Proceedings of the Thirtieth International Florida Artificial Intelligence Research Society Conference

A Text Mining Approach for Anomaly Detection in Application Layer DDoS Attacks

Maryam M. Najafabadi, Taghi M. Khoshgoftaar, Chad Calvert, Clifford Kemp
Florida Atlantic University
mmousaarabna2013@fau.edu, khoshgof@fau.edu, ccalver3@fau.edu, cliffkempfl@gmail.com

Abstract

Distributed Denial of Service (DDoS) attacks are a major threat to Internet security, with their use continuing to grow. Attackers are finding more sophisticated methods to attack servers. A lot of defense mechanisms have been proposed for DDoS attacks at IP and TCP layers. Those methods will not work well for application layer DDoS attacks that utilize legitimate application layer requests to overwhelm a web server. These attacks look legitimate in both packets and protocol characteristics, which makes them harder to detect. In this paper, we propose an anomaly detection method to detect application layer DDoS attacks. We take a text mining approach to extract features which represent a user's HTTP request sequence using bigrams. We apply the one class Support Vector Machine (SVM) algorithm on the extracted features from normal users' HTTP request sequences. The one class SVM labels any newly seen instance that deviates from the normal, trained model as an application layer DDoS instance. We apply our experimental analysis on real web server logs collected from a student resource website. Three different variants of HTTP GET flood attacks are implemented on our server, generated via penetration testing. Our results show that the proposed method is able to detect application layer DDoS attacks with very good performance results.

Introduction

Distributed Denial of Service (DDoS) attacks have consistently been among the major threats to Internet security in recent years. The main goal of DDoS attacks is to prevent legitimate users from using a service. This makes these attacks different from other classes of attacks in network security, which are focused on stealing or misusing user data. Usually, DDoS attacks exhaust a server's resources making these resources unavailable to server authorized users. The need for defending network servers and other systems, by detecting potential attacks, has become a serious topic in network security (Zargar, Joshi, and Tipper 2013).

In the past, it was more common to execute DDoS attacks that focus on the network and transport layers, such as ICMP flooding, SYN flooding or UDP flooding. These attacks intend to consume network bandwidth to overload the number of simultaneous connections a server can handle. A large number of mechanisms have been proposed in the literature for the detection of DDoS attacks; however, most of this research is focused on attacks at IP or TCP layers (Durcekova, Schwartz, and Shahmehri 2012). Compared to the amount of work that has been done on IP and TCP layers, only a few works involve the detection of DDoS attacks at the application layer.

The large number of detection mechanisms proposed for DDoS attacks at IP and TCP layers, has made it difficult for attackers to successfully launch DDoS attacks on these layers. This has caused attackers to switch to targeting the application layer in order to overload application servers. These attacks target the vulnerabilities in the application layer. Therefore, the connections at IP and TCP layers look normal making application layer DDoS attacks difficult to detect, due to their traffic being similar to normal traffic. On the other hand, compared to DDoS attacks at IP and TCP layers, the application layer attacks need fewer resources to be launched (McGregory 2013). The reason is that these attacks need less traffic to be performed. The goal is to exhaust the resources for a targeted service, which is always less than number of TCP and UDP connections needed to launch a transport layer DDoS attack. All these reasons have caused application layer DDoS attacks to become a very popular class of attack used against computer networks in recent years (M. Najafabadi et al. 2016b).

Since the application layer was not targeted often in past DDoS attacks, the application layer DDoS attacks are a relatively recent trend and there are only a limited number of research studies done for their detection. Among those is the work by Kandula (Kandula et al. 2005) which proposed a probabilistic authentication mechanism using CAPTCHA (acronym for Completely Automated Public Turing test to tell Computers and Humans Apart). It requires each client to solve a puzzle in order to authenticate him/herself before accessing the server. This method is not very effective, as it might annoy the users and also block web crawlers access to the site for indexing the content to be used by search engines.

