The Thirty-Second International Florida
Artificial Intelligence Research Society Conference (FLAIRS-32)

Detecting Slow HTTP POST DoS Attacks Using Netflow Features

Chad Calvert, Clifford Kemp, Taghi M. Khoshgoftaar, Maryam M. Najafabadi
Florida Atlantic University, Boca Raton, FL 33431
ccalver3@fau.edu, cliffkempfl@gmail.com, khoshgof@fau.edu, mmousaarabna2013@fau.edu

Abstract

Network security is a constant challenge, with new attacks and vulnerabilities being frequently introduced. Application layer Denial of Service (DoS) attacks are a rising attack variant, which inflicts network stress and service interruptions. The implementation of detection and mitigation techniques for such attacks have been a priority for some time, but more sophisticated attack permutations are constantly being introduced, often making prior prevention techniques ineffective. In this work, we focus specifically on the detection of Slow HTTP POST DoS attacks. We execute several Slow HTTP POST attack configurations within a live network environment to represent a real-world attack scenario, with varying levels of severity. For our methodology, we utilize features of network flow (Netflow) traffic to detect these attack configurations. Netflow has proven to be a more scalable solution compared to full packet capture when performing data collection, allowing for near real-time network monitoring. Eight machine learners were implemented to determine which learner would achieve optimal performance metrics when detecting Slow HTTP POST attacks. As our data is very large, we also evaluate the use of data sampling techniques to increase attack detection performance. Overall, our results show a high detection rate when detecting Slow HTTP POST attacks, achieving relatively low false alarm rates.

Introduction

The reliability of web-based systems are paramount, and attackers often exploit this reliability to render needed services inaccessible. This form of attack, known as Denial of Service (DoS), can be enacted through a number of techniques. Newer DDoS attacks tend to focus on exploiting application level protocols (Durcekova, Schwartz, and Shahmehri 2012), as a way to bypass network layer prevention techniques. Application layer attacks often target the Hypertext Transfer Protocol (HTTP), for which there are known exploits. One particular exploit utilizes legitimate HTTP POST headers to enact an attack. Due to their use of legitimate connections, Slow HTTP POST attacks can be difficult to detect before services are denied (Hirakaw et al. 2016).

This work focuses on the collection of network flow (Netflow) traffic and the ability to detect Slow HTTP POST DoS attacks using Netflow's feature set. Other data collection methods exist, such as full packet capture (FPC), but the process of analyzing all features available within a FPC can be quite costly in terms of time and computational resources. Other HTTP related DoS attacks have focused on the collection of web log traffic, which is often preferred as it can be captured and analyzed relatively quickly (Calvert et al. 2017). However, based on the behavior of Slow HTTP POST attacks, and how web logs handle requests, these logs are not a feasible solution. Using Netflow data, we evaluate the effectiveness of eight machine learners: 5-Nearest Neighbor (5NN), Naive Bayes (NB), Multilayer Perceptron (MLP), Support Vector Machines (SVM), JRip using RIPPER (Repeated Incremental Pruning to Produce Error Reduction), Random Forest (RF), and two variants of C4.5 decision trees. Many of these learners have been used successfully in the detection of other HTTP DoS related attacks (Calvert et al. 2018).

Other works performing similar collection efforts (Jati et al. 2016) are often limited in scope, and may not accurately represent normal traffic that is observed during an attack. To better represent these scenarios, our attacks were generated within a live-network environment, hosting a substantial user base. Performing our attacks in a live production environment enables our data to be more representative of real-world attack traffic. We also perform several configurations of Slow HTTP POST DoS attacks, to represent varied approaches potentially used by an attacker.

The primary contributions of this paper are as follows: 1) the implementation of a Netflow based data collection experiment with Slow HTTP POST DoS attacks on a live-network environment, and 2) the analysis of Slow HTTP POST attacks using Netflow features through Machine Learning.

