2015 IEEE 39th Annual International Computers, Software & Applications Conference

Web Server Protection against Application Layer DDoS Attacks using Machine Learning and Traffic Authentication

Jema David Ndibwile, A. Govardhan
School of IT. Jawaharlal Nehru Technological University
Hyderabad, India
jemablue86@gmail.com

Kazuya Okada, Youki Kadobayashi
Graduate School of Information Science. Nara Institute of Science and Technology
Nara, Japan
{kazuya-o, youki-k}@is.naist.jp

Abstract— **Application layer Distributed Denial of Service (DDoS) attacks are among the deadliest kinds of attacks that have significant impact on destination servers and networks due to their ability to be launched with minimal computational resources to cause an effect of high magnitude. Commercial and government Web servers have become the primary target of these kinds of attacks, with the recent mitigation efforts struggling to deaden the problem efficiently. Most application layer DDoS attacks can successfully mimic legitimate traffic without being detected by Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS). IDSs and IPSs can also mistake a normal and legitimate activity for a malicious one, producing a False Positive (FP) that affects Web users if it is ignored or dropped. False positives in a large and complex network topology can potentially be dangerous as they may cause IDS/IPS to block the user's benign traffic. Our focus and contributions in this paper are first, to mitigate the undetected malicious traffic mimicking legitimate traffic and developing a special anti-DDoS module for general and specific DDoS tools attacks by using a trained classifier in a random tree machine-learning algorithm. We use labeled datasets to generate rules to incorporate and fine-tune existing IDS/IPS such as Snort. Secondly, we further assist IDS/IPS by processing traffic that is classified as malicious by the IDS/IPS in order to identify FPs and route them to their intended destinations. To achieve this, our approach uses active authentication of traffic source of both legitimate and malicious traffic at the Bait and Decoy server respectively before destined to the Web server.**

Keywords—DDoS Mitigation, Machine Learning, JavaScript, False Positives, IDS/IPS

I. INTRODUCTION

Application layer Distributed Denial of Service (DDoS) are among the most dangerous form of network attacks that are launched by an attacker using specific and well-written scripted tools or through thousands of compromised networked hosts (botnets) through Command and Control (C&C) or Internet Relay Chat (IRC) servers. These kinds of attacks prevent legitimate users from accessing server resources due to service and connection unavailability. Companies also lose a sizeable amount of revenue due to the application layer DDoS attacks as attacks on commercial Web servers can successfully damage business-critical services and lead to a serious loss within a short period of time. The average financial cost that small-to-medium-sized businesses incur per DDoS attack incident is $52, 000 and $440, 000 for larger businesses [1]. Most companies today do not have adaptive filtering capability to filter the typical DDoS attacking methods like SYN and HTTP GET floods [1]. Synchronize (SYN) and Hypertext Transfer Protocol (HTTP) GET floods remain devastating for those companies that do not employ proper onsite DDoS mitigation [2]. An application layer DDoS attack has become a very popular and lethal method of attack due to its accomplishment, simplicity and defensible complexity [3]. While Web servers have become victims of legitimate lookalike attack traffic, Web users have become victims of false positives. False positives are common and happen to every intrusion detection and prevention systems [4]. The false positives can occur due to various reasons: IDS/IPS's imprecisely crafted rules, legitimate applications that do not strictly follow request for comments (RFCs), application not seen in the training phase of anomaly detection, too broadly written signature and anomalous standard differences such as behavior that is highly suspected in one area of an organization is acceptable in another. False positive is one of the biggest problem handicapping IDS/IPS implementations as they can easily drown out legitimate IDS/IPS alerts and create more alerts in a short period of time.

In order to protect Web servers against application layer DDoS attacks various techniques had been proposed, developed and implemented, however, malicious traffic that hides behind legitimate traffic is still a common plague among those mitigation techniques. Most of them fail to explicitly distinguish between mimicry malicious and legitimate traffic with inevitably false positives and negatives production. To prevent application layer DDoS attacks targeting Web servers, we can deploy intrusion detection/prevention system near or at the network gateway that is assisted by traffic source authentication implemented at a host that intercepts the communication before it reaches the destination server. There are various authentication methods for traffic source, however, they are handicapped by implementation methods where the same machine hosting the Website also does authentication concurrently, slowing down the communication and exhausting the host machine. Separation of these two processes is important and simplifies the mitigation process. Malicious source traffic can be authenticated at a specially designed destination such as a Decoy Web server, which is a clone of Real Web server, instead of dropping them so as to retrieve false positives packets.