There is a need for detecting application layer DDoS attacks on computer networks. Application layer DDoS attacks are particularly hard to detect because these attacks use legitimate application layer requests to exhaust server resources. These requests follow the standard networking protocols and look similar to normal users' requests. There are some studies on the differentiation between a normal user's behavior and an attacker's behavior by analysing web logs (Ranjan et al. 2009), (Wang, Yang, and Long 2011), (Liao et al. 2014). These studies follow two main strategies: anomaly detection and classification. In anomaly detection methods, the normal users' behavior is modeled. If a new behavior does not conform to this normal, or baseline, behavior model, it is marked as a potential attack. Anomaly detection methods are able to detect new or zero-day attacks. Training the model only requires normal data, which can be collected during the typical operation of a computer network. A potential downside of these methods is a high false alarm rate, because no attack data is used to train the model. The classification based studies, on the other hand, use both normal and attack data in order to build a classification model, which can distinguish between normal and attack instances. The issue with these studies is that they are not able to detect a new type of attack, because it was not a part of the model training dataset. In addition, in real world applications, it is hard to access data that includes real attack instances beforehand in order to train the model.

In this paper, we propose an anomaly detection method for detecting HTTP GET flood application layer DDoS attacks. A lot of application layer DDoS attacks target HTTP, in which they target a web server in order to exhaust its resources. The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers. HTTP works as a request-response protocol between a client and server. The web browser on a user's computer can be considered the client, and an application on a computer that hosts the website, which the user is browsing, is the server. Two commonly used methods for a request-response between a client and server are: GET and POST. The GET protocol requests data from a specified resource while the POST protocol submits data to a specified resource to be processed .

The attackers misuse the weaknesses in either HTTP GET or HTTP POST protocols. In HTTP GET attacks, the attacker floods the web server by sending a large number of HTTP GET requests. These packets have legitimate HTTP payloads, so the victim server cannot distinguish them from normal users' packets (Byers, Rubin, and Kormann 2004). Another type of misuse of the HTTP GET protocol is the Slowloris attack (Estevez-Tapiador, García-Teodoro, and Díaz-Verdejo 2005). In this attack, the attacker does not flood the server with spoofed requests, but it separates the lines of the HTTP headers and sends them at very slow rate to keep the server busy. Attackers also misuse weaknesses in HTTP POST protocol, which is similar to Slowloris attack in that the attacker tries to keep the server busy by sending pieces of data at very slow rate. The HTTP GET attack is different from Slowloris and HTTP POST attacks in that the attack has a flooding behaviour instead of sending pieces of information in a slow manner. In this paper, we focus on the HTTP GET attacks with flooding behavior.

In a HTTP GET flood attack, the attacker sends a large number of HTTP GET requests to a web server from different infected computers (bots). The server receives these requests and responds to them which causes its resources to become exhausted. It would be of benefit if a method existed which could differentiate between normal users' requests and attack requests, which enables the server to filter attack requests. Our approach addresses this need by proposing an anomaly detection method for detecting attack requests. We model the normal users' HTTP GET request sequence behavior and if a sequence of requests does not conform to the trained model, it is detected as a possible attack instance.

We focus our efforts on anomaly detection, rather than classification. Our main reason is that classification methods need labeled data. Since HTTP GET flood attacks can utilize numerous variants, building a labeled dataset that contains all these variants is not a feasible task. Attackers are capable of sending different HTTP requests to the web server in different orders. They can target their requests toward the main webpage, a random webpage, a particular resource such as an image file and even a combination of these. Considering that a website can contain a high number of webpages and resources, the number of possible HTTP GET Flood behaviors can get very high. It is not feasible to have some data which contains all variants of HTTP GET flood attacks by including an exhausted list of behaviors attackers can show while sending HTTP requests to the target server. However, it is possible, and also simpler, to collect normal data during normal server operations in order to have a representation of normal users' access behaviour. This data can be used to model users' behavior and detect anomalous behaviors (potential attacks).

