International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 18 (2018) pp. 13683-13688
© Research India Publications. http://www.ripublication.com

CEP Support for Detection of Application Layer Attacks

Balarengadurai Chinnaiah

Department of Computer Science and Engineering
(Principal Investigator, Department of Science &Technology, New Delhi)
Marri Laxman Reddy Institute of Technology and Management, Hyderabad, India.

Abstract

Application Layer attacks are constantly targeted by new techniques of distributed denial of service attacks (DDoS) causing service disruption and considerable financial damage. The most common type of DDoS attack involves flooding the target resource with external communication request. While some of the DDoS attacks exploit software vulnerabilities, most take advantage of the openness of the Internet and its ability to deliver packets of data from nearly any source to any destination. The aim of the research work is an overview of detection of application layer attacks for the support of Cardinality Estimation Problem (CEP) to propose new solutions, which take advantage of early detection of application layer attacks.

Keywords: Application Layer Attacks; CEP; DDoS; Detection; Security.

INTRODUCTION

The most common type of DoS attack involves flooding the target resource with external communication requests. This overload prevents the resource from responding to legitimate traffic, or significantly slows its response. While some DDoS attacks exploit software vulnerabilities, most take advantage of the openness of the Internet and its ability to deliver packets of data from nearly any source to any destination, as shown in figure 1. DDoS attacks in application layer that do not exploit software vulnerabilities can be divided into 3 types: volume based attacks, protocol attacks, application layer attacks [1] Seemingly legitimate and innocent requests whose goal is to force the server to allocate a lot of resources in response to every single request. Such attacks can be activated from a small number of attacking computers. The following are examples of such attacks [1], [2]

A. HTTP Request Attacks

Legitimate, heavy HTTP requests are sent to a web server, in an attempt to consume a lot of its resources. Each request is very short, but the server needs to work very hard to serve it.

B. SIP Request Attacks

A SIP server is flooded with legitimate or bogus SIP request messages. Again, each request is short, but server works very hard to process it. A specific SIP vulnerability is "INVITE of Death," which allows the attacker to crash the server by sending a single malformed packet.

C. DNS Request Attacks

The attacker overwhelms the DNS server with a series of legitimate or illegitimate DNS requests that waste a lot of CPU resources.

D. HTTPS/SSL Request Attacks

These attacks work against certain SSL handshake functions, taking advantage of the fact that a server will typically use a lot more computational resources in the SSL negotiation than the attacking system.

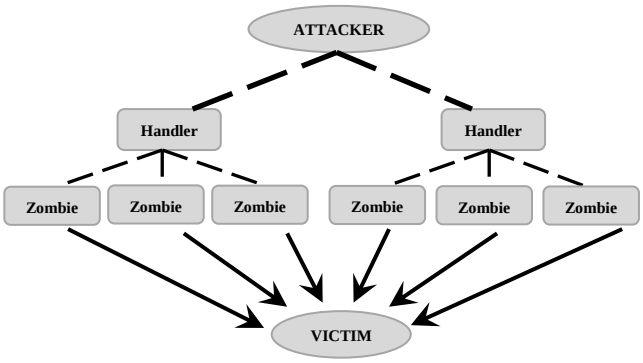

Figure 1: DDoS Attacks

These types of DDoS attacks focus on exhausting the entire resource of the networks [2], 3], [4],[16],[17]. These attacks are typically more efficient than transport layer attacks, requiring fewer network connections to achieve their malicious purposes. Application layer DDoS attacks are more difficult to deal with than volume based and protocol attacks for two main reasons [5]. First, the traffic pattern they generate is indistinguishable from legitimate traffic. Second, the number of attacking machines can be much smaller; typically, it is enough for the attacker to send only hundreds of resource intensive requests, instead of flooding the server with millions of TCP SYNs, as in a volumetric DDoS attack [6]. To better handle the various types of attacks, most DDoS protection architectures are discussed in next sections. The remaining part of the paper is prepared as follows; Technological background of Application layer attacks are described in part II, DDoS Protection Architecture were discussed in part III, In part IV compact with the cardinality estimation problem, basic plans of ESP, load variance and application layer attack detections and the experimental results for the proposed mechanism, and finally, we conclude in part V.

