the "List of Critical Files" maintained. Critical files for e.g. in Windows can be registry files, startup files, system configuration files, system libraries, system binaries, password files, etc. In case of Linux, critical files are in directories like: /bin, /boot, /etc, /root, /sbin, /tmp, /usr/bin, /usr/etc, /usr/sbin, /var/log, /var/run, /var/spool, /var/tmp. If the critical files are modified, it proves that the candidate is a real zero-day attack. Thus, the system does two-level evaluation for detecting a zero-day attack. First-level (*Detects UnKnown*) where Honeynet flags a new suspicious event and IDS/IPS sensors ignores it. Second-level (*Confirms Malicious*) where MD5 baseline is used to confirm its malicious intentions. This two-level (*Detects Unknown Malicious*) evaluation decreases the false positives to nearly zero. After confirming a zero-day attack, ZAD-Analyzer commands the Signature Generator to generate a signature for the new attack. On the other hand, if no critical file is changed then the candidate is false positive and the Whitelist is updated.

3.3 Signature Generation (SG) and Hotfix Update

After evaluation zero-day attack packets are fed to the next module for signature generation. This module generates a common token-subsequence signature for a set of attack packets by applying the Longest Common Subsequence (LCSeq) algorithm. The algorithm compares two zero-day attack packets to get the longest common subsequence between them. Let two sequences be defined as follows: $X = (x_1, x_2 ... x_m)$ and $Y = (y_1, y_2 ... y_n)$. Let $LCSeq(X_i, Y_j)$ represent the set of longest common subsequence of prefixes X_i and Y_j. This set of sequences is given by the following.

$$LCSeq(X_i, Y_j) = \begin{cases} \Phi & \text{if } i = 0 \text{ or } j = 0 \\ LCSeq(X_{i-1}, Y_{j-1}) + 1 & \text{if } x_i = y_j \\ longest(LCSeq(X_i, Y_{j-1}), LCSeq(X_{i-1}, Y_j)) & \text{if } x_i \neq y_j \end{cases}$$

After the new attack signatures are generated by ZAD, they are sent to a server responsible for global IDS/IPS hotfix update. This hotfix signature update approach is quick and proactive which is necessary for containing zero-day attacks at the right time. Moreover, the hotfix can be applied to other sensors without stopping or restarting their service. The Global Hotfix Server uses push technology to initiate the transaction. The client sensors have to subscribe to the hotfix server for receiving updates. The hotfix server provides live-update whenever a new signature is generated. It collects the new signature in a file and sends out to the client sensors. The signature file is sent over HTTPS to client sensors. When a client sensor receives signature file, it calculates MD5 checksum. The result of the checksum is sent to the hotfix server. If the checksum doesn't match, the client discards the download and the server in response sends the same signature file again. In case, the update fails due to any network or installation error, the hotfix server retries to update client sensor for a given number of retries and exceeding the limit assumes that client is down and disables it. The

Algorithm 2. Hotfix Update Algorithm

```
 1: procedure server()
 2:     set MAX_RETRY_COUNT = 5
 3:     Initialize DB drivers
 4:     Start secure communication with database
 5:     for iterate over $PUSH$ table do
 6:         set String $update$ = get...update to be pushed
 7:         set $current\_update\_date$ = get...current update date
 8:         for iterate over $CLIENTS$ table do
 9:             set String $clientInfo$ = get...string using IP and credentials
10:             set String $last\_update\_date$ = get...the last update date
11:             if $last\_update\_date < current\_update\_date$ then
12:                 Push update to $clientInfo$
13:                 invoke $client$
14:                 if SUCCESS then
15:                     update $CLIENTS$ table.
16:                     Set $last\_update\_date = current\_update\_date$
17:                 else
18:                     Raise Alert "Update Failed".
19:                     Resend update
20:                     set $RETRY\_COUNT = RETRY\_COUNT + 1$
21:                     if $RETRY\_COUNT > MAX\_RETRY\_COUNT$ then
22:                         Disable Client node.
23:                     end if
24:                 end if
25:             end if
26:         end for
27:     end for
28: end procedure
29: procedure client()
30:     Receive update from server
31:     String $new\_md5sum$ = generate md5sum of update
32:     if $new\_md5sum = original\_md5sum$ then
33:         update signature database and send SUCCESS to server
34:     else
35:         discard update and send DECLINE to server
36:     end if
37: end procedure
```

complete process is automatic that doesn't require and manual intervention. The best part of global update is that all the sensors remain updated and are in sync always. Algorithm 2 is optimized and depicts this scenario where new signatures are pushed to the various sensors. The Hotfix Update algorithm is optimized to decrease the delay between signature generation and update as a short update period leads to fast reaction time against new attacks. On evaluation it was observed that the optimized algorithm took less time to update IDS/IPS sensors as compared to our previous approach.

