13th IEEE International Conference on Dependable, Autonomic and Secure Computing, Oct 2015, Liverpool, UK © IEEE

CloudZombie: Launching and Detecting Slow-Read Distributed Denial of Service Attacks from the Cloud

Saeed Shafieian
School of Computing
Queen's University, Kingston, Canada
saeed@cs.queensu.ca

Mohammad Zulkernine
School of Computing
Queen's University, Kingston, Canada
mzulker@cs.queensu.ca

Anwar Haque
Bell Canada
Hamilton, Canada
anwar.haque@bell.ca

***Abstract*—As the Cloud is becoming more ubiquitous and less expensive to utilize, a new class of denial of service attacks is emerging. These attacks employ the Cloud to launch denial of service attacks against a target outside the Cloud. Slow-read denial of service can be one of those attacks. It is a new type of application-layer denial of service attacks that exploits vulnerabilities in the HTTP protocol in order to make services inaccessible for legitimate users on a target machine. This attack is difficult to detect by conventional intrusion detection systems, as it generates legitimate and complete packets in all networking layers and in a slow rate. The attack exhausts the target's resources such as Web server connection pool and generally needs much less bandwidth compared to traditional volumetric attacks. The Cloud is an ideal platform to launch slow-read attack, since virtual machines on the Cloud can be easily exploited as a botnet for the purpose of this attack. We show how this new phenomenon, CloudZombie, can happen by remotely launching slow-read attacks from the Cloud. We also present a new approach to detect slow-read attacks. Our method uses Random Forests to build classifiers based on which the incoming slow-read traffic can be detected at the destination. High performance and low error rates of our approach indicate its efficiency to detect the attack.**

***Keywords*—Denial of Service, Slow Read, Cloud, Random Forests.**

I. Introduction

Cloud is becoming more ubiquitous and also less expensive to utilize. Currently, almost all major companies in IT industry offer some type of Cloud services to their clients. There are, however, many security threats against the Cloud including distributed denial of service (DDoS) attacks [1]. In this work, we are interested in the opposite scenario; the potential threat that the Cloud brings against non-Cloud targets.

Being one of the oldest and most effective attacks on the Internet, a denial of service attack aims to make a victim's computing resources inaccessible to its legitimate users. Traditional denial of service attacks use flooding techniques to send a huge number of malicious requests to the victim machine [2]. Since processing, memory, networking and other hardware power and capacity is limited on any single machine, the target may finally become unable to serve the legitimate requests.

Application-layer denial of service attacks (ADDoS) target vulnerabilities in application-layer protocols such as HTTP in order to exhaust server resources (socket, CPU, memory, *etc.*). Unlike traditional DDoS attacks that send flooding network- or transport-layer packets towards the victim machine and may be easily detected, ADDoS attacks exploit vulnerabilities in application-layer protocols and require fewer number of connections. In general, there are three major differences between application-layer (layer 7) and infrastructure-layer (layers 3 and 4) DDoS attacks. First, application-layer attacks require much less resources on the attacker side in terms of the number of bots and bandwidth, because they normally exploit vulnerabilities. Second, they are more difficult to detect, since they establish legitimate TCP and UDP connections. Finally, more hardware power on the target may not prevent these attacks, because they target the application's limitations rather than the host's computing or networking limitations.

Slow-read attack is a new type of ADDoS attacks that reads the HTTP responses slowly; hence slow-read. This attack exhausts server resources such as connection pool. Most modern Web servers close a connection when there is no data transmission between the client and the server. However, if there is a flow of data, even minimal, most likely the connection will be kept open and this is exploited by this attack. Due to the nature of this attack, it does not require a botnet with tens of thousands of compromised machines. An attacker does not need huge bandwidth either. These would make this attack easier to launch and harder to detect as opposed to volumetric attacks that require tremendous amount of resources on the attacker side.

In this paper, we show how virtual machines on the Cloud can be exploited to launch successful slow-read denial of service attacks against a powerful target server outside the Cloud. We discuss CloudZombie as a new phenomenon to remotely launch denial of service attacks "from" the Cloud. This is different from many other work that discuss denial of service "against" the Cloud. In other words, the Cloud is the source of the attack in our work as opposed to being the target. The slow-read attack is generally able to make the target defunct within a few seconds. We then propose a novel solution based on models generated using random forests to detect such attacks. We show that our classifiers have high accuracy and low false positive and negative rates.

The rest of the paper is organized as follows. Section II explains the slow-read attack in detail. Section III discusses the related work and how our work is different. Section IV presents experiments showing how the slow-read can be launched from the Cloud. Section V discusses how to detect the slow-read attack using our proposed solution. Section VI presents and discusses the experimental results, and finally Section VII provides a summary of this work and its limitations.

HTTP Request	HTTP Response
http://www.example.com/image.png	HTTP/1.1 200 OK Date: Wed, 25 Mar 2015 15:48:40
GET /image.png HTTP/1.1 Host: www.example.com User-Agent: Mozilla/5.0 Accept: text/html,application/ xhtml+xml,application/xml	Server: Apache/2.2.22 (Ubuntu) Last-Modified: Tue, 10 Feb 2015 18:32:00 GMT Accept-Ranges: bytes Content-Length: 8890

Fig. 1. HTTP request for a file (image.png) on a server (www.example.com) and the successful (Ok) HTTP response from the server.