The remainder of this paper is organized as follows. In the Background section, we detail prevalent Slow HTTP POST attack methods and tools. In the Related Works section, we discuss works associated with the collection of network traffic and detection of Slow HTTP POST attacks. The Experimental Procedure section outlines our collection procedure and our empirical design. In the Results section, we discuss our findings. Lastly, in the Conclusion section, we conclude our works and identify future endeavors.

Background

In a Slow HTTP POST DoS attack, attackers begin by sending a legitimate POST request to a server. The packet information associated with this request specifies a content-length

header value of an extreme size. However, as the request is legitimate, communication between the server and attacker is allowed to continue. The server then waits for the entire message body specified by the header value to be received. During this communication, the attacker will send data at a rate as low as one byte per transmission. Since the data is transmitted in regular time intervals, the server's client idle timeout is not triggered, thus tying up the server and its resources until the request eventually completes.

This attack type can be implemented in a distributed fashion using multiple hosts, or can be configured to enact multiple attack instances from a single machine depending on the tool used. This allows the attacker to perform a potentially devastating attack while using very little resources. Various Slow HTTP POST DoS tools and scripts are publicly and freely available to attackers to utilize, such as Low Orbit Ion Cannon (LOIC), R U Dead Yet (R.U.D.Y.), and Open Web Application Security Project (OWASP) Switchblade.

Related Works

Bhosale et al. identify common defense strategies against application layer DDoS attacks (Bhosale, Nenova, and Iliev 2017). Two general categories of defense techniques are identified, each with various subcategories focusing on the detection of specific attack variants. The first category proposed is destination-based (server-side) identification. These approaches pertain to tools and techniques such as DDoS Shield, reflection/amplification attacks, and anomaly detectors based on a hidden semi-Markov model. The second outlines hybrid-based approaches, which pertain to the communication between clients and servers. Some of these mechanisms include Speak-Up, human/bot differentiation, and Trust Management Helmet (TMH). This work does well to detail potential application layer attack detection methods, but does not provide much in the way of detection performance.

Tripathi et al. identify which specific HTTP servers are vulnerable to Slow HTTP DDoS attacks (Tripathi, Hubballi, and Singh 2016), and then suggest a method of detection based on anomalies. In this work, four servers were configured to serve as attack targets, each with a different platform. The attack machine was configured with the SlowHTTPTest framework. The initial findings of the researchers demonstrated that several of the tested server platforms did indeed have specific vulnerabilities to Slow HTTP POST attacks. To further expand their experiment, attacks were performed on 100 partner websites. The 100 sites were separated into four categories and attacked from a single machine running SlowHTTPTest. Results from this set of expanded tests confirmed that even live servers demonstrated vulnerabilities to these types of attacks. For detection, a statistical abnormality measurement technique is proposed. Specifically, the normal web traffic is treated as a probability distribution consisting of four HTTP request types. As most normal requests will consist of complete headers and message bodies, the researchers aim to exploit the abnormal behavior of attack traffic which performs in a contradictory manner. This difference in expected behavior is measured using Hellinger distances. Their experiments showed that Hellinger distance values are much lower during normal traffic instances than traffic associated with attacks.

Work completed by Dantas et al. re-purposes Adaptive Selective Verification (ASV) (Danats, Nigam, and Fonseca 2014), typically used to mitigate network layer DDoS attacks, to now mitigate application layer attacks. ASV successfully detects network layer attacks by assuming communication is a simple client-server stateless SYN-ACK interaction. Application layer attacks introduce an element of "state", due to the protocols being used such as HTTP. SeVen, which is based on ASV, is capable of taking state into account. SeVen and several HTTP DDoS variations were formalized into the computational tool Maude. Their defense mechanism is verified using a statistical model checker known as PVeStA. Results demonstrate that SeVen allows for robust detection with strong traffic pattern analysis. The downside to this experiment is that it is entirely simulated and has not been validated on live-network traffic.

Experimental Procedure

The following section outlines our process for collecting data, the network architecture, the implementation of our Slow HTTP POST DoS attacks, the machine learning methods applied, and the metrics used for performance evaluation.