TECHNICAL BACKGROUND

Stateless intermediate devices usually estimate the load imposed on a remote server by estimating the number of distinct flows the server handles during the specified time period. The problem of findings the number of flows to which the received packets belong is known as the "cardinality estimation problem". This problem has received a great deal of attention in the past decade thanks to the growing number of important real-time applications, such as detecting DDoS attacks [7], estimating the propagation rate of viruses, and measuring general properties of network traffic. Several works address the cardinality estimation problem [8] and propose statistical algorithms for it. These algorithms are limited to making only one pass on the received packets, and use fixed small amounts of memory. A common approach is to map the flow ID fields in every packet into a uniformly distributed random variable. Then, one of the following schemes can be used to estimate the number of flows:

1. Order statistics estimators: the smallest (or largest) K flow identities are stored. These values are then used to estimate the total number of flows in the regions.
2. Bit pattern estimators: the highest position of the leftmost n-bit in the binary representation of the flow identities is used.

The above scheme uses M different hash functions in order to improve precision by reducing the variance. In some applications, the computational burden renders this scheme infeasible, even for small values of M (e.g., M= 10). Stochastic averaging [9], [11] is a method to overcome the computational cost at the price of reduced precision. The main idea is to use only two hash functions: the first for choosing one of the M buckets for every packet X, and the second for associating every packet X_i in every bucket with a $U\ (0,\ 1)$ value. The estimator then stores the maximum observed value of each bucket. The memory usage is the same as before (m maximum values).

The best known cardinality estimator is the HyperLogLog algorithm, which belongs to the family of min/max functions. This algorithm has a relative standard error of about $1.04/\sqrt{M}$ is the number of memory units (number of buckets) [10]. For instance, this algorithm makes it possible to estimate the cardinality of a set with 10^9 elements. There are cardinality estimation techniques other than min/max functions. The cardinality estimation describes a bit pattern sketch. Bottom m functions [9] are a generalization of min sketches, which maintain the m minimal values, where $m \geq 1$. A comprehensive overview of the different cardinality estimation techniques is given in [8]. The intuition is that the minimum observed value is exponentially distributed with a parameter that is the weighted sum of the elements. In [11], the weighted problem with integer weights is solved using binary representations. The number of storage units is not fixed, because it depends on the weights. Another weighted estimator, based on continuous bottom-m sketches, is given in [15].

All the cardinality estimation algorithms discussed above are designed to work on a fixed data set and not on an infinite data stream. A natural way to extend these algorithms to a data stream is to use a sliding window extension [10]. The objective of this extension is to estimate the cardinality of a given stream of elements at any time t over the last T time units. In [12], a sliding window extension for the HyperLogLog algorithm is provided. The main idea is to store all the elements that may be a maximum in a future window of time. By statistically monitoring the number of distinct flows during every time period, a packet level appliance can determine the Network load imposed on the end server and detect anomalies. Examples of such schemes are presented in [3]. These schemes detect DDoS attacks when the number of distinct flows becomes suspiciously high. Another statistical approach for anomaly detection is used in [12]. It monitors the entropy of selected attributes in the received packets and compares it to a pre-computed normal profile. The scheme can detect DDoS attacks when this value deviates from its normal profile.

Other techniques to detect traffic anomalies have been proposed in [1],[16]. These schemes exploit temporal patterns during a single traffic time series and operate on single time series traffic measured from a specific network link. In contrast, the scheme proposed in [12] uses correlation properties across different network links and is shown to detect network-wide anomalies. In contrast to [11], this scheme does not require detailed assumptions regarding the underlying normal traffic behavior.