Therefore, an ideal solution would be to deploy three separate Web server machines by setting up one machine to receive pre-processed legitimate traffic, a second machine to

0730-3157/15 $31.00 © 2015 IEEE
DOI 10.1109/COMPSAC.2015.240

host a Website that receives authenticated traffic from the first machine and a third machine receives malicious traffic to look up for misclassified legitimate traffic such as FPs. Most mitigation solutions do not explicitly restore false positive associated traffic but rather ignore them, focusing instead on lowering FPs production by the IDS/IPS. In other cases, an attacker, instead of compromising networked hosts to launch DDoS attack, uses popular application layer DDoS tools such as SlowLoris, Slow-Post, R-U-Dead-Yet (RUDY) and anonymous High/ Low Orbit Ion Canon (HOIC/LOIC). In such case, early detection and mitigation of the malicious traffic is very important and crucial towards application layer DDoS mitigation. Special anti-DDoS module can be developed by analyzing various kinds of traffic generated by those tools and compare them with normal traffic to establish rules for each individual tool.

In this paper, we propose, implement and prove our concept by an approach that uses three servers, namely: Real Web server, Bait Web server and Decoy Web server, together with a customized intrusion prevention system at the network gateway that uses rules generated by random tree machine-learning algorithm through supervised learning.

The rest of this paper is organized as follows: Section II briefly discusses about application layer DDoS attacks mitigation related works. Section III describes our proposed approach and its implementation. Section IV discusses some results of the experimental approach carried out. Section V discusses challenges and section VI makes a conclusion of this paper.

II. RELATED WORKS

In brief, we will first discuss the difficulty of protecting Web servers against application layer DDoS attacks. We will focus on attacks that specifically aim to manipulate Websites functionalities such as HTTP flood, GET and POST, and we will then discuss how our proposed approach can overcome this problem. There are numerous ways of protecting Web servers against DDoS attacks aiming to deny user's access to the hosted Website and its associated resources. Categorically, the protection methods mostly fall under prevention, mitigation, identification and redirection or routing phase. In each phase researchers deploy various methods of implementation to mitigate these attacks.

A. Complexity of Application Layer DDoS Attack Mitigation

DDoS attacks that are targeting the application layer services of a Web server are extremely tricky and difficult to mitigate due to variety of reasons. First, most can successfully establish a legitimate Transfer Control Protocol (TCP) connection [5], which is the major means of connection, and thus escape mitigation radar. Attacking packets are then well crafted by skillful attackers to disguise legitimate traffic and are shielded by legitimate traffic on their way to the destination host or network. Malicious traffic detection contrivances for Internet Protocol (IP) packet is then handicapped by the fact that the established legitimate TCP connection by an application layer DDoS attack necessitates a provision for legitimate IP addresses and IP packet [3, 5]. Finally, the application layer DDoS attacks utilize little bandwidth, sometimes sending fewer packets and require little attacking effort [6] which make them less suspicious, leaving the victim to suspect a trivial cause such as machine fault or malfunction.

B. Web Traffic Source Authentication

Some previous methods authenticate the traffic source by using CAPCTHA, checkboxes, cookies and JavaScript in various ways. The most popular authentication method among these is CAPTCHA [7], however, it annoys users, as they are required to enter some visually distorted characters so often in a designed blank prompt. It is also ineffective since smart bots can solve such kinds of CAPTCHA puzzles by using Artificial Intelligence (AI) with 99.8% accuracy [8]. Other approach that requires hardware manipulation is Google CAPTCHA reCAPTCHA approach when manipulating Web pages and forms [8], while other methods use mathematical puzzles either inside a plain page or JavaScript pop-up prompt [9]. However, all these methods require a little bit of user's time and intelligence. CAPTCHA also had been criticized and labeled as a significant barrier to Internet use for people with sight problems or disabilities such as dyslexia by a BBC journalist [10]. Our proposed method follows the same logic but with quite different approach and implementation; It does not require processing time and can help people with sight problems to successfully authenticate traffic source. We also include more suggestions to alternatives in our approach for visually impaired users in the discussion section.