Data Collection

Our data collection is performed within a real-world network environment. Our campus network services substantial users, handling a wide range of varying traffic types. The network is comprised of a series of switches, servers, and routers providing service to both local and online users. For our experiments, we used a pre-existing Apache web server hosting student resources. This server assumed the role of our attack target. The server runs CentOS 6.8 on a Dell 2950 Poweredge with two quad-core Intel Xeon 5300 processors and 16 GB of memory.

Our student resource server is configured using WordPress, and serves the primary role of hosting lecture material, assignments, assessments, and other content required by student users. The normal traffic for this experiment mostly relates to course work consisting of downloads, uploads, website navigation, and other communication with the web server. The server also is accessible through our extended network, which supports other faculty and student uses ranging from virtualization, email, web hosting, and audio/video streaming.

For our attack implementation, we used OWASP Switchblade 4.0 as our attack tool. We compared other common tools and settled on using OWASP Switchblade as it provided us with the most configurable options. We ran our attack from a single physical host machine utilizing multiple connections, rather than running the attack in a distributed fashion. It was determined from our initial testing that the resulting traffic from a distributed attack would be nearly identical to that of a single machine running numerous connections. The key feature difference would pertain to the multiple IP addresses associated with a distributed attack. However, as these addresses can be commonly spoofed, we remove them from our

tested feature set. Therefore, to save on physical resources and simplify our testing procedure, a single attack host was used.

We chose to implement nine different attack configurations, separated into three distinct categories. Our "stealthy" configuration utilizes single connections to prolong potential detection. We scaled up the connection amount to 125 to represent a "moderately stealthy" attack. Lastly, we increased the number of connections to 250 to represent a "non-stealthy" attack. For each of these three categories, we ran three attacks with varying timeout values: 10 seconds, 50 seconds, and 100 seconds. For all attacks, we used a content-length value of 1000. This value was chosen as it was a default value across most attack tools. All attacks targeted the same php form element on our web server, and ran for one hour each resulting in a total of nine hours of attack traffic.

FPCs allows us to observe all traffic communications as they are received. The key issue with analyzing full packets is that it can be quite resource intensive to analyze all available packet features. To lessen this resource impact, we opted to use Netflow features extracted from our FPCs. Netflow traffic refers to a high-level summary of network communications. A Netflow 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. Based upon which Netflow standard is being implemented, other attribute fields can also be produced. We utilize the IPFIX (Claise, Trammell, and Aitken 2013) standard for our flow extraction. The resulting features can be seen in Table 1.

Machine Learning Methods

We selected eight classification algorithms to build predictive models based on our collected dataset: RF, JRip, K-Nearest Neighbor (KNN), NB, MLP, SVM, and two variants of C4.5 decision trees. All models were built using the WEKA machine learning toolkit (Hall et al. 2009).

We employed default parameters for the selected classifiers within Weka, except for C4.5 where we used the default parameters (denoted by C4.5D) as well as a version (denoted by C4.5N) with Laplace smoothing activated and tree-pruning deactivated. For KNN, K is assigned a value of 5 (for 5-nearest-neighbors).

Table 1: Description of Selected Netflow Features

Feature Name	Description
Protocol	Transport-layer protocol number of flow
Packets	Number of packets in flow
Bytes	Number of bytes in flow
Flags	Logical OR of TCP flag fields of flow
Initial Flags	TCP flags in initial packet
Session Flags	All TCP flags in entire connection
Attributes	Flow attributes [SFTC]
Duration	Duration length (in milliseconds) of flow
Payload Bytes	Size of payload measured in bytes
Payload Rate	Non-overhead packet data per second
Packets/Second	Number of packets per second
Bytes/Second	Number of bytes per second
Bytes/Packet	Number of bytes per packet
Class	Class label (Attack or Normal)

Performance Metrics

Area Under the receiver operating characteristic Curve (AUC) is used to evaluate the performance of each model (Seliya, Khoshgoftaar, and Hulse 2009). The Receiver Operating Characteristic (ROC) curve plots the True Positive Rate (TPR) and False Positive Rate (FPR) of the model. TPR represents the percentage of the Slow HTTP POST attack instances that are correctly predicted as attack traffic. FPR represents the percentage of the normal data which is wrongly predicted as attack traffic. The ROC curve is built by plotting TPR versus FPR as the classifier decision threshold is varied. Higher AUC values tend to correlate to higher TPR and lower FPR, both of which are preferred outcomes.