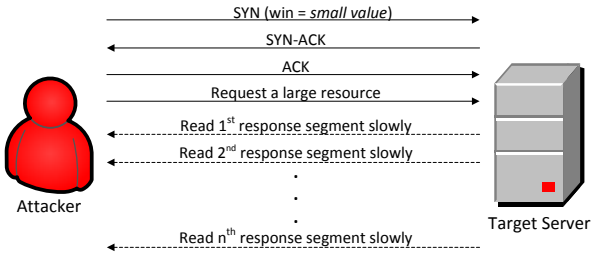

Fig. 2. Slow-read attack launched from a single source. A single HTTP request is shown.

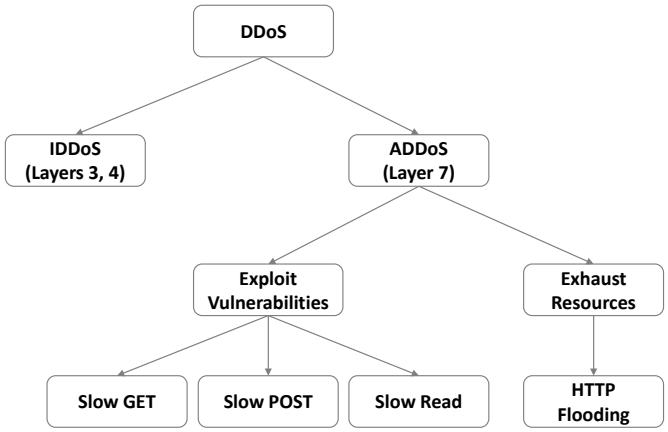

Fig. 3. High Level Classification of Application-layer DDoS (ADDoS) Attacks.

II. Slow-read Denial of Service Attacks

Slow-read is a new ADDoS attack invented by Sergey Shekyan [3]. To understand how the slow-read attack works, we first need to see how a normal HTTP connection works. Figure 1 shows a simplified HTTP request and response for a resource called image.png on a server with the address of www.example.com. In a normal scenario, the client sends an HTTP request for a resource on the server, image.png here. If the address is correct and the file is accessible, the server responds by a 200 Ok message and sends the requested resource. The HTTP protocol by design requires each connection to be completed in order to release its connection resources [4]. This requirement can be exploited by slow-rate DoS attacks to keep the server busy by sending complete requests to, but reading responses slowly from the server.

Figure 2 shows how slow-read works in a single malicious request. After the three-way handshake between the attacker and the target server, a TCP connection is established and each party is ready to send data to the other end. During the handshake, each party can advertise different TCP header values such as window size, window scaling, *etc*. By advertising a small window and requesting a large resource, the attacker forces the server to send data slowly. This figure shows the scenario for a single slow-read request initiated from a single machine. However, in a real case scenario, tens of thousands of such requests will be sent toward the target concurrently from different machines.

Figure 3 shows the position of the slow-read attack in a high level classification of ADDoS attacks. As the figure shows, DDoS attacks can be classified into Infrastructure-layer DDoS (IDDoS) and ADDoS attacks. ADDoS can be further classified into two types based on their behavior. Basically, an application-layer attack may exploit vulnerabilities in the application-layer protocols or applications, or exhaust resources on the victim machine [5]. Slow GET, Slow POST, and slow-read attacks all exploit vulnerabilities in the HTTP protocol. These attacks exploit the trustfulness of the HTTP protocol to the incoming connections. In other words, they exploit the fact that a Web server waits for the incoming connection to complete sending and receiving all the data which is required by the HTTP protocol by design.

The other class of ADDoS attacks only exhaust resources on the target. HTTP flooding attacks belong to this class. These attacks use different techniques and send flooding HTTP requests towards the victim. The attacker basically only needs to know the IP address of the victim and may requests random URLs based on that. There is no need to request a valid Web page or resource on the target server, since the attacker normally do not want to receive any responses and only aims to overwhelm the resources on the victim machine. These ADDoS attacks resemble flooding infrastructure-layer attacks, but send HTTP rather than TCP or UDP packets.

It should be noted that the first class of ADDoS attacks which exploit vulnerabilities, also apply flooding techniques. Any distributed denial of service attack has to use some types of flooding, otherwise it may not be considered a DDoS. However, some attacks such as HTTP flooding merely employ flooding techniques and do not normally try to exploit any specific vulnerabilities as well.

III. Related Work

The existing solutions for slow-read attack try to put a threshold on connections such as limiting the number of concurrent connections, connection lifetime, or TCP window size [3], [6]. The disadvantage of these solutions is that they can affect legitimate users, as well. Finding the right threshold may also be very tricky. Another approach [7] tries to address the slow-read attack by monitoring packet frequency. This approach is essentially a network-based method which is not tailored for this specific application-layer attack. As a result, it may result in high false positive and negative rates.

One of the techniques to mitigate the slow-read attack is to prevent a large number of connections from a single source IP address to be in the *busy* state on the server [6]. The problem of this approach is that the number of concurrent connections

from a single source IP address cannot be limited to a very small number, because several connections may be required to retrieve all the resources on a Web page. For example at the time of writing this paper, a total number of 161 GET and POST requests would be made by a client in order to retrieve the www.amazon.ca homepage (without HTTP pipelining). In versions prior to HTTP/1.1, 161 TCP connections were established to fetch all those resources. Furthermore, some legitimate users such as home Internet users, each with several devices nowadays, share a public IP address. This further impacts the number of legitimate connections from a single IP address that needs to be supported by a Web server.

The other suggested mitigation mechanisms include blocking connections that advertise abnormally small window sizes, disabling HTTP pipelining and persistent connections, and limiting the entire connection time [3]. In order for an attacker to launch a slow-read attack, the attacker needs to find a fairly large resource on the target server and set the receive window size just below that resource's size. In order to further evade detection, an attacker can randomly switch requesting among some files with fairly large sizes. Therefore, dropping connections merely based on small window sizes would not be an effective countermeasure if the attacker requests a large enough resource. Moreover, the attacker can send requests with different random window sizes to evade a detection mechanism detecting fixed small window sizes.