C. Oblivious DDoS Mitigation

Keeping an attacker oblivious of the mitigation effort is vital aspect of DDoS attack mitigations where malicious traffic detected at the network gateway is redirected to the Decoy Web server, which is a clone of the Real Web server [11]. This method can be incorporated to application layer DDoS attacks mitigation so as to dissuade an attacker while mitigation efforts continue. However, mimicry malicious traffic can circumvent this mitigation procedure by camouflaging within normal traffic [11]. So when deploying this method, some considerations should be taken for mimicry malicious traffic.

III. OVERVIEW OF OUR MITIGATION APPROACH

In this section, we technically describe our proposed mitigation approach that is based on active authentication of both normal and malicious traffic sources. We acknowledge Web user's legitimate traffic that is mistaken for malicious by the IDS/IPSs and consider them as important as any other legitimate traffic. In our approach, the Decoy Web server and Bait Web server use the same IP address and behave similarly [11]. An attacker has no idea where his traffic is going. At this point the custom IPS decides where to route them based on its given rules. We use random tree algorithm rules to incorporate with Snort Network Intrusion Prevention System (NIPS) signatures at the network gateway and redirect all traffic on alarm rise to the Decoy Web server. Since we are using Snort NIPS, we will use the term NIPS more often in the coming sections rather than IDS/IPS. In this case a client and attacker's computers are communicating with a Bait Web

server, which is seen publicly by the outside world unlike the Decoy and Real Web servers. We configure the Real Web server to host a full loaded Website and listen to all incoming requests through the Bait Web server. On the other hand, we configure the Bait Web server to primarily listen to all incoming requests through standard port and act as a proxy. The Bait Web server determines what to do with the received traffic; in this case it sends authenticated traffic to the Real Web server and unauthenticated traffic to the Decoy Web server, depending on the user's action on the authentication method. The sequence of traffic flow is as shown in Fig. 1 below.

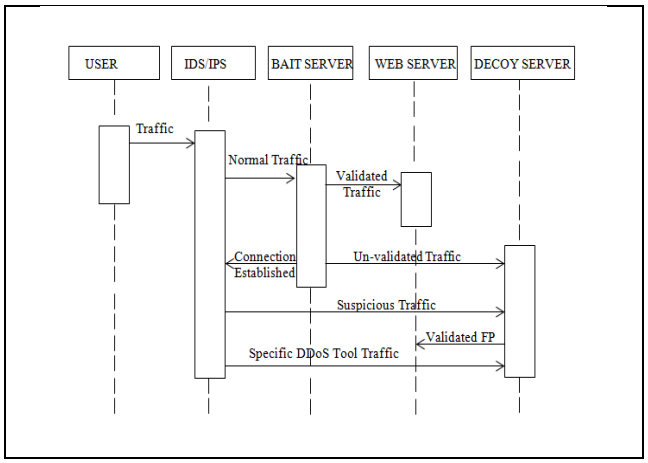

Fig. 1. Mitigation of DDoS attack traffic sequence.

Our mitigation approach uses rules generated by machine-learning algorithm that was trained offline by labeled datasets. Particularly, we choose decision trees to build a classification model because of their well-known good performance in classification of vast amount of data. We classify our data into two major classes of traffic that are malicious and normal traffic classes while malicious class includes sub-classes of various DDoS tool traffic. Traffic in classes are separated and routed to their respective destinations as per our architecture. We obtain labeled datasets from public repositories such as MAWILab [12], NETRESEC [13] and generate our own DDoS tools traffic from free downloadable tools such as SlowLoris, Slow-Post, RUDY, HOIC and LOIC. Training dataset is divided into 60% of the entire dataset while the remaining 40% is sub-divided into 50% cross-validation and 50% testing datasets. All datasets are disjoint and have no duplicate within a particular set.