The above schemes have considered all flows as imposing the same load on the end server. However, this is clearly not true in a realistic case where high-workload requests require significantly more server efforts than simple ones. We solve this problem by preclassifying the incoming flows and associating them with different weights according to the load they impose on the web server.

DDoS PROTECTION ARCHITECTURE

DDoS protection architectures are multitier in Figure 2 shows such architecture. The first tier logic is usually implemented by service and cloud providers in the customer-facing stateless routers or switches. The second tier is composed of relatively simple, but connection-oriented, reverse proxies and load balancers. The third tier consists of more sophisticated and also more CPU-intensive services including SSL termination and a web application firewall stack.

This research focuses on the 1-tier, which usually provides information to traffic collectors and analyzers. Since there may be many devices in this tier, and since it is dedicated to fast performance, its processing tasks must be simple and cheap. In particular, this tier does not have flow awareness, and it cannot perform per-flow tasks. It lacks deep knowledge of the end applications, and it is unable to keep track of the association between packets, flows and applications. For this reason, it is generally assumed that Application layer attacks cannot be detected by the first tier devices, but only by tier-2 and tier-3 devices, which are stateful. The purpose of this research is to allow a tier-1 stateless device to acquire significant application layer information and detect

Application layer attacks. It is well known that multi-tier protection is as strong as its weakest link. Very often this weakest link is tier-2 or tier-3, which will be the first to collapse in a targeted Application layer attack.

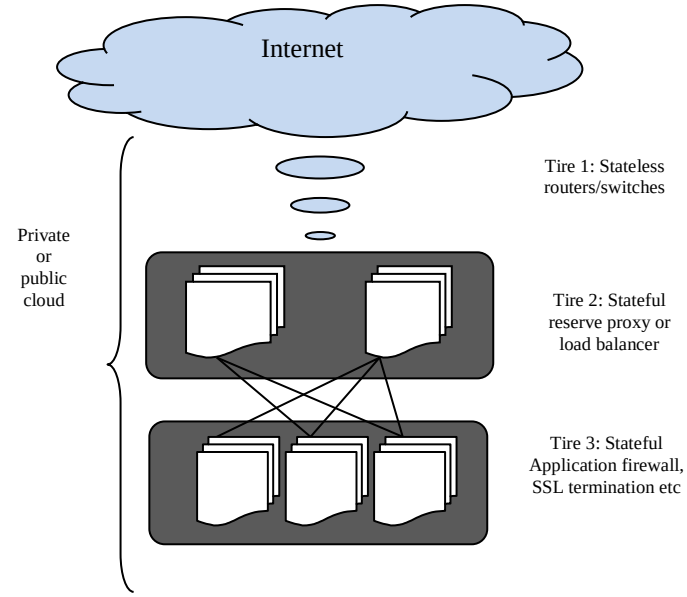

Figure 2. DDoS Protection Architecture

We believe that detecting such attacks early, in tier-1, will afford better overall protection for the following reasons:

- A suspected attack in tier-1 can trigger the opening of more tier-2 and tier-3 devices.
- A suspected attack in tier-1 can trigger the invocation of special tier-1 packet-based filtering rules, which will reduce the load on the tier-2 devices, and allow these devices to better cope with the attack.
- In many networks, tier-1 devices collect monitoring information and forward it to a centralized analyzing station. The sampling rate can be locally determined by the tier-1 device. Obviously, a high sampling rate results in significant bandwidth overhead, whereas a low sampling rate reduces the ability of the analyzing station to detect anomalies. Thus, a suspected attack in tier-1 can trigger the increase of the monitoring sampling rate.

To simplify the following description, we assume, unless otherwise explicitly stated, that all the flows continue to the same server and that the tier-1 device wants to detect attacks on this server. However, the same principles apply to the general case where the flows continue to multiple different servers and the stateless device needs to detect an attack on each of them or on any subset of them.