The other proposed mitigation technique is to disable HTTP pipelining and persistent connections. This method may mitigate the attack, but disabling those features would noticeably reduce the network performance for legitimate users. As a result, the impact of applying this technique may not be tolerable. The problem of limiting the entire connection time is that there may be legitimate users with low bandwidth. Moreover, in the case of an SSL connection which normally takes longer than a non-encrypted connection, limiting the entire connection time may be tricky and might prevent legitimate users to access the server.

H. Gonzalez *et al*. [4] discuss the impact of the application layer denial of service attacks. They provide a classification of application-layer attacks into six categories of request flooding, asymmetric, hybrid, exploit-based, low-rate and slow-rate. They test different attacks on different Web servers, and conclude that modern Web servers are resilient to these types of attacks. However, based on our experiments, Apache Web server which is currently the mostly used Web server on the Internet [8] is not resilient to the slow-read attack. Moreover, they do not offer any new countermeasures for the attack.

E. Cambiaso *et al*. [9] give a classification of slow-rate DoS attacks on Web applications. They categorize slow-rate attacks as pending-request, long-response, multi-layer or mixed attacks. They do not, however, provide any detection or mitigation techniques for the attacks.

J. Park *et al*. present an analysis of the slow-read attack on Apache Web server [10]. They discuss the relationship between the server's timeout, maximum number of connections, and the total number of connections made with the server as to how they may lead to a successful or unsuccessful slow-read attack. They also discuss an effective way of launching a distributed version of the attack, and discuss that a distributed slow-read attack can be more challenging to mitigate. They, however, do not propose any new solution to detect or mitigate the attack.

TABLE I. CURRENT SLOW-READ ATTACK COUNTERMEASURES

Solutions	**Disadvantages**
Blocking connections with very small window sizes	Not protecting against requests with bigger window sizes that are still below the target file size, difficult to find the right boundary
Disabling persistent connections and HTTP pipelining	Reducing the network performance for legitimate users
Limiting connection lifetime	Preventing legitimate users with low bandwidth, difficult to find the right limit
Limiting the number of requests from a single IP	Not protecting against distributed attacks
Monitoring the average rate of packets received by the server	Having high false negative for slow-read and high false positive rates for flash crowds

M. Aiello *et al*. [7] propose a solution for slow-rate attack detection. Their method is based on analyzing the average rate of packets received by the server over two periods of time. This method is a network-level technique that deals with the frequency of the data received by the Web server. Application-layer attacks including the slow-read tend to evade those techniques that are mostly suitable for volumetric, flooding attacks. The slow-read application layer attack can evade this detection technique, since it does not generally cause a traffic surge. Benign surges in Web traffic such as flash crowds may also be falsely detected as attacks by this technique.

There are also previous work on intrusion detection using Random Forests [11]. Most of these approaches have used the KDD '99 intrusion detection dataset which is very old for the purpose of the current work. The current work is focused on slow-read denial of service attack, which did not exist at the time of the KDD '99 dataset. The current datasets have been generated on the Cloud, since we aim to show the feasibility of the Cloud as a potential denial of service zombie. Moreover, the selected TCP attributes for the classification are also different from those previously used, as our datasets are tailored to this specific attack.

The idea of DDoS-as-a-Service has been already introduced [12], [13]. M. Karami *et al*. show how DDoS-as-a-Service works by briefly analyzing a Cloud service called TwBooter. However, their analysis is limited to that specific service, as opposed to our work which has employed Amazon EC2 as the current dominant Cloud service provider [14]. J. J. Santanna *et al*. discuss *Booter* services and how to find those services on the Web. Compared to our work, this work does not discuss exploiting the Cloud and is limited to existing Booter services. Table I summarizes current detection/mitigation techniques along with the disadvantages of each.

The difference between our work and previous work is threefold. First, we launch an Application-layer DDoS from the Cloud, thus showing the feasibility of initiating those attacks from the Cloud. Second, we generate Cloud-based, attack-specific datasets as opposed to using existing (old) datasets that are not tailored for these specific attacks. Third, we propose using random forests to detect the slow-read attack and show how fine-tuning the classifier's parameters can result in near

TABLE II. SLOW-READ ATTACK PARAMETERS

Parameter Name	Parameter Description
N	Number of HTTP requests sent by the attack
C	Number of connections per second
W	Window range in bytes
R	Receive buffer's read intervals in seconds
B	Bytes to read from receive buffer in a single read operation
P	Pipelining factor for HTTP requests
D	Duration of the slow-read attack

optimal classification of the attack and benign traffic.

IV. SLOW-READ DENIAL OF SERVICE LAUNCHED FROM THE CLOUD (CLOUDZOMBIE)

In this section, we discuss the implementation details of the slow-read attack and the results showing how Cloud can be exploited to launch such attacks with a minimum cost. The target Web server runs on an IBM System x3650 M4 server powered by two 10-core Intel Xeon E5-2680 v2 2.80 GHz processors, 384 GB of memory, and 100 Mbps Internet bandwidth. This powerful server runs Apache HTTP Server 2.4 on an Ubuntu Server 14.04 operating system. Table II shows the parameters that can be used to launch the slow-read attack. The number of HTTP requests (N) and the frequency (C) are the two most important parameters. They specify the volume and the rate of the attack. One of the key attributes of a slow-read is to set the TCP (receive) window size to a small value to force the server to send the responses slowly. Parameter W specifies a range for the random window sizes to be advertised for each request. The attacker can also fine tune at which rate the response data from the server should be read using parameters R and B in Table II. When supported by both parties, the attacker can also use HTTP Pipelining to send multiple requests without waiting for each response. Parameter P in the table can be used to specify the number of times the request can be repeated in a connection. Finally, parameter D determines the duration of the attack.