We use function approximation for our problem setting for this classification where x denotes set of possible instances and y denotes set of possible labels. Equation (1) is the function for an unknown target. Each instance x in X denotes a vector of a feature where (2) is a function hypotheses set. The unknown target function f in training is given by an example as in (3) where its hypothesis is given by (4).

$$f: x \rightarrow y \tag{1}$$

$$H = \{h \mid h: x \rightarrow y\} \tag{2}$$

$$\textbf{Input: } \{(x^{(i)}, y^{(i)})\}^{n} = \{(x^{(1)}, y^{(1)})\}, \ldots, \{(x^{(n)}, y^{(n)})\} \tag{3}$$

$$\textbf{Output: } h \in H \text{ best approximate } f \tag{4}$$

We measure impurity degree by entropy-based metric where $H(x)$ of a random variable x is given by (5) and n is the number of possible values of X. Fig. 2 shows classification model where it takes labeled data X and Y as an input and train the classifier to develop a model that predicts the outcome basing on the new data x.

$$H(x) = -\sum_{i=1}^{n} P(X=i) \log 2\, P(X=i) \tag{5}$$

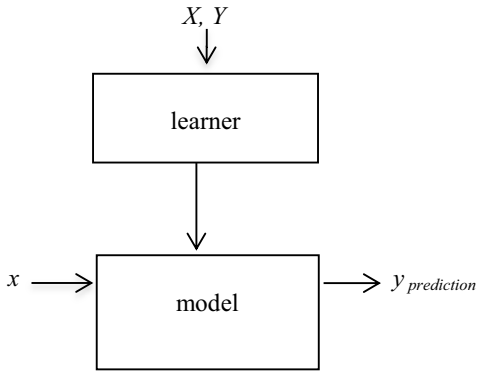

Fig. 2. Classification model for traffic classes prediction.

After training the classifier with labeled datasets as in (6), new data such that $x \sim D(x)$ with unlabeled instances is given by prediction model (7).

$$model \leftarrow classifier.train\ (X,Y) \tag{6}$$

$$y_{prediction} \leftarrow model.predict\ (x) \tag{7}$$

Table I shows the classification results of data from the random tree machine-learning algorithm in training, cross-validation and testing phase.

TABLE I. NORMAL VS. MALICIOUS TRAFFIC CLASSIFICATION RESULTS

Instances	Training	Cross-Validation	Testing
Correctly Classified	100%	97.5542 %	99.022%
Incorrectly Classified	0%	2.4458 %	0.978%
False Positive Rate	0.000	0.027	0.011

Malicious traffic detection is effective in machine learning as the random tree algorithm classification results show in the table I above. However, the classification prowess is best with

known data. Same classifier slightly reduces its classification prowess on the repeated tests in cross-validation with unseen datasets that was part of the entire datasets. In reality, this means that, new data traffic such as new attacks that were never a part of training phase have a higher likelihood of evading the classifier and fly under its radar.

Table II shows the classification results of data that are labeled either normal or specific DDoS tool such as (SlowLoris, Slow-Post, RUDY, HOIC, LOIC). We analyze traffic of various free DDoS software and create labels for each tool. In training phase we mix the traffic with normal traffic to develop a classification model. In this classification, the results are pretty much better compared to the former. There is a very small chance of these kinds of traffic to pass through the detectors undetected if their associated tools were part of the training phase. Classification of these kinds of attack traffic is more straightforward than attack traffic generated by C&C or IRC servers through compromised networked hosts.

TABLE II. NORMAL VS. DDOS TOOL TRAFFIC CLASSIFICATION RESULTS

Instances	Training	Cross-Validation	Testing
Correctly Classified	100%	99.8097 %	100%
Incorrectly Classified	0%	0.1903 %	0%
False Positive Rate	0.00	0.00	0.00

We use general approach for solving classification problems that involve two classes as in the table III below. Table IV shows results of real data on two-class classification approach with malicious and normal traffic labeled datasets.

TABLE III. CONFUSION MATRIX FOR A 2-CLASS PROBLEM

		Predicted Class	
		Class = 1	Class = 2
Actual Class	Class = 1	f_{11}	f_{10}
	Class = 2	f_{01}	f_{00}

TABLE IV. CONFUSION MATRIX FOR LABELED DATASETS

a	b	<-- classified as
12475	0	a = MALICIOUS
0	22475	b = NORMAL

A. Anti- DDoS Tool Module

This module is added to the custom Snort NIPS and easily identifying malicious traffic associated with a specific DDoS tool. It can identify as many application layer DDoS attacks as possible, subject to the number of tools used in the training phase. In this paper, we have chosen only few tools as mentioned earlier for demonstration purpose and its classification is straight forward as traffic behavior of these tools does not change unless there are some major updates or overhaul by the developers. Table V below shows the confusion matrix of these tools.

