2021 IEEE 22nd International Conference on Information Reuse and Integration for Data Science (IRI) | 978-1-6654-3875-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/IRI51335.2021.00030

Detecting Slow Application-Layer DoS Attacks With PCA

Clifford Kemp, Chad Calvert, Taghi M. Khoshgoftaar
Email: {cliffkempfl@gmail.com; ccalver3@fau.edu; khoshgof@fau.edu;}
Florida Atlantic University, Boca Raton, FL 33431

***Abstract*—Countering Denial of Service (DoS) attacks is becoming ever more challenging with the vast resources and techniques increasingly available to attackers. One of these techniques is application-layer DoS. Due to these challenges, network security has become increasingly more challenging to ensure. Hypertext Transfer Protocol (HTTP), Hypertext Transfer Protocol Secure (HTTPS), Domain Name System (DNS), Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP), Voice over Internet Protocol (VoIP), and other protocol applications have seen increased attacks over the past several years. It is common for application-layer attacks to concentrate on these protocols because of some weaknesses that attackers can exploit. With some leveraging, application-layer attacks can vary, such as flood and "low and slow" attacks. Low and slow approaches are particularly well-known, mainly targeting weaknesses in the HTTP protocol, which is the most broadly used application-layer protocol on the Internet. Our paper aims to develop a generalized detection approach to identify features for application-layer DoS attacks that is not specific to a single slow DoS attack. We combine four application-layer DoS attack datasets: Slow Read, POST, Slowloris, and Apache Range Header to produce a single dataset for multiple attacks. We performed a feature extraction technique Principal Component Analysis (PCA), with the single dataset to reduce dimensionality. PCA transforms our existing dataset onto a new feature space to identify four separate application-layer attacks. We explore this method to improve six machine learners used in our work. Experimental results show that the machine learners successfully identified the multiple Slow DoS attacks with high detection while minimizing false alarm rates. The experiment demonstrates that when machine learners use Netflow features and feature extraction methods, they can discriminate and detect such attacks.**

Keywords: Application Layer DoS Attacks, DoS Detection, Machine Learning, Intrusion Detection, Netflow, Principal Component Analysis

I. Introduction

Network cyber-attacks have become commonplace in today's world. These attacks have become very sophisticated and challenging to prevent. Many stealthier attacks target the application layer where they take advantage of vulnerabilities on web servers. Because web servers are open to the public, they are accessed frequently by many users. The goal of attackers is to simulate legitimate, normal traffic as close as possible, which they do efficiently. The task for those defending the networks is to determine the difference between normal and attack traffic. To make it even more of a challenge, the attackers are constantly updating their attack methods.

Additionally, there are techniques used to enhance the data before the machine learning algorithm is applied. Usually, after collecting the data, the next step in the machine learning process is selecting the most relevant attributes, commonly referred to as features, to improve the machine learner's performance. It is essential to keep in mind that a set of features that perform well with one machine learner may not work well for other machine learners. Discovering the correct set of features for machine learning is referred to as feature selection. The goal of feature selection is to determine features that will produce the best accuracy and predictability for the machine learner while reducing feature dimensionality [1].

Denial of Service (DoS) is an attack that aims to prevent normal communication with a resource by disabling that resource itself or an infrastructure device providing connectivity to it. There are distinct types of DoS attacks, each performing at the various levels of the Open Systems Interconnection (OSI) model, as seen in Figure 1. The OSI model helps vendors create interoperable network devices and software in the form of protocols so that different vendor networks could work with each other. The OSI model's central concept is that the process of communication between two endpoints in a network is divided into seven separate groups of related functions or layers [2]. The application layer supplies network services to user applications. Network services are protocols that work with user data. For example, a web browser application uses the application layer protocol HTTP to package the data that can be sent and receive web page content. There is a current internet draft for HTTP by the Internet Engineering Task Force (IETF) to improve the standard [3].

In recent years, there has been a rise in DoS and Distributed Denial of Service (DDoS) attacks targeting application protocols, such as HTTP HTTPS, DNS, SMTP, FTP, VOIP, and other applications protocols. It is common for application-layer DoS attacks to concentrate on these protocols because of some weaknesses, such as not limiting the connection duration for data flows on a web server that attackers can exploit. Much like attacks targeting network resources, attacks targeting application resources come in various types, including HTTP GET, Slow POST, Slow Read, and Apache Header attack. Of these varieties, "low and slow" [4] approaches are particularly prominent, mostly targeting weaknesses in the HTTP protocol, which, as mentioned previously for attackers. A low and slow attack relies on a small stream of prolonged traffic targeting application or server resources. Unlike more traditional brute-force attacks, low and slow attacks require very little bandwidth. They can be hard to mitigate, as they