A. Threat Model

We use Amazon Elastic Compute Cloud (EC2) to launch the slow-read attack. On the EC2 platform, like other Infrastructure as a Service platforms, an attacker can easily get access to a number of virtual machines without being required to spend huge amount of money. Furthermore, the automatic and easy registration process can help the attacker conceal his real identity [1]. Many Cloud service providers also offer free trial periods to users to use their services. Amazon AWS Free Tier offers Micro instances at no charge for 12 months. Thus, launching these attacks will have no cost for an attacker.

Figure 4 shows the slow-read DDoS threat model using the Amazon EC2. In this threat model, attacker A does not need to compromise victim machines and form a botnet to launch the attack. Instead, A creates a virtual machine image I and installs all the required attack tools on it. A can then launch n virtual machines VM_i based on I and attack the target T. This is feasible partly because a slow-rate attack does not need a tremendous amount of bots as opposed to volumetric DDoS attacks. Furthermore, since the attack is not resource

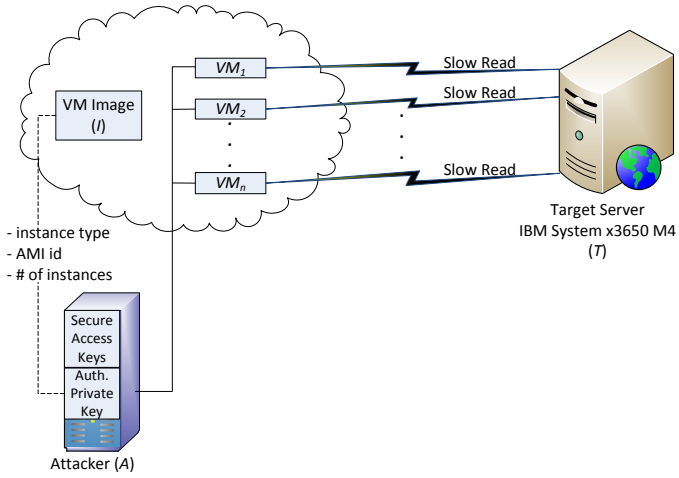

Fig. 4. Slow-read DDoS launched remotely from the Amazon Cloud. The attacker launches a number of Amazon EC2 instances using a VM image which contains the attack tool.

intensive, it can evade the Cloud misuse detection systems. One distinctive feature of the Cloud is that each time a new VM is launched, a new public IP address will be assigned to it. This brings the opportunity for the attacker to evade DDoS detection techniques that blacklist IP addresses.

As the figure shows, A possesses two sets of keys: secure access keys and an authentication private key. Secure access keys consist of an access key ID and a secret access key. Access keys are used to sign Amazon Web Services (AWS) API requests. This allows AWS to authenticate the requester [15]. In order to authenticate users, Amazon EC2 uses public-key cryptography. The authentication private key stored on the attacker machine must be provided to EC2 in order to connect to the virtual machine instances.

Slow-read requires neither high bandwidth nor high computing power on the attacker side. This means that even the least powerful EC2 virtual machines (Micro instances) can be used to launch the attack. Figure 5 shows what happens when a slow-read DoS attack is launched against the target Web server with aforementioned configuration. We have used a Micro instance on Amazon EC2 which has very limited hardware power and network bandwidth; however it has been able to knock down the powerful server after a few seconds (see Service Available line in the figure). In order to confirm the status of the target server, a probe connection automatically probes the server in regular intervals. Moreover, the status is manually checked using the Apache Module *mod_status* to make sure that the victim server is inaccessible for any new connections. The server is considered to be down if no response is received by the probe connection for five seconds.

Security of a system is determined by the weakest link in the system. Although the target Web server was running on a powerful machine, it was easily knocked down through its weakest link; connection pool in this case. This is because the attack exhausted the target server's connection pool and not the powerful computing and networking hardware. One of the other reasons why the Web server in this attack scenario was very easily knocked down is because of the underlying architecture used to handle the incoming requests. In gen-

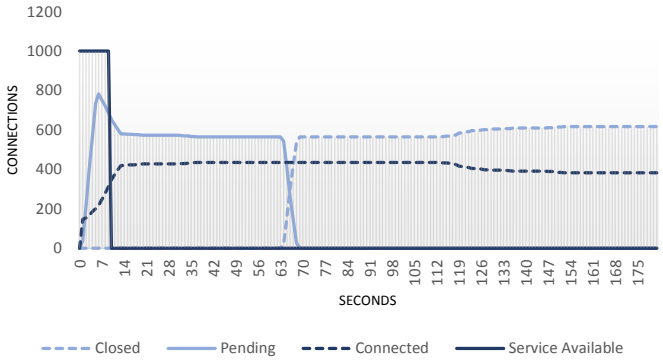

Fig. 5. Slow-Read DoS attack launched from Amazon EC2. The target Web server goes down after 14 seconds. Total number of connections = 1000, Connections/second = 200.