TABLE V. CONFUSION MATRIX FOR MULTI CLASS DDOS TOOLS

a	b	c	d	e	f	<-- classified as
22475	0	0	0	2	0	a = NORMAL
0	3267	0	0	0	0	b = SlowLoris
0	0	1952	0	0	0	c = LOIC-HTTP
1	0	0	5721	0	0	d = LOIC-TCP
0	0	0	0	12835	0	e = XOIC
0	0	0	0	0	3568	f = HOIC

Our special anti-DDoS module detects specific software used to generated the attack traffic and mitigate it. The module uses the trained classifier to check important network traffic metrics and compare them to the one always exhibited by the attacking tools. Metrics such as port numbers, unique bytes sent, unique bytes received, number of packets and duration of time the associated traffic last in one flow can be matched with the pre-existing trained classifier information. For example a tool like SlowLoris most likely sends only one byte of request data at a time and receives nothing in return rather than making server busy by reserving resources for later completion of the request that will never happen. Furthermore it mostly uses source port numbers between 46,021 and 60,483 when total unique bytes sent from source to destination is less or equal to seven and port number between 25, 796 and 46, 021 when the unique bytes sent are 17 to 19 and 0 bytes replied. Same goes to other tools with same metrics but different classification criteria. On the other hand, tools like anonymous HOIC and LOIC are written in basic codes that are not too sophisticated to write, however the attacks by these tools are very intense as they use boosters and multiple threads at one time. In our classifier, the model has been developed to recognize and categorize any attack that comes from the tools that its traffic was seen in the training phase as in Fig 3 below.

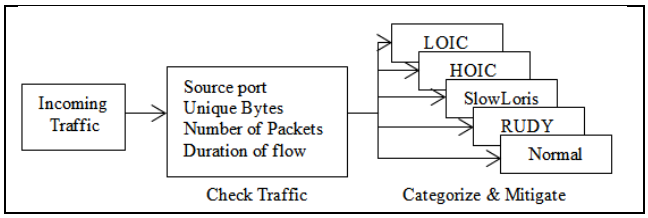

Fig. 3. Incoming traffic checked against traffic metrics based on the decision rules for specific DDoS tools attack traffic.

B. Overview of Bait Web Server

The Bait Web server is a lightweight server with only a copy of a single front page of the real Website and an automatic JavaScript that pops-up on loading the page. The JavaScript pop-up contains a welcome note for user, such as "hello welcome to *www.example.com*". The user then has to

click “Ok” as a sign of accepting the invitation. This method thwarts and traps botnets and mimicry malicious traffic hiding behind legitimate traffic. The method is less annoying than CAPTCHA since it does not require much user effort. In addition, the user may not realize that he has been tricked to verify himself as human. Unattended JavaScript pop-up signifies that a human being did not generate the associated traffic and the system treats it as malicious traffic. At the Decoy Web server there is the same authentication method where malicious traffic is authenticated in order to retrieve false positives associated traffic to the intended Web server.

C. Overview of Decoy Web Server

In contrast to Bait Web server, the Decoy Web server is configured to behave like Real Web server by mirroring each other and directly receives malicious traffic from NIPS, authenticate it and forward to the Real Web server with the help of JavaScript pop-up as implemented at the Bait Web server. Most intrusion detection and prevention systems, when receiving completely new traffic, tend to produce false positives at some point. So this method helps the system to identify user’s legitimate traffic like false positives among malicious traffic and retrieve them to the intended destination.

IV. EXPERIMENT

In this section we show how traffic flows through our custom NIPS and is authenticated at their respective destinations. We also show the amount of correctly identified malicious traffic by feeding to the system with unfamiliar legitimate traffic together with application layer DDoS traffic. We evaluate the Bait and Real Web server performances in terms connection response time. In Fig. 4, we use Ubuntu Server 14.04.2 LTS for Decoy, Bait and Real Web servers.