978-1-6654-3875-9/21/$31.00 ©2021 IEEE
DOI 10.1109/IRI51335.2021.00030

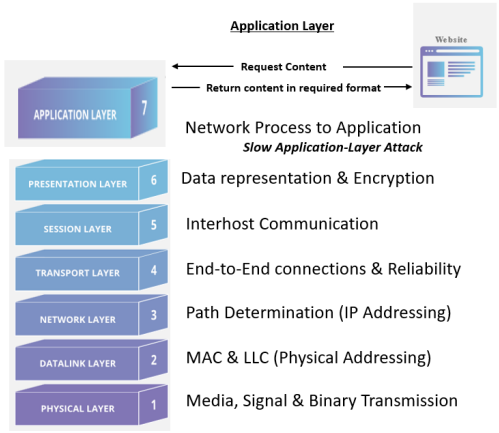

Fig. 1: OSI and Layer 7

generate challenging traffic to distinguish from normal traffic. Attackers apply various evasion techniques to bypass intrusion detection systems, leaving the network vulnerable to specific DoS attacks. A stealthy DoS attack variant can disrupt routine web services covertly without triggering any alerts. One potential solution is the application of flow-based (Netflow) analysis.

Netflow, also referred to as session data, represents a high-level summary of network conversations. A Network flow record is identified based upon the standard 5-tuple attribute set that makes up a conversation: source IP, destination IP, source port, destination port, and transport protocol [5]. System for Internet-Level Knowledge (SiLK) is a software tool suite to generate and analyze Netflow session data. Netflow is a more space-efficient format than a Full Packet Capture (FPC), not so much because of its size, but because it records the packets into service-specific binary flat files and can parse flows in a timely and efficient way without the need for complicated CPU intensive scripts. Netflow is a crucial factor when considering RAM and hard drive requirements for servers. Additionally, the well-designed features are utilized in Internet Protocol Flow Information Export (IPFIX) Netflow to avoid attackers' evasion techniques [6].

There are two unique contributions we provide through our experiment. The first is the analysis of multiple Slow HTTP DoS attacks with IPFIX Netflow datasets and machine learning predictive models that help detect those attacks. Netflow session data and machine learning build predictive models to detect evasive slow application-layer DoS attacks. We take advantage of user-defined data types in its features because of the IPFIX protocol being freely extensible and adaptable to different scenarios. We apply machine learning to Netflow features with the following six machine learning algorithms. Random Forest (RF), two variants of decision trees (C4.5D, C4.5N), 5-Nearest Neighbors (5NN), JRip, which uses Repeated Incremental Pruning to Produce Error Reduction (RIPPER), and Naïve Bayes (NB). We have chosen these machine learners based on our previous work with network traffic and the variations they represent [7] [8]. These six learners provide us with a comprehensive array of algorithms to use on our Netflow features.

Our second contribution focuses on the integrity of our data. In our work, we capture our data from a web server in a real-world network interacting with students, faculty, and the public on our web server located on a college campus. This network environment helps to produce results more representative of a real-world scenario. All attack data was created on a live and currently used platform. Adjusting variables in the Slow Read attacks produced three different levels of concurrent connections to give us a broader scope of this type of attack. Therefore, the live setting produced quality data that reflects an actual network environment.

The rest of the paper is structured as follows. In Section II, we detail slow application-layer attack methods, tools, and data types. In Section III, we discuss related works associated with collecting and detecting multiple slow application-layer HTTP attacks. Section IV outlines our collection procedure and our experimental designs. Section V provides the results of our experiment. Finally, in Section VI, we conclude the paper.

II. Background

There are a variety of methods for enacting an application-layer DDoS attack. Contingent on the network's characteristics, various types of attacks are chosen based on the targeted network. In this section, we detail the slow application-layer attacks, data collection process, and Netflow traffic.

The distributed nature of DDoS attacks makes them extremely difficult to trace. Attackers will spoof (fake) IP addresses to hide their identity, making the tracking of DDoS attacks challenging. There are security vulnerabilities in many Internet hosts that intruders can exploit. Moreover, incidents of attacks that target the application layer are increasing rapidly. It is vital to be aware of all aspects of DDoS attacks when deploying a comprehensive DDoS defense mechanism. Various classifications of DDoS attacks have been proposed in the literature over the past decade.

A. Application-Layer Attack Methods

HTTP flood attacks are the most common DDoS attack targeting application sources. Session-based sets of HTTP GET or POST attacks mimic legitimate requests sent to a victim's Web server, making it hard to detect. HTTP flood attacks are injected concurrently from multiple machines. Bots continuously request to download the target site's pages (HTTP GET flood), exhausting application resources and resulting in a DoS [9].