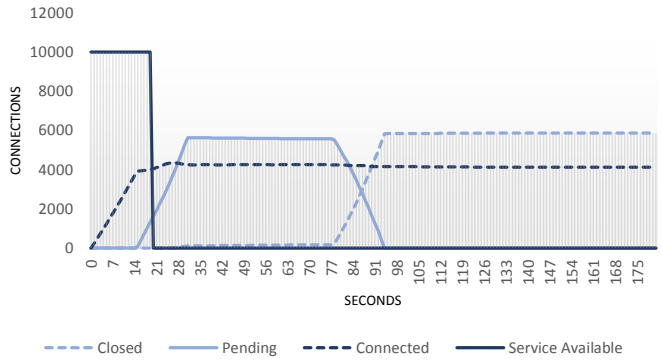

Fig. 6. A target Web server capable of handling 4,000 concurrent connections going down by a slow-read attack launched from Amazon EC2. Total number of connections = 10,000, Connections/second = 1000.

eral, modern Web servers are built on different architectures including non thread-based, thread-based, event-based or a combination of those. In the non thread-based model such as Apache *prefork*, a separate process takes care of answering each incoming request. This may be the most reliable model, because requests are isolated from each other. However, this uses a high amount of system resources and is not suitable for handling numerous number of connections concurrently. In the thread-based architecture model, a single thread is responsible to handle each request as opposed to a separate process.

In this experiment, the number of simultaneous connections that can be handled by the Apache Web server was limited to the default value of 256. By increasing this number to 1,000 and repeating the same experiment, we can prevent the slow-read DoS. This may be one of the simplest techniques to mitigate the attack. Nonetheless, the attacker can also increase the number of requests, and this will not cost much in terms of bandwidth, because each request requires only a tiny amount of bandwidth. In general, the attacker has less limitations in this regard compared to those of the target machine. This is mainly because the number of concurrent connections that can be served by a Web server is always limited to some number l and the attacker can almost always send more than l requests.

Figure 6 shows how a server capable of handling 4,000 concurrent connections can be knocked down by a slow-read attack. In this attack, a total number of 10,000 connections were sent out at the rate of 1,000 connections per second. The receive window ranged randomly between 512 to 1024 bytes and the response was read at the rate of 32 bytes per 5 seconds.

If the receive window size is w, the send buffer size is b, and the requested file size is f, then the following requirement has to be satisfied for a successful slow-read attack:

$$w < b < f \tag{1}$$

V. CloudZombie Detection

We perform experiments for slow-read attack in both DoS and DDoS settings. In our approach, we use Random Forests [16] to detect slow-read denial of service attacks at the destination. Random forests is an ensemble, supervised machine learning technique with excellent classification performance [17]. The learning algorithm is based on building a large number of random decision trees. The forest is the combination of all the trees, and classification is done by voting; the class that has the most votes among all the decision trees will be assigned to the new object. This approach does not have most of the drawbacks of the current solutions, since it is not blocking, disabling, or limiting any functionalities at the server as opposed to the existing solutions.

There are different places as to where a denial of service attack can be detected. The attack can be detected at the source, at the destination, in the intermediate networks or a combination of those (hybrid) [18]. In this paper, we are interested in the destination-based approaches, as they are the most practical among the four options. Network-based approaches cannot be used to detect application-layer DDoS attacks such as slow-read, since network devices (switches and routers) do not have access to application layer data. The other detection deployment location is at the source of the attack. There are a number of source-based DDoS mitigation techniques such as *Ingress/Egress* filtering, monitoring inbound and outbound traffic and comparing to a normal flow, detecting bidirectional packet flow imbalances, *etc.* [18]. Furthermore, besides the technical limitations of each of these techniques, there is a big concern in implementing the techniques in practice; the party implementing a source-based DDoS detection technique may not necessarily be the one benefiting from it. In other words, a company needs to have source-based mitigation techniques in place to protect victims residing on other companies' networks. This makes source-based DDoS mitigation techniques much less practical compared to source-based approaches.

In our experiments, the attack and benign HTTP requests are launched on Amazon EC2, and the training dataset is created based on the generated TCP logs. Figure 7 shows the simplified process of generating the datasets for the random forests classifier. We have developed a Python program to remotely launch the attack from outside the Amazon Cloud. The program employs Amazon EC2 APIs to create and launch the requested instances. The program then remotely calls the attack script which is stored on the instance in order to run the slow-read. The benign dataset is generated using a similar process, but using a different program.

The *tcpdump* packet analyzer which is run on the target server collects the raw (binary) TCP log for all the malicious

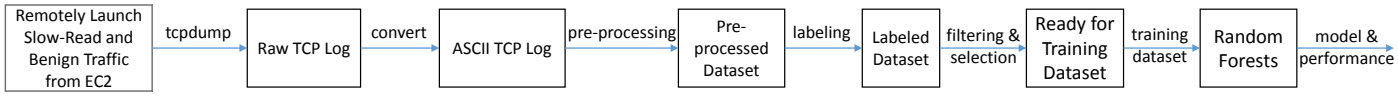

Fig. 7. The simplified process of preparing training dataset for the random forests classifier.

and benign connections. This raw log is then converted into a readable ASCII format. During the pre-processing, different fields of each packet are shifted so that the resulting output has an ordered list of fields for each packet. This creates the initial dataset which is then labeled to identify attack and benign packets. After filtering redundant records and selecting appropriate attributes, the resulting dataset is fed into the random forests classifier and the performance of the classification is measured. The dataset consists of 12,357 TCP attack and 36,069 TCP benign packets in the non-distributed (DoS) experiment.

Each TCP packet captured during the experiments has 15 different fields, but not all of them are relevant attributes. A relevant attribute is one that has correlation with the target attribute (here, attack or benign) and also has predictive power. For instance, the *Destination port* field has no predictive power, because it is always 80 (HTTP default), or the *Destination IP* is always the same in our experiments. Therefore, we have only selected TCP header fields that are the most relevant in a slow-read attack. These attributes are: source port, source IP, flags, window size, and window scaling.