4 Experimental Results

All experiments run on an isolated network in the research lab. Honeynet comprises of Honeywall Roo-1.4 and high-interaction honeypots with the Linux Sebek client installed on them. For IDS/IPS sensors SNORT is used. We have also developed a prototype for ZAD System with Signature Generator for our experiment. It is implemented in Java using Eclipse as an IDE and Mysql as a database. Four standard metrics were used to evaluate the performance of our technique: True Positive Rate (TPR), False Positive Rate (FPR), Total Accuracy (ACC) and Receiver Operating Characteristic (ROC) curve. TPR is the percentage of correctly identified malicious code shown in Eq. 1. FPR is the percentage of wrongly identified benign code (Eq. 1). ACC is the percentage of absolutely correctly identified code, either positive or negative, divided by the entire number of instances as shown in Eq. 2. 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. In the equations below, True Negative (TN) is the number of correctly identified benign code and False Negative (FN) is the number of wrongly identified malicious code.

$$TPR = \frac{|TP|}{|TP| + |FN|}; \quad FPR = \frac{|FP|}{|FP| + |TN|} \tag{1}$$

$$ACC = \frac{|TP| + |TN|}{|TP| + |FP| + |TN| + |FN|} \tag{2}$$

The dataset comprises of 54,502 samples in total consisting of 40,112 malware samples (both obfuscated &non-obfuscated) and 14,390 benign samples. The dataset with obfuscated and unknown malware have been collected from various sources like Honeynet project, VX heavens [25] and other online malware repositories. The benign samples include: application software, system software, and many other user applications. The distribution of malware samples is represented in Table 2.

Table 2. Distribution of malware samples

Malware type	No. of samples	Not-obfuscated	Obfuscated
Virus	13,509	3,053	10,456
Worm	10,150	2,741	7,409
Rootkit	257	130	127
Backdoor	4,688	1,876	2,812
Exploit	1,206	262	944
Trojan	10,302	2,782	7,520

To compute the accuracy of the proposed approach both benign and malware samples were redirected towards Honeynet and IDS/IPS sensors simultaneously. Table 3 represents the recorded values of TPR, FPR, ACC and ROC for obfuscated and non-obfuscated zero-day malware.

Table 3. System detection accuracy

Malware type	Not-obfuscated				Obfuscated			
	TPR	FPR	ACC	ROC	TPR	FPR	ACC	ROC
Virus	0.993	0.021	0.992	0.982	0.987	0.022	0.99	0.98
Worm	0.996	0.018	0.986	0.991	0.976	0.0301	0.972	0.969
Rootkit	0.983	0.0233	0.971	0.981	0.967	0.032	0.961	0.957
Backdoor	0.972	0.0281	0.975	0.975	0971	0.031	0.972	0.970
Exploit	0.984	0.025	0.973	0.985	0.968	0.0323	0.972	0.972
Trojan	0.965	0.031	0.955	0.958	0.891	0.0331	0.903	0.893

Experiments were also conducted to measure the performance of each ZAD component. For each ZAD component, their average execution time was recorded under various attacks. The evaluation was performed on a system with a processor core i7, and 8GB of RAM. All components executed quickly to perform desired analysis. Figure 5 shows the experimental results. From the Fig. 5, it is clear that CEU took more time as it has to compare attack trace with entire online repository where filtered traffic is stored. The emulator takes slightly less time than the CEU to execute an attack trace, track file system accesses and send reports to the validator. On the other hand, the validator and analyzer took approx. similar time for comparison with a set baseline of md5-checksum and critical files respectively. However, the signature generation component requires more time to generate new signatures than to push hotfix updates to IDS/IPS sensors.