Slow HTTP GET request attacks are used to control all or most of an application's resources using open connections, blocking it from delivering service to users that need it for opening genuine connections. An attacker creates and provides incomplete HTTP GET requests back to the server with an attack like this. The attack will open separate threads for

each of these connection requests. The server will wait for the remaining packets to be sent. HTTP header data is still being sent by the attacker at gradual periods, ensuring the connection stays open and does not time out. The needed data comes at a slow rate.

A more recent attack method utilizes QUIC (Quick UDP Internet Connections), a transport layer network protocol, combined with the latest security protocol version, Transport Layer Security (TLS). The QUIC flood DDoS attack denies service by exhausting the resources of a targeted server with data sent over QUIC. The server processes all the QUIC data it receives from the attacker, slowing service to legitimate users or crashes the server altogether.

Another attack implementation is a Slowloris attack [10] [11]. It is also based on HTTP GET request weakness, but the victim is not flooded with spoofed requests. This technique instead uses time-delayed HTTP GET headers. In principle, the attacker does not send HTTP GET request headers simultaneously, but they separate the header's lines and send each line separately. The web server creates connections with the attacker, waiting until the end of the request header, and the connection is reserved for an extended period.

Application-layer DoS attacks, such as Slowloris, Slow HTTP POST, Slow Read, and Apache Range Header attack drain concurrent connection pools causing significant memory and CPU usage on the server. HTTP Apache servers are vulnerable to these attacks because they are thread-based.

Slow HTTP POST attacks can be performed by issuing a lot of concurrent POST requests, and each of them will send the POST body at a slow rate[7]. The malicious attacker will send HTTP POST requests via forms located on the webserver. Instead of the POST requests being sent in a typical manner, byte-by-byte communication is used. The attacker ensures that their vulnerable connection stays open, sending routine packets at slow regular intervals for each new byte of POST data. The HTTP POST request's content length is maintained and recognized by the server as a legitimate communication, producing the only option to wait for the complete POST request to be received.

In previous studies, we explored the use of machine learners combined with Netflow for detecting Slow Read DDoS attacks on web servers at the application layer[8]. Slow Read attacks send legitimate HTTP requests and read the response slowly, aiming to keep as many connections active as possible to tie up resources on the server until it cannot handle any further requests. The attacker controls the connection open by receiving the response from the server slowly using a minimal TCP window size. By advertising a zero receive window and acknowledging probes, a malicious receiver can cause a sender to consume resources (TCP state, buffers, and application memory), preventing the targeted service or system from handling legitimate connections.

The Apache Range Header attack exploits the byte-range filter in the Apache HTTP Server 1.3.x, 2.0.x through 2.0.64, and 2.2.x through 2.2.19, allowing remote attackers to execute a DoS attack (memory and CPU consumption) via a Range header that expresses multiple overlapping ranges [12]. A remote attacker can slow down a service or server to a crawl or exhaust memory, causing it unable to serve legitimate clients on time. The result of this vulnerability produces the result of a DoS. The server is unable to service any requests and refuses any additional connections. Using the SlowHTTPTest tool, a simple command generates a HEAD request with header Range: 0-, x-1, x-2, x-3, x-y where x is set by -a argument, y is set by -b argument and increments in the step of 1 byte. The test can be performed with various connection rates and numbers over SSL.

B. Attack Tools

This subsection gives a brief overview of the primary attack tools used to execute application-layer attacks while staying under the radar. The development of specific attack tools focuses on performing attacks more efficiently and quickly.

Two tools used for flooding networks are Low Orbit Ion Cannon (LOIC) [13] and High Orbit Ion Cannon (HOIC) [14]. These DoS attack tools used today offer an easy-to-use means of performing multi-threaded HTTP flood attacks. LOIC can produce a substantial volume of TCP, UDP, or HTTP traffic that can overload a server and bring down a network. As previously mentioned, HOIC utilizes a cross-platform script that inserts HTTP POST and GET requests with an easy-to-use graphical user interface (GUI). The use of booster scripts allows a user to specify lists of target URLs and identifying information for HOIC to cycle through as it generates its attack traffic. That, in turn, makes HOIC attacks slightly harder to block. HTTP headers are sent slowly in tiny chunks to the targeted server.

RUDY (R U Dead Yet) [15] is a tool similar to Slowloris. RUDY performs a DoS attack using a long-form field for HTTP POST submissions rather than HTTP headers. By injecting one byte of information into an application POST field at a time, a considerable backlog of application threads. The long "Content-Length" field prevents the server from closing the connection. An attacker can implement many concurrent connections to the server, ultimately exhausting the server's connection table and causing a DoS.

The SlowHTTPTest tool can execute the previously mentioned attack methods [16]. It efficiently implements various application layer DoS attacks such as Slowloris, Slow HTTP POST, Slow Read, and Apache Range Header. A slow application-layer attack exploits the fact that most modern web servers are not limiting the connection duration if a data flow occurs. It is the concurrent connections that eventually bring the server down.