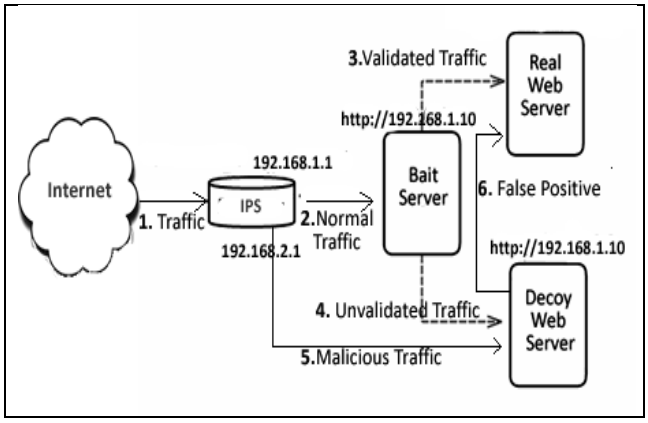

Fig. 4. Network setup on a VMware for experiments.

A. Methodology

Figure 4 outlines the traffic flow through the virtualized network topology used in our experiments. The network consists of a Decoy, Bait, and Real Web server and custom intrusion prevention system. The Internet represents hosts, such as attackers and benign users that access the virtualized network. We implement our proposed approach by editing and adding our random tree decision rules into custom IPS which is Snort IPS where we fine-tune it by modifying *snort.conf* file in */user/local/etc/snort.conf*. The default action in Snort is *“alert”* but it can be changed to *log, pass, drop, sdrop or reject.* Since Snort IPS does not redirect traffic, we manually redirect traffic that has raised alarms to the Decoy Web server by using multi-IP in conjunction with custom codes, Linux netfilter and iproute2. We configure incoming traffic from the Internet to pass through eth0 interface at out NIPS and outgoing traffic through eth1 and eth2 interfaces. By multi-IP address we route malicious and normal traffic through different gateways to two different destinations with the same IP address according to the rules on the Snort IPS. The gateway router maintains whitelist and blacklist tables and updates them according to the NIPS alerts. The router is then responsible for routing the blacklist-associated traffic to the decoy server and maintains the whitelist-associated traffic to the Bait Web server. The template configuration of this tool is publicly available for customization in [14].

We installed Apache2 Web server in the Decoy, Bait and Real Web server and host a real Website on the Decoy and Real Web server. Bait Web server hosts only an *index.html* page of the actual Website. At the Bait and Decoy Web servers we attach a small JavaScript code within index page html codes. Fig. 5 shows the script that authenticate traffic source and redirect them to the Real Web server.

JavaScript pop-up on loading the homepage

```
<body onload = "myFunction">
<script type = "text/javascript">
function myFunction()
{
var answer = confirm ("Hello, Welcome to IPLab.")
if (answer)
window.location = "http://192.168.1.11/"
else
window.location = ""
}
</script>
// JavaScript Document
```

Fig. 5. JavaScript codes for redirection of authenitcated requests.

The above script can be intelligently modified into a more sophisticated manner to accept input in the form of voice by using speech recognition applications. For example speech API ID can be bound to the speechchange event. The purpose is to consider people who are not able to use JavaScript clickable pop-ups for various reasons such as sight problems, and visual impairement. These features might not be available in all browsers at the moment but work fine with Google Chrome browser. The conceptual script to perfom the function in pseudo codes is shown in Fig. 6 below.

Speech to text conceptual JavaScript pseudo codes

```
<script type = "text/javascript">

$(document).ready(function()
{
$("speechapi") bind ('speechchange event', function (e) )
{
var val = $(this).val();
alert('what is today's date?')
alert timeout = 3 seconds
val=document.prompt(speech2text)
var d = new Date();
var n = d.getDate();

if (val == +n)
{
window.location="http://----"
else
window.location=" "
}
</script>
```

Fig. 6. Conceptual JavaScript pseudo codes for manipulating the JavaScript pop-up by using speech to text API for traffic source authentication.

B. *Experimental Results*

In this section, we show performance of our mitigation approach in terms of response time to connect to the Web server. We show the Real Web server and the Bait Web server average response time in Table VI during an HTTP flood and GET attack without NIPS and with NIPS enabled. Secondly, we show received packets.