The simplest metric to measure the performance of a machine learning algorithm is the prediction accuracy, namely what percentage of the objects are correctly classified. Usually the cross validation technique is used to get an unbiased estimate of the test set. Cross validation partitions the dataset into a number of subsets (*k*). Then each of the k subsets is used for testing once while the remaining $k-1$ subsets are being used for training. The average of all the generated models is calculated to estimate the statistical performance of the algorithm. However, this type of validation is not needed to estimate the performance of random forests, since the training algorithm uses only 2/3 of the objects and the remaining 1/3 can be used for testing the error rate [16].

$$Entropy(D) = -\sum_{i=1}^{n} p_i \log_2 p_i \qquad (2)$$

In order for the random forests algorithm to split a tree node, different splitting criteria can be used such as Information Gain, Gain Ratio, Gini Index, *etc.* If Information Gain is used, the entropy of all the attributes is calculated, then the attribute with the minimum entropy is selected for splitting. Formula 2 shows the Shannon entropy [19] of a dataset *D*. The number of distinct classes is denoted by *n*, and p_i denotes the probability of a random record in the dataset to belong to class C_i. This method is biased towards the attributes with a large number of distinct values. Gain Ratio is a variant of Information Gain and tries to reduce the bias by taking into account the size and number of a decision tree branches when selecting an attribute. The Gini Index measures the heterogeneity of the dataset. It selects an attribute so that the average Gini Index of the resulting subsets decreases. This heuristic is not biased like Information Gain. Based on a theoretical and empirical study [20] and the initial experiments with different splitting criteria on our dataset, the Gain Ratio criterion results in the highest accuracy and lowest false positive and negative rates on average for our dataset. As a result, we select this criterion for splitting the decision trees within the random forests. We use RapidMiner Studio Professional [21] to perform the modeling task.

The distributed attack dataset consists of a total of 234,004 attack and 247,161 benign TCP packets created by simultaneously launching 10 Micro instances on the Amazon EC2. The non-distributed (DoS) version of the slow-read attack may not be very effective, since like other DoS attacks, all the packets are coming from a single IP source. Modern intrusion detection systems should be able to detect such attack quite easily. A distributed attack, however, is more challenging to detect, and can be more effective. In order to initiate a slow-read Cloud-based DDoS attack, our program uses Amazon EC2 APIs to launch a number of EC2 instances in parallel from an EC2 virtual machine image. It then runs the slow-read attack script on each of them. The image already stores the required attack binaries and scripts.

VI. Experimental Results and Discussion

We have generated both slow-read DoS and slow-read DDoS datasets employing Amazon EC2 Cloud. For building the random forests classifier, we have tried different number of trees ranging from 2 to 128 on a logarithmic scale. Increasing the number of trees in a random forests classifier generally increases the classification performance. Nonetheless, the performance gain grows asymptotically [22]. In other words, the performance gain beyond some point is not worth the higher computational cost anymore.

We have also repeated all the experiments with and without pre-pruning decision trees in random forests. Pre-pruning may result in generating smaller trees which reduces memory and processing costs. However, these trees may not be grown enough to provide a high classification performance. We have also performed all the experiments using pruning process. However, pruning did not affect the accuracy or error rates. In the pruning technique, leaf nodes in a decision tree that do not improve the discriminative power of the classifier are removed. This reduces the complexity of the trees and also overfitting. Pre-pruning is a type of pruning which is done before the tree classifies the training set, in parallel to the tree creation. We try to find out the best number of trees and whether pre-pruning has a positive or negative impact on the performance of the generated classifiers.

There are different metrics for measuring a binary classification performance. Accuracy is the ratio of correct predictions to the total number of records in the dataset. Area Under the Curve (AUC) is the area under the Receiver Operating

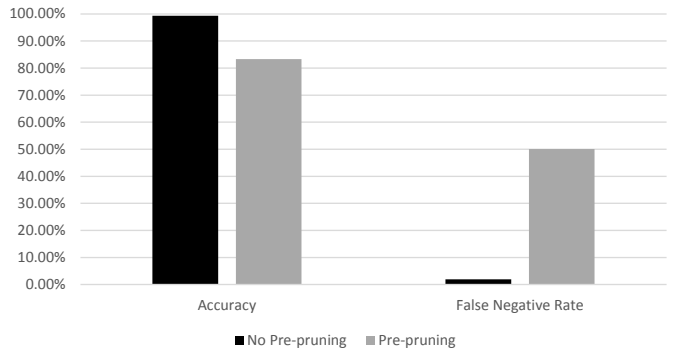

Fig. 8. Comparison of the accuracy and false negative rate with and without applying pre-pruning to detect Slow-Read DoS. No pre-pruning outperforms pre-pruning.

Characteristic (ROC) curve. The ROC curve is a plot of the true positive rate against false positive rate at different thresholds. This plot visualizes the performance of a binary classifier. An AUC of 0.5 basically represents a random classifier and an AUC of 1.0 belongs to a perfect classifier.

Figure 8 shows the results of using the random forests algorithm on the generated dataset for the slow-read DoS. The figure shows the accuracy and false negative rate metrics with and without applying tree pre-pruning. False positive rate is not shown in this figure as it is 0% for both cases. As the figure shows, higher (99.37% *vs.* 83.34%) accuracy is resulted from the classifier when pre-pruning is not used. Moreover, a much lower (1.90% *vs.* 50.10%) false negative rate is resulted in this case. As shown in the figure, pre-pruning has prevented the trees in the forest to grow enough to provide a better classification performance. The high false negative rate shown in the figure for pre-pruning shows that half of the attack packets were not correctly classified.