We applied 4 runs of stratified 5-fold cross validation to evaluate our AUC values, resulting in 20 AUC values per learner. Using 5-fold cross validation divides the data into 5 non-overlapping parts, representing our folds. For each iteration, one part is reserved as test data and the remaining four parts are used as training data. Our final AUC values are calculated by aggregating the AUC values of the models being tested for each of the 5 parts of the data.

The size of our resulting dataset consists of approximately 1.6 million total instances, with 2,391 attack instances. This results in a significant class imbalance ratio, with less than 1% of data being attack traffic. Large amounts of class imbalance can often lead to a classifier being biased in favor of the majority class. To better balance data classification, we utilize random under-sampling to randomly select fewer instances of the majority class. We utilize a random under-sampling ratio of 50:50 to provide a balanced dataset from which to build our models. For comparison, we ran our eight learners on our dataset with and without RUS applied. For all learners, a marked improvement was shown with RUS. In the following section, metrics both with and without RUS are displayed in Table 2. However, due to the significant performance increases, only values for our RUS dataset are discussed.

Results

We calculated the average AUC values and their standard deviations, seen in Table 2. RF performed the best out of the eight total learners with a mean AUC of 0.9989. However, all learners performed exceedingly well, with the worst learner

Table 2: Cross Validation Results

Classifier	AUC-RUS	AUC std-RUS	AUC	AUC std
RF	0.9989	0.02×10^{-2}	0.9763	0.13×10^{-2}
C4.5N	0.9971	0.04×10^{-2}	0.9716	0.48×10^{-2}
5NN	0.9970	0.02×10^{-2}	0.9925	0.03×10^{-2}
C4.5D	0.9923	0.07×10^{-2}	0.9689	0.08×10^{-2}
JRip	0.9922	0.20×10^{-2}	0.7376	8.75×10^{-2}
SVM	0.9899	0.04×10^{-2}	0.9632	1.43×10^{-2}
MLP	0.9897	0.18×10^{-2}	0.9011	3.86×10^{-2}
NB	0.9713	0.10×10^{-2}	0.9694	0.05×10^{-2}

(NB) resulting in a mean AUC of 0.9713. The overall resulting values demonstrate the effectiveness of Netflow features when detecting this form of attack.

Evaluating both the C4.5 decision tree variants allows us to further identify key discriminating Netflow features for Slow HTTP POST detection. When evaluating the C4.5D tree, a greater emphasis is placed upon duration. The first level shows that a duration greater than 410.77 ms corresponds to a majority of attack instances. This result is to be expected, as a typical Slow HTTP POST attack will attempt to keep a connection alive for significant amounts of time. For instances with lesser duration values, bytes/second less than 277 resulted in a high number of normal instances, with a few outlying attack instances being found both above and below this threshold. As the majority of attack instances had already been accounted for due to the duration value, the resulting tree levels serve more to segregate different varieties of normal traffic.

The C4.5N tree places greater importance on relevant TCP flags at the first level. Although duration serves as a discriminating feature in later levels, the first level focuses primarily on separating instances based upon the flags associated with each flow. Descriptions of each corresponding flag can be found in Table 3. Flag values of particular relevance to this attack are "F","S","R","P" and "A", which stand for "FIN", "SYN", "RST", "PSH" and "ACK", respectively. Flag combinations of "FSRPA", "R", and "A" result in the highest number of classified attack instances. Attack instances resulting from the "FSRPA" combination at the first level, are then distinguished by a duration <= 12.474ms at the second level, and a bytes value <= 0.431 at the third. This represents a rare occurrence where the duration values are relatively low in comparison to the other attack instances. Both the "R" and "A" tree nodes correspond to relatively short remaining tree structures. This behavior could correspond to failed connection attempts by the attack tool, which would then need to request resets to continue the attack.