III. Related Works

Zargar et al. [17] classify various forms of application layer DDoS flooding. Reflection amplification-based flooding attacks use the same techniques as their network/transport-level peers (i.e., sending forged application-level protocol requests to many reflectors). HTTP flooding attacks use session

connection request rates from the attackers that are higher than the legitimate users' requests; hence, this exhausts the server resources and leads to DDoS flooding attacks on the server. One of the famous attacks in this category is the HTTP GET/POST-flooding attack.

Related works with HTTP GET Requests include Yadav et al. [18], whose attack dataset has been created by performing an attack on the same website, nitt.edu, in a testbed environment. A web server was designed for the nitt.edu website and used for offline browsing. Then around 100 PCs were connected to the webserver through a switch. For performing an attack, two bots used Java LOIC and Golden Eye Master. All incoming traffic towards the webserver was captured using traffic capture software. The captured traffic comprised three different attack types. These contributions were much appreciated, but improvements still need to be made by collecting attack and normal traffic in a live environment instead of a testbed.

Durcekova et al. [19] discuss using HTTP GET and POST to exploit HTTP protocol's weakness. In this type of attack, attackers send many malicious HTTP GET requests to a target server. Since these packets have legitimate HTTP payloads, victim servers cannot distinguish normal HTTP GET requests from malicious requests. Thus, servers must serve all requests as normal requests, and they exhaust their resources finally.

Devi et al. [20] proposed a scheme to defend against DDoS attacks in the application-layer. Their main goal is to drop the suspicious traffic and to provide services to legitimate users with HTTP GET requests and pulling large files from the victim server in overwhelming numbers.

Dantas et al. [23] employ two different DDoS attacks in the computational tool Maude. Maude is a logic-based tool supporting formal specification and analysis of real-time systems. The two attacks analyzed are HTTP PRAGMA and the HTTP POST. HTTP PRAGMA attacks exploit an HTTP field called PRAGMA, a header field intended for use in HTTP protocol requests. PRAGMA sends messages to the application and any intermediate caches requesting that it wants a new version of a previously requested resource. The application receives PRAGMA messages to reset timeouts allowing the connection to continue.

IV. Experimental Procedure

The outline of our experiment concerning slow application-layer attacks in this section is divided into four subsections that include the data collection process, classification algorithms, metrics used to evaluate each model, and feature extraction with PCA. Our network topology and normal network data are defined as well.

A. Data Collection Process

Collecting Netflow data is based on network monitoring using a combination of hardware and software that are used to generate, organize, and store data for detection and analysis. Figure 2 shows our architecture in more detail.

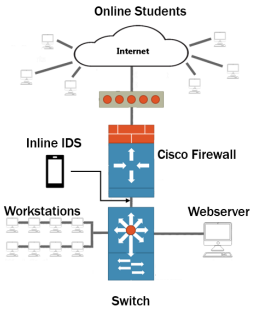

Fig. 2: Network Architecture

As in previous work, our capture framework allows us to perform our attacks within a real-world network environment servicing numerous active users [21] [8] [22]. The campus network consists of hosts from classrooms, labs, and offices. An Apache webserver serves as the target for our attacks as well as network resources. The server's configuration is composed of a Linux CentOS operating system, an Intel 3.30GHz processor, and 32 GB of memory. A Linux-based server that captures packets and writes them to disk is Dumpcap [23], which is part of the Wireshark software suite. It applies the Libpcap packet capture library to capture packets and write them in PCAP-NG format. Dumpcap also captures packets on a 24-hour basis and saves the full packet data to a directory with daily logs.

Before labeling our data records, it was necessary to perform a data-cleansing process. We serve four separate attacks (POST, Slowloris, Slow Read, Apache Range Header) and variations ranging from 5 - 60 minute time frames. As our work focuses on collecting DDoS and DoS attacks on a web server, all web server traffic is extracted from the raw packets collected. Labeling our full packet captures for attacks is a relatively simple process because the attacks originate from machines with known IP addresses. The attackers' IP address labels our attack traffic, and all other instances are labeled normal. This labeling process assures an accurate attack labeling by including controlled attacks. Since this data generated by network equipment, missing values were not present. Once the data is cleansed, it is imported into Wireshark and then exported as a CSV file.

Our Netflow data is collected and stored with the SiLK software suite [24]. SiLk collects and records Netflow data directly or by extracting them from its FPC data. SiLK can take any existing FPC file and convert it to a SiLK network flow record. The extracted data is then written to files as a SiLK record through a command-line interface and exported as a CSV dataset. One additional benefit of Netflow data is that its small size allows it to be retained for a much longer time, which is incredibly valuable when performing an analysis [6].