CARDINALITY ESTIMATION PROBLEM

A. Basic Scheme

Our basic building block is a sketch-based cardinality estimator, which is able to count the number of distinct flows that cross a stateless device while using a small constant memory. Such an estimator can help in the detection of protocol attacks, such as SYN floods, but it cannot detect Application layer attacks. We therefore propose the concept of "application weight class". A weight class l is associated with a weight W_i and it comprises all the flows that impose (roughly) the same load W_i on the end server. To measure the total load imposed on the end server during a specific time interval, the stateless device measures the value of $W = \sum_{l=1}^{C} w_l . n_l$, where,

W = total load imposed on the server

C = number of weight classes.

n_i = number of flows belonging to class l and traversing the stateless device during the considered time interval.

This measurement should be performed during every window of time. We will then seek to design an enhanced scheme. Instead of solving the cardinality estimation problem once per each class as the basic scheme does, the enhanced scheme solves the weighted cardinality estimation problem. In this problem, the total load is estimated directly, without estimating the number of flows in each class. We believe that the enhanced scheme will have a much better (smaller) variance in the trade - off between precision and memory cost.

The basic scheme and the enhanced scheme allow the stateless device to estimate the total load imposed on the Application layer of a server or a set of servers. However, they do not detect an extreme and sudden increase in the load due to an Application layer attack. Therefore, we plan to present additional schemes, which will estimate the variance of the weighted sum of the flows and the normalized variance of the weighted sum of the flows. We intend to build a prototype of the system and to use real data sets to validate our algorithms.

B. ESP Approach

Consider a stream of packets $x_1, x_2, \ldots . x_n$ passing through a router, where each packet belongs to one of N flows $y_1, y_2, .. y_n$. Each flow consists of a sequence of packets that have the same flow ID. A flow ID typically consists of the source and destination IP addresses, source and destination port numbers, and the Protocol UDP or TCP. The router needs to estimate the number N of distinct active flow belonging to the same weight class. The value of N is equal to the number of stored flow IDs. This simple approach does not scale for two reasons: (a) it requires an expensive high-speed memory of $O(n)$; (b) it requires $O(\log n)$ operations per packet. To address these issues, the following problem needs to be solved:

Instance: A stream of $x_1, x_2, \ldots . x_n$. Each packet belongs to one of N flows $y_1, y_2, .. y_n$.

Objective: Find an estimate $\hat{n}$ of N while minimizing the total memory usage and the number of operations per packet.

To estimate the cardinality $\hat{n}$ of the flow at a given time, the HyperLogLog algorithm requires computing $\hat{n} = a_m m^2 Z$, where:

(a) m is the number of buckets;

(b) $Z = (\sum_{j=1}^{m} 2^{-c_j})^{-1}$ is the harmonic mean of the m stored values (denoted by Cj); and

(c) $\alpha_m = (m\int_0^{\infty} (\log_2(\frac{2+y}{1+y}))^m dy)^{-1}$ is used for bias correction.

The use of the harmonic mean reduces the variance of the estimator and therefore increases its accuracy. The pseudo-code of the HyperLogLog algorithm is as follows:

1. Initialize m registers: $C_1, C_2, \ldots\ldots C_m$ *to 0.*
2. For each monitored packet whose flow ID is x_i do:
   (a) Let $p = \log_2(h_1(x_i))$ be the leftmost 1-bit position of the hashed value
   (b) Let $j = h_2(x_i)$ be the bucket for this flow.
   (c) $C_j \leftarrow \max\{C_j, p\}$.
3. To estimate the value of n do:
   (a) $Z = (\sum_{j=1}^{m} 2^{-c_j})^{-1}$ Is the harmonic mean of 2^{-c_j}
   (b) Return $a_m m^2 Z$,
   where $\alpha_m = (m\int_0^{\infty} (\log_2(\frac{2+u}{1+u}))^m du)^{-1}$