Conclusion

Our work outlines an approach for collecting and detecting Slow HTTP POST attack traffic in a live production environment. Netflow data was selected to build our datasets, to lessen the potential resource overhead required to analyze other data sources and still allow for near real-time detection. This work demonstrated the use of eight machine learning models to aid in the detection of Slow HTTP POST attacks. We performed 4 runs of 5-fold cross validation to obtain our AUC performance metric. Our results showed that all eight learners were capable of detecting our attack traffic with strong performance, suggesting that Netflow features are indeed a viable feature set for the detection of Slow HTTP POST DoS attacks. Our future work will aim to compare traffic patterns across several HTTP DoS variants and examine the detection performance of various machine learners.

Table 3: TCP Flags

TCP Flag	Description
SYN	Signals the first step in a three-way handshake between two hosts.
ACK	Acknowledge the successful receipt of a SYN packet.
FIN	Finished send more data from the sender
URG	Process the urgent packets before processing all other packets
PSH	Process these packets as received instead of buffering them.
RST	Indicates a reset packet has been sent from the host.
ECE	Indicates if the TCP peer is Explicit Congestion Notification capable.
CWR	Congestion Window Reduced, indicates received an ECE packet.
NS	Nonce Sum, protects against malicious concealment of packets

References

Bhosale, K. S.; Nenova, M.; and Iliev, G. 2017. The distributed denial of service attacks prevention mechanisms on application layer. In *2017 13th International Conference on Advanced Technologies, Systems and Services in Telecommunications (TELSIKS)*, 136–139. IEEE.

Calvert, C.; Kemp, C.; Khoshgoftaar, T. M.; and Najafabadi, M. M. 2017. A framework for capturing http get ddos attacks on a live network environment. In *23rd ISSAT International Conference on Reliability and Quality in Design*, 1–7. ISSAT.

Calvert, C.; Khoshgoftaar, T. M.; Kemp, C.; and Najafabadi, M. M. 2018. Detection of slowloris attacks using netflow traffic. In *24th ISSAT International Conference on Reliability and Quality in Design*, 1–6. ISSAT.

Claise, B.; Trammell, B.; and Aitken, P. 2013. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. CISCO. Technical report, Cisco.

Danats, Y. G.; Nigam, V.; and Fonseca, I. E. 2014. A selective defense for application layer ddos attacks. In *2014 IEEE Joint Intelligence and Security Informatics Conference*, 75–82. IEEE.

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

Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; and Witten, I. H. 2009. The weka data mining software: An update. In *SIGKDD Explor. Newsl*, volume 11, 10–18. ACM.

Hirakaw, T.; Ogura, K.; Bista, B. B.; and Takata, T. 2016. A defense method against distributed slow http dos attack. In *2016 19th International Conference on Network-Based Information Systems (NBiS)*, 519–523. IEEE.

Jati, G.; Hartadi, B.; Putra, A. G.; Nurul, F.; Iqbal, M. R.; and Yazid, S. 2016. Design ddos attack detector using ntopng. In *Big Data and Information Security (IWBIS), International Workshop on*, 139–143. IEEE.

Seliya, N.; Khoshgoftaar, T. M.; and Hulse, J. V. 2009. A study on the relationships of classifier performance metrics. In *2009 21st IEEE International Conference on Tools with Artificial Intelligience*, 59–66. IEEE.

Tripathi, N.; Hubballi, N.; and Singh, T. 2016. How secure are web servers? an empirical study of slow http dos attacks and detection. In *2016 11th International Conference on Availability, Reliability and Security*, 454–463. IEEE.