This paper takes a novel approach to model a normal user's web access behaviour, by combining a text mining approach for feature extraction and an anomaly detection method using one class SVM. This normal behavior model is used to detect anomalous behaviors/potential attacks. We define each sequence of a user's HTTP GET requests as a document. The requested resources are considered as words/tokens. We extract features from each document using bigrams. A one class SVM is then trained on the extracted features to model normal users' behaviour, and is tested on the collected attack data to calculate the model's performance. We collect server logs from a student resources website. We generate three different types of HTTP GET flood attacks through penetration testing using 45 machines. Our results show that our approach in using one class SVM for detecting anomalies, along with the text mining method in extracting features, provides good performance results for detection of attacks with a small false positive rate.

The remainder of this paper is organized as follows. In Related Work Section, we discuss related work on the topic of the detection of HTTP GET flood attacks. The Methodology Section presents our approach for detecting these attacks. The Experimental data Section explains our data collection and the experimental data used. In Results Section, we discuss our results. Finally, in Conclusions Section, we conclude our work and provide suggestions for future research.

Related Work

Xie and Yu (Xie and Yu 2009) proposed an extended, hidden semi-Markov model to model the browsing behavior of web surfers. Markov state space is used to describe the webpage set of the website. The state transition probability matrix presents the hyperlink relationship between different webpages. When a user clicks a hyperlink pointing to a page, a number of HTTP requests are generated for the page and its in-line objects. To get an estimate of the order of pages a user is requesting, the requested objects in an observation sequence is grouped into different clusters. This allows each user request sequence to be transformed into the corresponding webpage. The order and transition sequence of consecutive groups show the user's browsing behavior. The entropy of an observed request sequence made by a user is defined as the anomaly measure. This model is very complicated to train and it is computationally heavy.

Ye et al. (Ye, Zheng, and She 2012) proposed clustering user sessions. They calculate the deviation between sessions and normal clusters as the abnormality measure. Four features are extracted from each session in order to cluster the normal users sessions to describe normal users behaviour. The attack data contains a fixed number of requests per second with random objects to the web server. Their generated attack characteristic is based on the normal data they have collected. The number of attack requests per second is calculated by multiplying the number of requests in a session randomly selected from normal data by two. This makes the attack characteristics not representative of a real attack that happens in computer networks.

Liao et al. (Liao, Li, and Kang 2015) used a classification method to classify attacks from normal data. They extracted two features to represent user's browsing behavior in one time window. The first feature describes the number of user requests in every sub-time window. The second feature describes the time intervals between user requests. The idea is that human visitors spend time on their interested webpages. First, the average interval for a sequence of user accesses during one hour is calculated and compared with a threshold value to filter users who are 100% normal. Both normal and attack data are used to define this threshold. Rythm matching is used on frequency sequences (first feature) to group the remaining sequences in clusters. The suspected attack clusters are defined as clusters, with which their scale is less than a threshold. This filters out the normal users. Each of the remaining suspected clusters are again clustered using a clustering algorithm with label (L-Kmeans). Their approach is dependent on defining threshold values in different steps to filter the data. In addition, they did not provide results on test datasets, the results are provided on the training datasets only. Providing the results on the test data is important as it verifies whether the parameter values, which are selected solely based on the trained data, provide the same performance results on the test data as well.

Wang et al. (Wang, Yang, and Long 2011) presented two different methods to characterize users' web access behavior in order to detect application layer DDoS attacks. To characterize web access behavior with webpage ratios, they construct the website's priori click rate on vector which represents the click ratio for each webpage in the website. Each user session is defined as its subsequent webpage requests in 30-minute time interval and each user has its empirical click ratio vector which shows user's interest in different webpages during a session. The second method builds the transition probability matrix between webpages. Again, each user's empirical access behavior matrix is calculated depicting user's access logic on the website. They compare users empirical click ratio vector to the websites priori one, and adopts large deviation theory estimating the probability of the deviation. They do the same to measure the deviation of ongoing user's behavior from website's priori transition probability matrix. Their simulation results show that the first approach can detect application layer DDoS attacks accurately, while the second approach has high false negatives. Clicking on a webpage produces several HTTP requests, in order to apply this method, it should be clear how the user's webpage access can be derived from the users' HTTP requests. Also, their first approach might not provide good detection results if the attacker targets only highly accessed webpages on a website.