In another experiment, to check and verify hotfix updates, various unknown obfuscated attacks were launched and signature updates were observed. The experiment was conducted for 7 days and hotfix updates were recorded along the days. Figure 6 shows the actual updates, successful updates and failed updates for a week. Day 1, the experiment was started and total 10 updates were processed from which 8 were successful and 2 were declined by the client IDS/IPS sensors. The clients can decline the update if the update file is corrupted and the server couldn't resend the same update again. Day 2, there were 5 updates with no refused cases. On an average, minimum 2 updates were rejected a day and in worst case we recorded 3 rejections. From the results it is proved that new signatures were generated and updated efficiently with minimum misses to contain the zero-day attack in future.

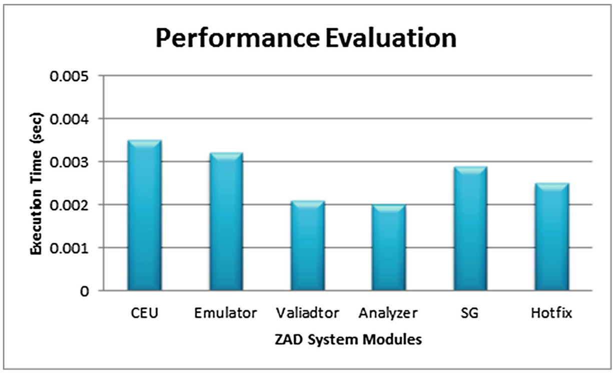

Fig. 5. Performance evaluation of ZAD components

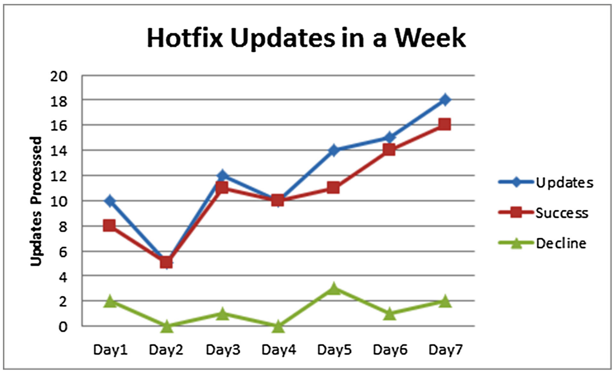

Fig. 6. Hotfix updates in a week

5 Conclusions

In this paper a two-level automated approach is proposed for detecting obfuscated zero-day attacks. This paper extends our previous work with more detailed and optimized methodology. It addresses the research problems with existing approaches and tries to provide a solution to the whole problem. The proposed approach provides an online detection mechanism against obfuscated zero-day attacks with automatic evaluation at two levels and automatically generating signatures with optimized global hotfix update. Experiments were conducted on real obfuscated zero-day malware, collected from various online malware repositories. The results were very promising achieving the best detection rate of nearly 99 % with 0.021 false positive rate and in the worst case, detection rate was 89 % with 0.033 false positive rate. Other results also showed that new signatures were generated and updated efficiently with least declines to contain the zero-day attack. The future work includes: (1) defining a system baseline to gather more information from other file system objects and attributes rather than from just one attribute i.e. md5-checksum. This detailed information can further help to categorize and provide more insight about the behavior of a zero-day malware. (2) To construct reliable signatures for obfuscated and polymorphic attacks. (3) To consider issues regarding anti-emulation techniques.