TABLE I: Slow Application-Layer Feature Set

Feature Name	Description
Protocol	IP protocol
Packets	Number of packets in flow
Bytes	Number of bytes in flow
Flags	TCP flags all packets [FSRPAUEC]
InitialFlags	TCP flags in initial packet
SessionFlags	TCP flags second through final packet
Durmsec	Duration of the flow (in seconds)
PayloadInBytes	Size of payload measured in bytes
PayloadRate	Non-overhead packet data per second
PacketsPerSecond	Number of packets per second
BytesPerSecond	Number of bytes per second
BytesPerPacket	Number of bytes per packet

The Aggregated Slow Attack dataset has 828,510 instances consisting of 34,097 attacks and 794,413 normal traffic resulting in a wide margin of class imbalance between attack and normal instances. High levels of class imbalance can often lead to a classifier biasing the majority class. To counteract this effect, we utilize random under-sampling to randomly select fewer of the majority class [25]. We utilize a random under-sampling ratio of 50:50 to give us a balanced dataset from which to build our models. We use the IPFIX standard for Netflow, providing significant flexibility and features. The Netflow features and descriptions are listed in Table I.

B. Machine Learning Algorithms

Six classification algorithms were selected to build predictive models based on our collected datasets: RF, two variants of C4.5 decision trees (C4.5N, C4.5D), 5NN, JRip, and NB. We selected this variety of learners to broaden the scope of our analysis. We built all the models using the Waikato Environment for Knowledge Analysis (Weka) machine learning toolkit [26]. Weka includes visualization tools and algorithms for data analysis.

RF is an ensemble algorithm that creates multiple decision trees by randomly choosing a set of features to use for each tree. RF consists of many decision trees and has low classification errors compared to other traditional classification algorithms. C4.5 is a decision tree learning algorithm to assist with classification problems and building decision trees. The classifiers organize in a hierarchy and learn parameters from the data model when training is used. We utilized a version of C4.5 using the default parameter values from WEKA (denoted C4.5D) and a version (denoted C4.5N) with Laplace smoothing activated and tree-pruning deactivated. KNN is a non-parametric algorithm that uses a distance function to find the instances which are most like the current instance. The distance function for our work is Euclidean. The method used in this research is majority voting and K=5, so there are no tied votes. NB is a simple probabilistic classifier based upon Bayes' theorem. The classifier assumes independence between features. In our experiment, we made one adjustment from the default settings. We choose to enable "useSupervised-Discretization" in Weka. We use supervised discretization to convert numeric attributes to nominal ones. The last machine learner used in our experimental process is JRip. This rule-learning technique classifies data samples into a single class and seeks a set of rules to classify data best. As the rule increases size, an initial set of rules for the class generates incrementally reduced errors.

C. Metrics

For evaluating AUC values, we generate a stratified five-fold cross-validation with four iterations. Stratified five-fold cross-validation divides the data into five non-overlapping parts and maintains the class ratio among the folds. Each iteration consists of two parts, one for test data and the remaining iterations for training data. We calculate our final AUC values by aggregating the AUC values of the models tested for each of the five elements of the data. Our experiment applied four runs of 5-fold cross-validation to provide average performance values and decrease the bias of randomly selected folds.

We evaluate each model using the AUC, True Positive Rate (TPR), Type 1 error, False Positive Rate (FPR), and Type II error metrics. TPR represents the percentage of the slow attack instances correctly predicted as an attack label. FPR is the percentage of the normal data that wrongly predicted as an attack label. The AUC curve plots TPR vs. FPR as the classifier decision threshold is varied. Higher AUC values correlate to higher TPR and lower FPR, both of which are preferred outcomes. Type I error measures the total amount of false positives. A false positive (FP) is when the outcome is incorrectly predicted as yes (positive) when it is no (negative).

False positives are a nuisance for machine-learning systems. Examples of this can be blocked email messages, trustful software flagged by monitoring systems, and application traffic categorized as potentially malicious by an intrusion detection system. False alarms account for roughly 40% of the alerts cybersecurity teams receive daily, and at large organizations, that can be overwhelming and an enormous waste of time [27]. Critical Start, a leading cybersecurity company, stated in their third-annual Security Operations Center (SOC) professional 2020 survey that more than 25% of the alerts they investigate are false positives [28]. Training a new device and or software is standard practice on the network for a predetermined duration [29]. The training process may include previously identified malicious traffic behavior from other companies, formulate new rules, and transfer them to the monitoring system. The new rules can help detect actual threats, but at the same time, a vulnerability scanner, unusual user behavior, or an additional event can set off the security system. All networks are different, and benchmarking is a critical process when making continuous improvements and implementing changes in network software, detection methods, and services.