Methodology

In HTTP GET flood attacks, a large number of infected computers (bots) send HTTP GET requests to a targeted HTTP server. Since the server cannot distinguish between normal and attack HTTP GET requests, it responds back to all the received GET requests. Each server response consumes a percentage of the server resources. The large number of GET requests eventually exhausts the server resources and the server is not able to service legitimate users. A normal user behaviour in accessing resources on a web server is based on the structure of the website and the webpages the user is browsing. The web mining studies have shown that about 10% of webpages on a website may draw 90% of the attention. In addition, web user browsing behavior can be profiled by user's webpage request sequences (Kantardzic 2011). Thus, we can model the normal user's behaviour in accessing the website webpages. On the other hand, each click on a webpage makes the browser send a number of HTTP GET requests to the web server which includes the webpage and its in-line objects, such as images, flash, video and audio. This means we can also model the sequence of resources a normal user is accessing on a web server. We can then use such models for detecting any anomalous sequences of resource usage on the web server.

We use server logs to extract the sequence of resources each user is requesting on the server side. In the following subsections, we explain how we combined feature extraction using text mining (bigrams) and one class SVM for anomaly detection, in order to model the access behavior of normal users and detect anomalous events (possible attacks).

Feature Extraction Using Bigrams

Each web server provides users with different resources. These resources include webpages, images, configuration files, etc. The web server collects a log that records requests the server has received and some operational

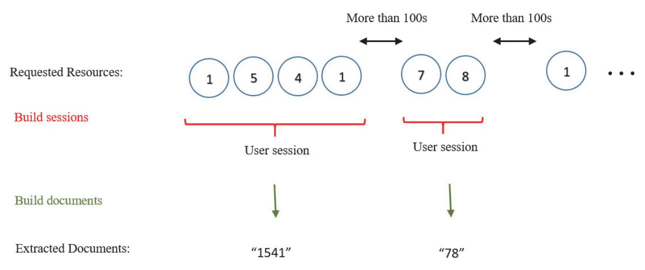

Figure 1: Extracting documents from the sequence of resources requested by a particular user

information[1]. A typical record in a web server log looks similar to the following: "222.222.222.222", "57890", "837074" , "/var/www/html/wordpress/index.php", "HTTP/1.1" "GET", "/wordpress/index.php", "Mozilla/5.0 (compatible; Baiduspider/2.0; +http://www.baidu.com/search/spider.html)." "222.222.222.222" is the client IP address of the request. "57890" is the size of the response in bytes and "837074" is the time that took the server to serve this request. "HTTP/1.1" is the request protocol and "GET" is the request method. "/wordpress/index.php" shows what URL path is requested by the client and finally, "Mozilla/5.0 (compatible; Baiduspider/2.0; +http://www.baidu.com/search/spider.html)" shows the browser from which the request has been sent, known as the user agent.

We use a text mining approach to extract features from the server logs. In order to apply a text mining approach, we need to define a corpus of documents where each document contains tokens. We consider each user session as a document and each requested resource in the user session as a token. By extracting all the user sessions from the server log file, we build our document corpus. We define each user session as the sequence of resources a particular user requests, where the time between requests is less than 100 seconds (we selected this value based on our preliminary analysis). Figure 1 shows how documents are built from user sessions.

After documents are extracted from the server log file, we then create bigrams to extract features from each document. Bigrams are a particular form of ngrams. An Ngram is a contiguous sequence of *n* items from a given sequence of text or speech. Bigrams are a sequence of two adjacent elements from a string of tokens. In our analysis, each token is actually a HTTP request made on the web server which is the URL field in the log file above.