References

1. Bilge, L., Dumitras, T.: Before we knew it: an empirical study of zero-day attacks in the real world. In: Proceedings of ACM Conference on Computer and Communications Security, pp. 833–844. ACM Press, New York (2012)
2. Symantec's Internet Threat Report of 2013. https://scm.symantec.com/resources/istr18_en.pdf
3. Mohammed, M.M.Z.E., Chan, H.A., Ventura, N.: Honeycyber: automated signature generation for zero-day polymorphic worms. In: Proceedings of the IEEE Military Communications Conference (MILCOM 2008), pp. 1–6. IEEE Computer Society, Washington (2008)
4. Mohammed, M.M.Z.E., Chan, H.A., Ventura, N., Hashim, M., Amin, I., Bashier, E.: Detection of zero-day polymorphic worms using principal component analysis. In: Proceedings of the 6th IEEE International Conference on Networking and Services, pp. 277–281. IEEE Computer Society, Washington (2010)
5. Newsome, J., Karp, B., Song, D.: Polygraph: automatically generating signatures for polymorphic worms. In: Proceedings of the IEEE Symposium on Security and Privacy, pp. 226–241. IEEE Press, New York (2005)
6. Portokalidis, G., Bos, H.: SweetBait: zero-hour worm detection and containment using low-and high-interaction honeypots. J. Comput. Telecommun. Netw. **51**(5), 1256–1274 (2007)
7. Wang, L., Li, Z., Chen, Y., Fu, Z., Li, X.: Thwarting zero-day polymorphic worms with network-level length-based signature generation. J. IEEE/ACM Trans. Netw. **18**(1), 53–66 (2010)
8. Polychronakis, M., Anagnostakis, K.G., Markatos, E.P.: Network-level polymorphic shellcode detection using emulation. J. Comput. Virol. **2**(4), 257–274 (2006)
9. Leita, C., Dacier, M.: SGNET: A Distributed Infrastructure to Handle Zero-day Exploits. Research report, EURECOM institute (2007)
10. Ting, C., Xiaosong, Z., Zhi, L.: A hybrid detection approach for zero-day polymorphic shellcodes. In: International Conference on E-Business and Information System Security, pp. 1–5. IEEE, Wuhan (2009)
11. Li, Z., Sanghi, M., Chen, Y., Kao M.Y., Chavez, B.: Hamsa: fast signature generation for zero-day polymorphic worms with provable attack resilience. In: Symposium on Security and Privacy, pp. 15–47. IEEE, Oakland (2006)
12. A 0-Day Attack Lasts On Average 10 Months. http://hackmageddon.com/2012/10/19/a-0-day-attack-lasts-on-average-10-months/
13. Polychronakis, M., Anagnostakis, K.G., Markatos, E.P.: Emulation-based detection of non-self-contained polymorphic shellcode. In: Kruegel, C., Lippmann, R., Clark, A. (eds.) RAID 2007. LNCS, vol. 4637, pp. 87–106. Springer, Heidelberg (2007)
14. Alazab, M., Venkatraman, S., Watters, P., Alazab, M.: Zero-day malware detection based on supervised learning algorithms of api call signatures. In: Proceedings of the 9th IEEE Australasian Data Mining Conference (AusDM 2011), Australia, pp. 171–182 (2011)

15. Aleroud, A., Karabtis G.: A contextual anomaly detection approach to discover zero-day attacks. In: IEEE International Conference on Cyber Security (CYBER-SECURITY 2012), pp. 40–15, Washington (2012)
16. Jain, P., Sardana, A., Defending against internet worms using honeyfarm. In: CUBE International Information Technology Conference (CUBE 2012), Pune, India, pp. 795–800. ACM Press, New York (2012)
17. Comar, P.M., Liu, L., Saha, S., Tan, P.N., Nucci A.: Combining supervised and unsupervised learning for zero-day malware detection. In: Proceedings of INFO-COM, pp. 2022–2030. IEEE Press, Turin (2013)
18. Aleroud, A., Karabatis G.: Toward zero-day attack identification using linear data transformation techniques. In: Proceedings of the 7th IEEE International Conference on Software Security and Reliability (SERE 2013), pp. 159–168. IEEE Press, MD (2013)
19. Kim, I., et al.: A case study of unknown attack detection against zero-day worm in the honeynet environment. In: Proceedings of the 11th IEEE International Conference on Advanced Communication Technology (ICACT 2009), pp. 1715–1720. IEEE Press, Ireland (2009)
20. Sophos Security Threat Report of 2014. http://www.sophos.com/en-us/media library/PDFs/other/sophos-security-threat-report-2014.pdf
21. Kaur, R., Singh, M.: Automatic evaluation and signature generation technique for thwarting zero-day attacks. In: Martínez Pérez, G., Thampi, S.M., Ko, R., Shu, L. (eds.) SNDS 2014. CCIS, vol. 420, pp. 298–309. Springer, Heidelberg (2014)
22. Kaur, R., Singh, M.: A survey on zero-day polymorphic worm detection techniques. J. IEEE Commun. Surv. Tutorials **99**, 1–30 (2014)
23. Cavallaro, L., Lanzi, A., Mayer, L., Monga, M.: Lisabeth: automated content-based signature generator for zero-day polymorphic worms. In: Proceedings of the 4th ACM International Workshop on Software Engineering for Secure Systems, pp. 41–48. ACM Press, Germany (2008)
24. Karp, R.M., Rabin, M.O.: Efficient randomized pattern-matching algorithms. J IBM J. Res. Dev. **31**(2), 249–260 (1987)
25. VX Heavens, VX Heavens Site. http://vxheaven.org/