Figure 9 shows the ratio of the AUC metric to the number of the decision trees with and without using pre-pruning. As the number of the trees increases, AUC increases as well. In both cases, AUC converges when random forests is made of 32 trees. This asymptotic growth means that there is no performance gain if more than 32 trees are used in the random forests classifier. Using more trees will only increase memory and processing costs in this case. As the figure shows, for all different numbers of trees tested, no pre-pruning outperforms pruning. Moreover, when pre-pruning is used with only two or four trees in the forest, AUC is 0.5. This means that the classifiers have performed not better than a random classifier.

Based on the results obtained for the slow-read DoS attack detection, we found that applying pre-pruning results in lower accuracy and AUC, and higher error rates. The empirical results show that pre-pruning prevents most trees to grow enough to provide an accurate prediction. We did similar experiments on the distributed attack dataset to find out what can best detect the slow-read DDoS. Figure 10 shows the accuracy and false negative rate of the random forests classifier for the slow-read DDoS dataset. Like the DoS dataset discussed earlier, higher accuracy (99.60% *vs.* 88.89%) and much lower false negative rate (0.64% *vs.* 10.80%) are resulted when no pre-pruning is used.

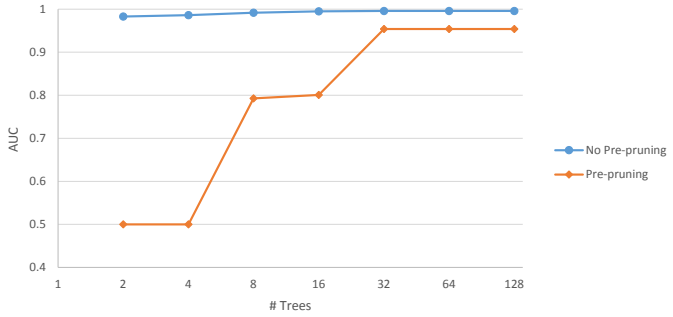

Fig. 9. Comparison of the AUC metric with and without pre-pruning using different numbers of trees to detect Slow-Read DoS. No pre-pruning outperforms pre-pruning, but both converge (at # Trees = 32) as the number of trees in random forests increases.

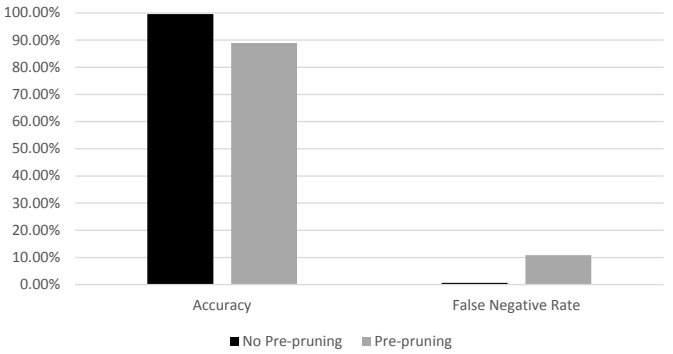

Fig. 10. Comparison of the accuracy and false negative rate with and without applying pre-pruning to detect Slow-Read DDoS. No pre-pruning outperforms pre-pruning.

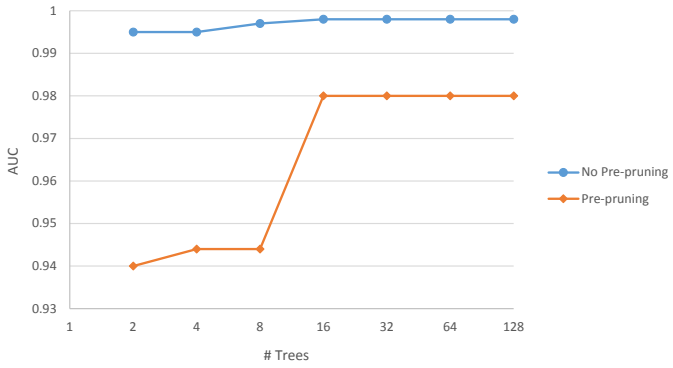

Fig. 11. Comparison of the AUC metric with and without pre-pruning using different numbers of trees to detect Slow-Read DDoS. No pre-pruning outperforms pre-pruning, but both converge (at # Trees = 16) as the number of trees in random forests increases.

Figure 11 shows the AUC to the number of trees ratio for the distributed attack. As the number of trees increases, the AUC increases, as well. The growth happens whether pre-pruning is used or not. Nevertheless, AUC is higher with no pre-pruning for all the different numbers of trees tested. Generally, by increasing the number of trees, true positive rate increases and false positive rate decreases. Consequently, AUC increases asymptotically by increasing the number of trees in the random forests. As shown in the figure, AUC converges where 16 trees comprise the random forests.

We are not aware of any detection technique for the slow-read attack that is quantitatively comparable with our proposed technique. This is partly due to the fact that we have generated our unique dataset on the Cloud and for the specific slow-read attack. Moreover, since it is the first time machine learning is used to detect this attack, no other results are available with the same metrics such as classification accuracy.

Our approach does not have most of the drawbacks of the existing solutions, since it is not blocking, disabling, or limiting any functionalities at the server as opposed to the current solutions. We have also tried other machine learning algorithms such as K-NN to detect slow-read attack. However, K-NN had an accuracy of about 50%, which is much lower than the accuracy of the Random Forests algorithm.