A Type II error measures the total amount of false negatives. A false negative (FN) occurs when an instance is predicted as negative, but it is positive. These misclassification errors need to be minimized, with more emphasis placed on minimizing the Type II errors. Minimizing Type II errors while maintaining acceptable Type I errors is the most crucial focus of our paper. The explanation for this is that it is more important not

to miss an attack in a network instead of identifying an attack that is not. If a Type II error occurs, an attack has not been detected, and the network has been compromised whereas, mislabeling normal traffic as an attack is not as severe as a missed attack.

We use Analysis of Variance (ANOVA) as a metric that builds statistical models and their associated procedures in variation among and between groups. We use this to analyze the differences among group means. A one-factor ANOVA compares means from two independent (unrelated) groups using the F- distribution. For our test, the null hypothesis is that the two means are equal. Therefore, a significant result means that the two means are unequal. We used ANOVA to compare AUC means of a Slow attack detection among the eight learners and check if differences are statistically significant. Tukey's Honestly Significant Difference (HSD) post hoc test is used on our data in conjunction with ANOVA to discover significantly different means from each other. Tukey's HSD post hoc test compares all possible pairs of means to determine which specific group means (compared with each other) are different.

D. Feature Selection

As mentioned previously, PCA transforms the set of attributes. The ranking for the new attributes comes in order of their eigenvalues. The eigenvalue measures the amount of variance retained by each principal component. A subset is selected to account for a given proportion of the default variance of 95 percent by choosing enough eigenvectors. Reducing the dimensionality of our dataset can be produced by compressing it onto a new feature subspace. We achieve this by selecting the subset of the eigenvectors, otherwise known as principal components, which contain most of the information variance. The eigenvalues describe the significance of the eigenvectors. We then sort the eigenvalues in decreasing order and focus on the top k eigenvectors based on their corresponding eigenvalues. PCA is a linear transformation technique widely used across different fields, most prominently for feature extraction and dimensionality reduction. PCA's main benefit is to locate directions of the most significant variance for high-dimensional information provided by the data. It projects onto a new subspace with less than or equal dimensions compared to the original one. The principal components, also known as the new subspace's orthogonal axes, can be construed as the maximum variance directions. The constraint is that the new feature axes are orthogonal to each other.

V. Results

For each classifier, four runs of five-fold cross-validation runs are applied, producing 20 AUC values. The following sub-sections include our results and analysis for machine learner performance and the PCA feature selection method for detecting multiple slow application-layer DoS attacks. We compare our initial twelve-feature Netflow dataset results with the PCA dataset using six different classifiers. As previously mentioned, the Aggregated Slow Attack dataset has 828,510 instances consisting of 34,097 attacks and 794,413 normal traffic. Our goal is to attain the same performance or better applying the PCA method. This performance is measured by focusing on the total number of Type I and Type II errors, with more weight assigned to the Type II errors. Maximizing the detection of the 34,097 attacks is critical because an incorrect prediction results in a successful attack. We focus on expressing Type II errors in absolute terms to minimize those errors to the lowest value while maintaining acceptable AUC values with our models.

A. Initial Results

The Netflow dataset overall results indicate that five out of six of our predictive models perform well at detecting multiple slow application-layer attacks. This performance demonstrates that when machine learners use features extracted from Netflow data, they can sufficiently distinguish between normal and attack traffic and therefore, are discriminatory enough to detect slow application-layer attacks.

TABLE II: Machine Learner Results

Classifier	AUC	FPR	Type I	FNR	Type II
RF	0.981	0.0001	80	0.3820	13,021
C4.5N	0.982	0.0002	163	0.3820	13,052
C4.5D	0.981	0.0002	135	0.3830	13,074
JRip	0.656	0.0001	118	0.6850	23,355
5NN	0.982	0.0002	211	0.3830	13,060
NB	0.976	0.0910	72,369	0.0370	1,246

The results for our classifiers in Table II demonstrate that all but JRip achieved good performances. The RF, C4.5N, C4.5D, 5NN, and NB learners had similar AUC values of 0.981, 0.982, 0.981, 0.982, and 0.976, respectively for the dataset. NB produces the most favorable results for our most critical metric type II errors with 1,246. The other four top performers had over 13,000 Type II errors, followed by JRip at 23,355. Due to the high AUC values, there is the potential that certain features may be causing overfitting. In the following subsection, we address overfitting by applying the PCA feature selection method.

As mentioned previously, we used ANOVA to compare the mean AUC values of slow application-layer attack detection among the six learners to check if differences are statistically significant. Table III ANOVA results show an F value of 120.88 and a low p-value less than 6.62e-15. These values indicate the variation of means among different learners is much more significant than the variation of mean values within each learner at a 95% confidence interval.