TABLE VI. AVERAGE RESPONSE TIMES FROM THE SERVERS DURING AN ATTACK

Bait server without NIPS enabled	Bait server with NIPS enabled	Web server on receiving authenticated traffic packets
01.615s	00.032s	00.025s

When our customized IPS is enabled, an average response time to connect to a Web server improves considerably. Since this simulation is on a small scale on a virtualized network, techinically in a very large and complex network the response time could be very high or infinite with a very high magnitude HTTP attack. The above results are based on Ubuntu Servers version 14.04.2 LTS with at least 10GB memory in a virtualized environment in the institution cloud. Web server average response time by using "*CURL*" Linux terminal command is 01.165s during an attack without NIPS enabled which is relatively longer than 00.032s when the NIPS is enabled. Real Web server has relatively shortest response time due to uninteruptible services. In Fig. 7, the graph depicts various connection response time captured randomly at a Bait and Real Web servers during an HTTP attack without NIPS enabled. Even though the system is under attack, the Real Web server is hardly troubled.

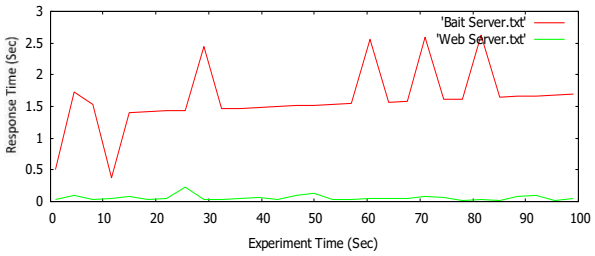

Fig. 7. HTTP Flood and GET DDoS traffic at a Bait Server without custom Snort NIPS.

Fig. 8 depicts the connection response time to a Bait Web server when our custom Snort NIPS is enabled. The response time is reduced considerably and give the server respite.

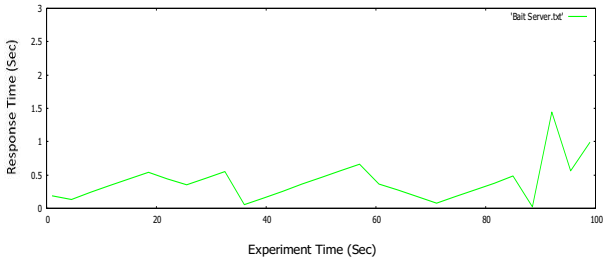

Fig. 8. Bait server response time gets respite with custom IPS enabled. More assurance for a Web server safety.

V. DISCUSSION

In this section we will discuss issues related to the JavaScript authentication, provisioning of datasets, traffic authentication for visually impaired users and scalability of anti-DDoS tool.

A. *Efficiency*

In our approach we use JavaScript pop-up as a way of authenticating traffic source. For users who disable JavaScript plugins for various reasons, alternative methods can be used. Alongside JavaScript pop-up, clickable images, hyperlinks or a banner to inform user to enable JavaScript plugins can be deployed. Even though JavaScript authentication method works efficiently, it can be prone to application layer DDoS attacks that use artificial intelligence. AI has ability of finding a way to beat the many currently deployed systems. We are still optimistic since there is currently no known application layer DDoS attack that is bypassing the JavaScript pop-ups.

Visually impaired users on the other hand may not favor the approach since it requires a user to see and click what is on the device screen. There may be considerations for such users by integrating this approach with assistive technologies such as quality voice recognition systems and sensors that performs

the same function as in the JavaScript pop-ups. In reference to Fig. 6, we have shown how the expected application would be like in pseudo code. The prompt can be made simple to use and user can be prompted for information input such as *word spell, birthdate and time of a day* with the help of screen reader.

B. Provisioning of Datasets

In our approach we obtain labeled datasets from various repositories for training a classifier using random tree machine- learning algorithm. Most of the data are current data from MAWILab and some are old data from NETRESEC where they are available in *pcap* file format. To read labels we convert *pcap* files into *csv* file format in Linux. These datasets have many labels but we filter them to suit our classifier and user's traffic. Users of the system interact with these rules at the IDS when they send their requests to the Webserver, however new attacks may not be detected by these rules because they may not match, as the rules get old. As far as machine learning is concerned, the classifier should periodically be trained with new data traffic in order to be able to match new attacking patterns.

C. Scalabilty.

In our approach we used few DDoS tools to analyze their behaviors so as to develop classification model. Since decision trees machine-learning algorithms are stable with huge amount of data, more tools can be analyzed and used to train its classifier. Behaviors of specific DDoS tools are most of the time consistent and hence make it easier to analyze them. Therefore this module for detecting and mitigating specific DDoS tool attacks can scale as much as possible depending on how many tools are experimented. In this paper we only used several free DDoS tools but in a large-scale experiment more tools can be included. One can include also some commercial DDoS tools such as *Fg Power DDOSER, GB DDoSeR, silentDDoSer, Drop-Dead DDoS, Hypocrite, Good Bye v3 etc*.