VII. Conclusion

Cloud is becoming more prevalent and less expensive to use. There are, however, many security threats to the Cloud including denial of service. Denial of service attacks are one of the most notorious attacks on the Internet. A slow-read attack is a new type of application-layer denial of service attack that sends complete HTTP requests, but reads the responses slowly to exhaust the server resources such as the connection pool. The slow-read attack is easier to launch and harder to detect as opposed to volumetric attacks that require tremendous amount of resources on the attacker side.

In this work, we look at the Cloud from a different perspective (CloudZombie). We show how easily the least powerful virtual machines on the Amazon EC2 Cloud can make a powerful IBM System x server defunct within a few seconds. We propose a novel solution based on random forests classifier to detect such attacks. Our experiments show that this approach offers a high classification accuracy and AUC on the attack and benign traffic along with low false positive and negative rates. High accuracy and low false positive and negative rates show the performance of the random forests for the datasets used in the experiments. On the other hand, high AUC shows the high aggregated classification performance and predictive power of the random forests to classify unseen data. We further show the minimum number of trees required in the random forests to have the highest performance while keeping computing costs at the lowest possible.

There are two drawbacks in using the random forests classifier. First, it is an opaque and not a transparent classifier. As a result, it is hard to tell how the prediction is done. This is not a technical limitation, though. Second, the random forests classifier can be expensive to deploy using large number of trees. The other limitation of our approach is that we build a binary classifier which can classify attack and benign packets for only a specific attack. Therefore, it is not capable of detecting different types of denial of service attacks.

Acknowledgment

The first author would like to thank Sergey Shekyan for answering his many questions regarding the slow-read attack. The authors would also like to thank Jenny Chien for helping in pre-processing the datasets for the experiments. Finally, we would like to thank the anonymous reviewers for their insightful and constructive comments.

References

[1] S. Shafieian, M. Zulkernine, and A. Haque, "Attacks in public clouds: Can they hinder the rise of the cloud?," in *Cloud Computing*, pp. 3–22, Springer, 2014.

[2] R. Chang, "Defending against flooding-based distributed denial-of-service attacks: a tutorial," *Communications Magazine, IEEE*, vol. 40, pp. 42–51, Oct 2002.

[3] S. Shekyan, "Are you ready for slow reading?." https://community.qualys.com/blogs/securitylabs/2012/01/05/slow-read, 2012. Last Accessed: 15 July 2015.

[4] N. S. H. Gonzalez, M.A. Gosselin-Lavigne and A. Ghorbani, "The impact of application layer denial of service attacks," in *Case Studies in Secure Computing - Achievements and Trends* (B. Issac and N. Israr, eds.), CRC Press, 2014.

[5] G. Kambourakis, T. Moschos, D. Geneiatakis, and S. Gritzalis, "Detecting dns amplification attacks," in *Critical Information Infrastructures Security*, pp. 185–196, Springer, 2008.

[6] Trustwave, "Modsecurity." https://www.modsecurity.org. Last Accessed: 15 July 2015.

[7] M. Aiello, E. Cambiaso, M. Mongelli, and G. Papaleo, "An online intrusion detection approach to identify low-rate dos attacks," in *Security Technology (ICCST), 2014 International Carnahan Conference on*, pp. 1–6, IEEE, 2014.

[8] Netcraft, "May 2015 Web server survey." http://news.netcraft.com/archives/2015/05/19/may-2015-web-server-survey.html, 2015. Last Accessed: 15 July 2015.

[9] E. Cambiaso, G. Papaleo, and M. Aiello, "Taxonomy of slow dos attacks to web applications," in *Recent Trends in Computer Networks and Distributed Systems Security*, pp. 195–204, Springer, 2012.

[10] J. Park, K. Iwai, H. Tanaka, and T. Kurokawa, "Analysis of slow read dos attack," in *Information Theory and its Applications (ISITA), 2014 International Symposium on*, pp. 60–64, IEEE, 2014.

[11] J. Zhang, M. Zulkernine, and A. Haque, "Random-forests-based network intrusion detection systems," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 38, no. 5, pp. 649–659, 2008.

[12] M. Karami and D. McCoy, "Understanding the emerging threat of ddos-as-a-service.," in *6th USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET '13)*, 2013.

[13] J. J. Santanna and A. Sperotto, "Characterizing and mitigating the ddos-as-a-service phenomenon," in *Monitoring and Securing Virtualized Networks and Services*, pp. 74–78, Springer, 2014.

[14] "Gartner magic quadrant for cloud infrastructure as a service." https://www.gartner.com/technology/reprints.do?id=1-1UKQQA6&ct=140528&st=sb, 2014. Last Accessed: 15 July 2015.

[15] "Signing aws api requests." http://docs.aws.amazon.com/general/latest/gr/signing_aws_api_requests.html. Last Accessed: 15 July 2015.

[16] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[17] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proceedings of the 23rd international conference on Machine learning*, pp. 161–168, ACM, 2006.

[18] S. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks," *Communications Surveys Tutorials, IEEE*, vol. 15, pp. 2046–2069, Fourth 2013.

[19] C. E. Shannon, "Prediction and entropy of printed english," *Bell system technical journal*, vol. 30, no. 1, pp. 50–64, 1951.

[20] V. Y. Kulkarni, M. Petare, and P. Sinha, "Analyzing random forest classifier with different split measures," in *Proceedings of the Second International Conference on Soft Computing for Problem Solving (SocProS 2012), December, 2012*, pp. 691–699, Springer, 2014.

[21] "Rapidminer studio analytics platform." https://rapidminer.com/products/studio. Last Accessed: 15 July 2015.

[22] T. M. Oshiro, P. S. Perez, and J. A. Baranauskas, "How many trees in a random forest?," in *MLDM*, pp. 154–168, Springer, 2012.