In order to build a model, we need to convert each document into a fixed size feature vector. First, we determine the most frequently occurring bigrams in our entire corpus. We count the number of times each bigram is repeated in the entire corpus. We then calculate the quintiles of all the bigrams frequency counts. Since in our analysis, the third quintile was 5, we decided to consider all the bigrams with counts more than 5 as the frequent bigrams. These frequent bigrams build our features. We also consider one additional feature, called NF, for all the non-frequent bigrams. To build the feature vector for each document, we checked how many times each of the frequent bigrams are repeated in it. The number of all the non-frequent bigrams happened in the document are added up to fill the value for the NF feature.

[1]https://httpd.apache.org/docs/1.3/logs.html

One Class SVM Anomaly Detection

One class Support Vector Machine (SVM) (Schölkopf et al. 2001) can be used for the task of anomaly detection (Heller et al. 2003). Support vector machine is a discriminative classifier. Given labeled training data (supervised learning) of two different classes, the algorithm finds an optimal hyperplane which categorizes new instances to either of the learnt classes. Schölkopf (Schölkopf et al. 2001) extended the SVM methodology to handle training using only one class of data. By just providing the normal training data, the algorithm creates a (representational) model of this data. If newly encountered data is too different from the trained model, it is labeled as out-of-class, i.e anomaly.

We use one class SVM to detect sessions that belong to HTTP GET Flood attacks. We use the feature vectors from normal users' sessions to train the one class SVM. This learns how the sequence of a normal user's requests should look like. Each newly seen session then is provided to this trained model and, if it is detected as an out-class, we labeled it as a HTTP GET flood session.

Experimental data

Our collection efforts were performed on an active, full-scale campus network which have been used in our previous data collections as well (Calvert et al. 2017; M. Najafabadi et al. 2016a; Zuech et al. 2015). A web server was setup within our network to host a publicly accessible student resource website. The server itself runs CentOS and uses Apache to host our student resource website, which was developed using Wordpress. This resource site is utilized by students and faculty to access course information such as lecture notes, lab materials, syllabi, assignment schedules, and course announcements. All traffic directed to this server is captured continuously and stored in a single incremental access log.

Our attack implementations focus on variants of the HTTP GET flood DDoS attack, which is a common method of enacting application layer DDoS attacks. As previously explained, this attack exploits web servers or applications by continuously sending what seem to be legitimate HTTP GET requests in order to exhaust server resources. In our experiments, we have implemented and collected traffic from three different HTTP GET flood variants. These variants are described as follows:

Single Page Single Page HTTP GET flood attacks target a specific page from a website, typically a page that is visited frequently by users. For our attack, we targeted the home page of our student resource site, as this is frequented by all users and is also a common target for many real-world attacks.

Random Page This variant targets a random page out of all possible pages accessible within the website. Page selection is not impacted by page popularity. Also, pages do not need to be accessible from one another, meaning that a link does not have to lead from one previously selected page to the next.

Top Five This variant continuously requests the top five most visited pages from the resource site. For our purposes, page popularity was tracked using a separate Wordpress plug-in which ranked page popularity is based upon visits by unique users.

Our penetration testing involved the use of customized Python (G. van 1995) scripts, created for each of the aforementioned attack variants. These scripts were utilized with the intent of exploiting our web server's HTTP GET protocol to consume resources. Our scripts utilize the socket interface to connect to our designated host (web server) using a designated port. Once connected, our scripts send a request for the desired resource from the web server. Once the request is sent, the connection is closed and the process repeats continuously to instigate the flood.

For each attack session, 45 host machines within our network were configured to launch the attack simultaneously based on our written scripts. Only one attack variant was implemented at a time and all hosts were configured to run the same attack. Each attack session was enacted for a full hour and, for all attack variants, each HTTP GET request was made using a random interval of 1 to 5 seconds. As with our normal traffic, attack traffic was also collected and stored in our continuous web logs. For easy identification of traffic, all attack traffic is noted as originating from the same subnet, as all attacks were performed on local hosts. All other normal traffic existing in the log consist of outside, public IPs which do not conflict with our subnet.