Hence, we reject the null hypothesis (i.e., means are not the same). The differences are statistically significant between learners and multiple slow application-layer DoS attack detection. To determine the statistically significant differences between learners, we conducted a Tukey's HSD post hoc test. The Tukey's test divided our learners into two groups based upon their mean AUC values and standard deviations. As

shown in Table IV, most groupings resulted in close pairings, with the top five learners in group A and JRip in group B.

TABLE III: ANOVA Results

	Df	Sum Sq	Mean Sq	F Value	Pr>F
Learner	5	0.022656	0.002265	120.88	Pr<6.62e-15
Residuals	18	0.018740	0.000187		

TABLE IV: Tukey's HSD Group Results

Classifier	AUC	AUC StD	Group
RF	0.981321	0.000056	A
C4.5N	0.982419	0.000078	A
C4.5D	0.981376	0.000049	A
5NN	0.982128	0.000066	A
NB	0.976344	0.000056	A
JRip	0.656214	0.003458	B

B. PCA Feature Selection Results

Table V displays the results of six classifiers using PCA. We sorted the eigenvalues by decreasing significance and selected the eigenvectors based on the largest eigenvalues representing 95% of variance with our dataset. We then used the new feature subspace from PCA with our six learners.

The PCA dataset had the best results with NB. It had the lowest value for Type II errors with 554 and slightly improved its AUC and FPR at 0.981 and 0.089. NB performs substantially better than all other learners in minimizing false negatives, demonstrating that it can better protect a network. PCA produced some noteworthy results. JRip, which had been significantly high with Type II errors, was similar to RF, C4.5N, and C4.5D, and 5NN had an improved AUC of 0.774 though it was lowest. C4.5N, C4.5D, RF, and KNN had similar type II errors, and Type I. PCA produced significant results for Type II errors with NB. Compared to the initial results, Type II errors decreased from 1,246 to 554, representing more than a 50% reduction.

TABLE V: PCA Machine Learner Results

Classifier	AUC	FPR	Type I	FNR	Type II
RF	0.981	0.0001	104	0.3830	13,049
C4.5N	0.982	0.0002	164	0.3830	13,073
C4.5D	0.972	0.0002	159	0.3840	13,100
JRip	0.774	0.0003	553	0.4540	15,495
5NN	0.982	0.0002	240	0.3850	13,127
NB	0.981	0.0890	70,976	0.0160	554

VI. Conclusion

We proposed an approach to successfully detect multiple slow DoS attacks using machine learning with Netflow data. The AUC performance metric measured our models by using four runs of stratified 5-fold cross-validation. We generated our experiment on a live web server for faculty and students utilizing actual attacks alongside normal data reflecting a real-life network environment. We performed four separate attacks with different variations of each using the SlowHTTPTest tool. Our experiment aims to analyze whether machine learners using features aggregated from four separate slow DoS attacks are successful at detection while minimizing type II errors. Our investigation shows that this approach to multiple slow DoS attack detection generates high AUC values and low FPRs and FNRs for Netflow features.

NB performs the best in detecting multiple slow application-layer attacks. The high AUC values from the five Group A models address potential overfitting using the PCA feature selection method. The PCA method extracted features from the original twelve-feature set that produced the most favorable and reliable results. NB generated good results and the lowest false negatives at 554. Although NB had the most significant false positives at 70,976, the number may be low enough for IT staff to eliminate by knowing more about the network's assets, such as which systems are essential and which to disregard [30].

We also demonstrated that five out of six learners performed well using Netflow data to detect multiple slow application-layer DoS attack traffic. Therefore, machine learning with Netflow features can successfully discriminate between attack and normal data for multiple slow application-layer DoS attacks.

Future work will perform a cumulative and comparative evaluation using multi-classifiers with Netflow data to detect and discriminate multiple slow application-layer attacks.

Acknowledgment: We would like to thank the reviewers in the Data Mining and Machine Learning Laboratory at Florida Atlantic University. Additionally, we acknowledge partial support by the NSF(CNS-1427536). Opinions, findings, conclusions, or recommendations in this paper are the authors' and do not reflect the views of the NSF.

References

[1] J. L. Leevy, J. Hancock, R. Zuech, and T. M. Khoshgoftaar, "Detecting cybersecurity attacks across different network features and learners," *Journal of Big Data*, vol. 8, no. 1, pp. 1–29, 2021.

[2] S. Hares and D. Katz, "Administrative domains and routing domains a model for routing in the internet (rfc 1122)," 1989. [Online]. Available: https://www.ietf.org/rfc/rfc1136.txt

[3] M. Bishop, "Hypertext Transfer Protocol Version 3 (HTTP/3)," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-http-34, Feb. 2021, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-34

[4] "Radware's ddos handbook: The ultimate guide to everything you need to know about ddos attacks." [Online]. Available: https://security.radware.com/uploadedfiles/resources_and_content/ddos_handbook/ddos_handbook.pdf

[5] "nfcapd netflow capture daemon." [Online]. Available: https://www.mankier.com/1/nfcapd

[6] C. Sanders and J. Smith, *Applied network security monitoring: collection, detection, and analysis*. Elsevier, 2013.