VI. CONCLUSION

Considering application layer DDoS attack traffic that hides behind legitimate traffic and false positives traffic that resulted from IDS misclassification, we proposed simple network architecture that makes use of Bait and Decoy Web servers to determine the legitimacy of normal traffic and malicious traffic. We trained classifier in random tree machine-learning algorithm using labeled datasets and generate rules that help us to fine tune Snort custom NIPS. We set Bait and Decoy Web server as recipients of legitimate and malicious traffic respectively before routing them to the Real Web server. Linux multi-IP enables our network to use same IP for Bait and Decoy Web server through different gateways.

The simulation results of a simple network topology show that the proposed approach can efficiently classifies traffic as malicious or legitimate. We use traffic authentication to assist our custom NIPS for the percentage of data that it cannot distinguish. The action on the JavaScript pop-up will determine where to route the traffic. The connection response times in the Fig. 7 and Fig. 8 prove efficiency of our approach.

ACKNOWLEDGMENT

I would like to sincerely express my gratitude to everyone who supported this research. I would like to acknowledge my internal guide, Professor A. Govardhan, my colleagues Sirikarn Pukkawana, Kazuya Okada and Damien Rompapas for their constant support towards this research.

REFERENCES

[1] B2B International in conjunction with Kaspersky Lab, Consumer Security Risks Survey , "Global IT Security Risk Survey - Distributed Denial of Service (DDos) Attacks Report", July 2014.

[2] Manthena, R. Application-layer Denial of Service, on http://forums.juniper.net/t5/Security-Now/Application-layer-Denial-of-Service/ba-p/103306, September 2011.

[3] Dimitar Kostadinov on Layer 7 DDoS Attacks: Detection and Mitigation, INFOSEC Institute on http://resources.infosecinstitute.com/layer-7-ddos-attacks-detection-mitigation/, December 2013.

[4] Ho, C. Y., Lin, Y. D., Lai, Y. C., Chen, I. W., Wang, F. Y., & Tai, W. H. (2012). False positives and negatives from real traffic with intrusion detection/prevention systems. *International Journal of Future Computer and Communication*, *1*(2).

[5] Prabha, S., & Anitha, R. (2010). Mitigation of Application Traffic DDOS Attacks with Trust and Am Based Hmm Models. *International Journal of Computer Applications IJCA*, *6*(9), 26-34.

[6] Xiang, Y., Li, K., & Zhou, W. (2011). Low-rate DDoS attacks detection and traceback by using new information metrics. *Information Forensics and Security, IEEE Transactions on*, *6*(2), 426-437.

[7] Mehra, M., Agarwal, M., Pawar, R., & Shah, D. (2011, February). Mitigating denial of service attack using CAPTCHA mechanism. In *Proceedings of the International Conference & Workshop on Emerging Trends in Technology* (pp. 284-287). ACM.

[8] Are you a robot? Introducing "No CAPTCHA reCAPTCHA" - on http://googleonlinesecurity.blogspot.jp/2014/12/are-you-robot-introducing-no-captcha.html.

[9] Miu, T. T., Lee, W. L., Chung, A. K., Luo, D. X., Hui, A. K., & Wong, J. W. Kill'em All--DDoS Protection Total Annihilation!.

[10] BBC News – "The evolution of those annoying online security tests" http://www.bbc.com/news/magazine-18367017 retrieved on 2015/03/28.

[11] Okada, K., Hazeyama, H., & Kadobayashi, Y. (2014, June). Oblivious ddos mitigation with locator/id separation protocol. In *Proceedings of The Ninth International Conference on Future Internet Technologies* (p. 8). ACM.

[12] Fontugne, R., Borgnat, P., Abry, P., & Fukuda, K. (2010, November). Mawilab: combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. In *Proceedings of the 6th International COnference*(p. 8). ACM.

[13] Network Forensics and Network Security Monitoring on www.netresec.com.

[14] Bait and Switch, v.2.0, September 2003 on https://sourceforge.net/project/showfiles.pho?group_id=64718.