Results

Since we want to detect HTTP GET flood attacks, we only included GET data in our analysis. We divided our data into 3 sections, which includes normal data divided in two sections and the attack data. We randomly selected 70% of the normal samples as training data. The remaining 30% of the normal data, plus the whole attack data, are used for testing. We did all the implementations in R (R Core Team 2016). We used the "e1071" (Meyer et al. 2015) library for the one class SVM and tm (Feinerer and Hornik 2015) and Rweka (Hornik, Buchta, and Zeileis 2009) packages for extracting bigrams. We used Radial Basis Function (RBF) kernel and the ν and gamma parameters were chosen by using grid-search method (Hsu et al. 2003).

Since in our application, the attack sessions are the class of interest, we consider the attack class as the Positive class and the normal class as the Negative class in presenting our performance results. After the one class SVM is trained, we applied the trained model on the test data, which includes both normal and attack instances. During the test, if an instance gets an out-class label, it means the instance does not belong to the same class as training data. In our experiments, training data is the Negative class, i.e. normal data, and an out-class instance means a Positive class, i.e. attack instance.

By considering the attack class as the Positive class, True Positive Rate (TPR) is defined as the percentage of attack instances which are correctly labeled as attack and False Positive Rate is the percentage of normal instances which are wrongly labeled as attack data. In an anomaly detection application, we want TPR to be high and FPR to be low. Testing the trained one class SVM on the test data provided 100% TPR and 2% FPR. This result shows that the proposed methodology provides very good performance results in detection of three different types of HTTP GET flood attacks. In future work, we decide to analyze the false positive instances in more detail in order to include additional analysis which can reduce the FPR even more.

Compared to approaches such as (Xie and Yu 2009) and (Wang, Yang, and Long 2011), which model how the users access different webpages on a website, our approach models how the users access the resources on a website. This eliminates the need to extract which webpage the user is accessing by examining the sequence of the requests. This can be a complicated task considering webpages can share different resources. In addition, those approaches do not fit attacks where the attacker is targeting resources on a website instead of the webpages.

Conclusions

The large number of studies done on the detection of DDoS attacks on IP and TCP layers necessitates a switch to application layer methods to launch DDoS attacks in recent years. Therefore, in comparison to the IP and TCP layer DDoS attacks, application layer DDOS attacks can be considered a more recent trend in network attacks that needs to be detected in order to make sure legitimate users receive their requested services. These attacks do not show any abnormal behavior in packets or protocol, which make them harder to detect. In this paper, we proposed an anomaly detection mechanism for the detection of HTTP GET flood DDoS attacks. We collected web server logs from a student resource website. We also generated three different types of HTTP GET flood attacks through penetration testing. We extracted features from users request sequences by using bigrams from text mining. We then applied a one class SVM to model the normal users behavior in sending HTTP GET requests to the server. Any newly seen instance, which is labeled as out-class by the trained one class SVM, would be detected as an HTTP GET attack instance. Our experiments show that the proposed method provides very good performance results to distinguish between normal users instances and the HTTP GET flood instances. For future work, we plan to collect more normal data as well as more variants of attack data in order to expand our analysis.

Acknowledgment: We acknowledge partial support by the NSF (CNS-1427536). Opinions, findings, conclusions, or recommendations in this material are the authors' and do not reflect the views of the NSF.

References

Byers, S.; Rubin, A. D.; and Kormann, D. 2004. Defending against an internet-based attack on the physical world. *ACM Transactions on Internet Technology (TOIT)* 4(3):239–254.

Calvert, C.; Khoshgoftaar, T. M.; Najafabadi, M. M.; and Kemp, C. 2017. A procedure for collecting and labeling man-in-the-middle attack traffic. *International Journal of Reliability, Quality and Safety Engineering* 24(1):19 pages.

Durcekova, V.; Schwartz, L.; and Shahmehri, N. 2012. Sophisticated denial of service attacks aimed at application layer. In *ELEKTRO, 2012*, 55–60. IEEE.

Estevez-Tapiador, J. M.; García-Teodoro, P.; and Díaz-Verdejo, J. E. 2005. Detection of web-based attacks through markovian protocol parsing. In *10th IEEE Symposium on Computers and Communications (ISCC'05)*, 457–462. IEEE.