[7] C. Calvert, C. Kemp, T. M. Khoshgoftaar, and M. Najafabadi, "Detecting slow http post dos attacks using netflow features," in *The thirty-second international FLAIRS conference*, 2019.

[8] C. Kemp, C. Calvert, and T. M. Khoshgoftaar, "Netflow feature evaluation for the detection of slow read http attacks," in *Reuse in Intelligent Systems, CRC Press*, 2020, pp. 181–219.

[9] M. M. Najafabadi, T. M. Khoshgoftaar, C. Calvert, and C. Kemp, "User behavior anomaly detection for application layer ddos attacks," in *Information Reuse and Integration (IRI), 2017 IEEE International Conference on*. IEEE, 2017, pp. 154–161.
[10] C. Calvert, C. Kemp, T. M. Khoshgoftaar, and M. Najafabadi, "A framework for capturing http get ddos attacks on a live network environment," in *International Society of Science and Applied Technologies*. ISSAT, 2017, pp. 136–142.
[11] C. Calvert, T. M. Khoshgoftaar, C. Kemp, and M. M. Najafabadi, "Detection of slowloris attacks using netflow traffic," in *24th ISSAT international conference on reliability and quality in design*, 2018, pp. 191–6.
[12] CVE, "Cve-2011-3192," 2011. [Online]. Available: https://www.cvedetails.com/cve/CVE-2011-3192/
[13] "Low orbit ion cannon," 2016. [Online]. Available: https://sourceforge.net/projects/loic/
[14] "High orbit ion cannon," 2016. [Online]. Available: https://sourceforge.net/projects/high-orbit-ion-cannon/
[15] "r-u-dead-yet," 2016. [Online]. Available: https://sourceforge.net/projects/high-orbit-ion-cannon/
[16] "Slowhttptest — penetration testing tools." [Online]. Available: https://tools.kali.org/stress-testing/slowhttptest0
[17] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks," *IEEE communications surveys & tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013.
[18] S. Yadav and S. Selvakumar, "Detection of application layer ddos attack by modeling user behavior using logistic regression," in *2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO)(Trends and Future Directions)*. IEEE, 2015, pp. 1–6.
[19] V. Durcekova, L. Schwartz, and N. Shahmehri, "Sophisticated denial of service attacks aimed at application layer," in *2012 ELEKTRO*. IEEE, 2012, pp. 55–60.
[20] S. R. Devi and P. Yogesh, "An effective approach to counter application layer ddos attacks," in *2012 Third International Conference on Computing, Communication and Networking Technologies (ICCCNT'12)*. IEEE, 2012, pp. 1–4.
[21] C. Calvert and T. M. Khoshgoftaar, "Impact of class distribution on the detection of slow http dos attacks using big data," *Journal of Big Data*, vol. 6, no. 1, pp. 1–18, 2019.
[22] C. Kemp, C. Calvert, and T. M. Khoshgoftaar, "Utilizing netflow data to detect slow read attacks," in *2018 IEEE International Conference on Information Reuse and Integration (IRI)*, 2018, pp. 108–116.
[23] "dumpcap dump network traffic." [Online]. Available: https://www.wireshark.org/docs/man-pages/dumpcap.html
[24] "Silk." [Online]. Available: https://tools.netsa.cert.org/silk/index.html
[25] J. L. Leevy and T. M. Khoshgoftaar, "A survey and analysis of intrusion detection models based on cse-cic-ids2018 big data," *Journal of Big Data*, vol. 7, no. 1, pp. 1–19, 2020.
[26] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.
[27] Infocyte, "Cybersecurity 101: What you need to know about false positives and false negatives," 2021. [Online]. Available: https://www.infocyte.com/blog/2019/02/16/cybersecurity-101-what-you-need-to-know-about-/false-positives-and-false-negatives/
[28] CriticalStart, "Alert overload still plagues cybersecurity industry," 2021. [Online]. Available: https://www.criticalstart.com/resources/third-annual-criticalstart-research-report-reveals-persistent-//challenges-and-some-silver-linings/
[29] R. Lemos, "3 steps to keep down security's false-positive workload," 2013. [Online]. Available: https://www.darkreading.com/analytics/security-monitoring/3-steps-to-keep-down-securitys-false-positive-workload/d/d-id/1140513
[30] L. Mandolini, "False positives and false negatives," 2021. [Online]. Available: https://securityreviewer.atlassian.net/wiki/spaces/KC/pages/818380852/False+Positives+and+False+Negatives