Feinerer, I., and Hornik, K. 2015. *tm: Text Mining Package*. R package version 0.6-2.

G. van, R. 1995. Python tutorial, Technical Report CS-R9526. Technical report.

Heller, K. A.; Svore, K. M.; Keromytis, A. D.; and Stolfo, S. J. 2003. One class support vector machines for detecting anomalous windows registry accesses. In *Proc. of the workshop on Data Mining for Computer Security*, volume 9.

Hornik, K.; Buchta, C.; and Zeileis, A. 2009. Open-source machine learning: R meets Weka. *Computational Statistics* 24(2):225–232.

Hsu, C.-W.; Chang, C.-C.; Lin, C.-J.; et al. 2003. A practical guide to support vector classification.

Kandula, S.; Katabi, D.; Jacob, M.; and Berger, A. 2005. Botz-4-sale: Surviving organized ddos attacks that mimic flash crowds. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, 287–300. USENIX Association.

Kantardzic, M. 2011. *Data mining: concepts, models, methods, and algorithms*. John Wiley & Sons.

Liao, Q.; Li, H.; Kang, S.; and Liu, C. 2014. Feature extraction and construction of application layer ddos attack based on user behavior. In *Control Conference (CCC), 2014 33rd Chinese*, 5492–5497. IEEE.

Liao, Q.; Li, H.; and Kang, S. 2015. Application layer ddos attack detection using cluster with label based on sparse vector decomposition and rhythm matching. *Security and Communication Networks* 8(17):3111–3120.

M. Najafabadi, M.; Calvert, C.; M. Khoshgoftaar, T.; and Kemp, C. 2016a. Detecting man in the middle traffic using packet header information. In *22nd ISSAT International Conference on Reliability and Quality in Design*, 197–201.

M. Najafabadi, M.; M. Khoshgoftaar, T.; Napolitano, A.; and Wheelus, C. 2016b. Rudy attack: Detection at the network level and its important features. In *The Twenty-Ninth International Flairs Conference, Special Track on Artificial Intelligence and Cyber Security*, 282–287. AAAI.

McGregory, S. 2013. Preparing for the next ddos attack. *Network Security* 2013(5):5–6.

Meyer, D.; Dimitriadou, E.; Hornik, K.; Weingessel, A.; and Leisch, F. 2015. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*. R package version 1.6-7.

R Core Team. 2016. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Ranjan, S.; Swaminathan, R.; Uysal, M.; Nucci, A.; and Knightly, E. 2009. Ddos-shield: Ddos-resilient scheduling to counter application layer attacks. *IEEE/ACM Transactions on Networking (TON)* 17(1):26–39.

Schölkopf, B.; Platt, J. C.; Shawe-Taylor, J.; Smola, A. J.; and Williamson, R. C. 2001. Estimating the support of a high-dimensional distribution. *Neural computation* 13(7):1443–1471.

Wang, J.; Yang, X.; and Long, K. 2011. Web ddos detection schemes based on measuring user's access behavior with large deviation. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, 1–5. IEEE.

Xie, Y., and Yu, S.-Z. 2009. A large-scale hidden semi-markov model for anomaly detection on user browsing behaviors. *IEEE/ACM Transactions on Networking (TON)* 17(1):54–65.

Ye, C.; Zheng, K.; and She, C. 2012. Application layer ddos detection using clustering analysis. In *Computer Science and Network Technology (ICCSNT), 2012 2nd International Conference on*, 1038–1041. IEEE.

Zargar, S. T.; Joshi, J.; and Tipper, D. 2013. A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. *IEEE Communications Surveys & Tutorials* 15(4):2046–2069.

Zuech, R.; M. Khoshgoftaar, T.; Seliya, N.; M. Najafabadi, M.; and Kemp, C. 2015. A new intrusion detection benchmarking system. In *The Twenty-Eighth International Flairs Conference*, 252–256. IAAA.