Recall that the basic scheme invokes C instances of the HyperLogLog algorithm, where C is the number of weight classes. Each execution of the HyperLogLog algorithm requires updating m independent buckets (estimators). Thus, the total number of estimators required by the basic scheme for a finite flow of packets is $C.m$ Minimizing the number of buckets is crucial for a large scale implementation of the proposed scheme, e.g., when a cloud operator wants to use this scheme for thousands different servers simultaneously. This is for the following reasons:

(1)If the scheme is implemented locally, at the router, each bucket needs one high-speed memory unit to hold the maximum value it has seen. This memory unit is very expensive because it must be maintained very close to the router's forwarding logic.

(2)If the scheme is implemented at a remote monitoring server, sampled packets must be forwarded to the monitoring server over the network. The sampling ratio depends on the number of buckets. Thus, the forwarding cost is proportional to this number.

Reducing the value of m while using the basic scheme increases the estimation variance. In particular, if $C > m/5$ the variance is infinite. Thus, the main challenge is to decrease m without affecting the variance, or equivalently, minimizing the variance for a given value of m. This is one of the main targets of this research.

C. Load Variance and Application Layer Attack detections

The schemes discussed so far, as well as the one we want to develop using the algorithm for the cardinality problem, are useful for performing management tasks, such as adding a virtual machine to a web server or adjusting the load balancing criteria, but not for detecting an extreme and sudden increase in the load imposed on the server due to an application layer attack. Our next goal is to extend this scheme for detecting such attacks.

Let us define $n(t)$ to be the number of active flows sampled at time t over the last T units of time, and let $w(t)$ be the weighted sum of these flows. Then, estimates the weighted sum of the flows sampled during time interval [t - T, t].

Let

$$E[w(t)] = \sum w_j(t) \qquad (1)$$

Where $w_j(t)$ are the weights of the flows received during [t-T,t]. In the same way, let

$$E[w(t)]^2 = \sum w_j(t)^2 \qquad (2)$$

Combining Equation 1 & 2 with the variance definition yields that:

$$Var\ [w(t)] = \sum w_j(t)^2 - w(t)^2 \qquad (3)$$

Thus, $Var[w(t)]$ can be estimated in the following ways

1. Estimate $w(t)$
2. Estimate $\sum w_j(t)^2$; this time, associate each flow with the square of its original weights , i.e, $w^2{}_j$;
3. Compute and return $Var[w(t)]$ using equation 3.

However, the variance can be affected not only by excessive load imposed by a few flows originated by an attacker, but also by an excessive number of new legitimate flows. To distinguish between the two cases, we will investigate another algorithm that normalizes the variance by dividing it by the number of flows n: $\dfrac{\sum w_j(t)^2 - w(t)^2}{n(t)}$

D. Experimental results

The obtained algorithm with the complementary parts is tested against real IP traffic containing packets of application layer attacks. The considered traffic trace was captured in

October 2017, in our well established laboratories by CISCO on network security. The traffic capture was performed on a router belonging to the IP backbone network. Some global characteristics of this traffic trace are given in Table 1.

Table 1: Traffic Trace

Time	No of Packets	No of Flows
60 mins	$30.10^{6`}$	254.10^{3}

The benchmark error on the estimated number of receiver ports is plotted. We notice that the benchmark error is most often within the hypothetical error δ of 4.25% It sometimes slightly exceeds this value, in fact δ is not a higher bound, but just an estimation of the benchmark error. The approximation provided by HyperLogLog is likely to be within δ , 2δ , 3δ of the exact count in, respectively, 75%, 85%, and 99% of all the cases. The results are confirmed that ESP mechanism does not affect the correctness of the original HyperLogLog algorithm.

CONCLUSIONS

To better handle the various types of application layer attacks, most DDoS protection architectures are multi-tier. We believe that detecting such attack early, in the tier -1, will afford better overall protection. Thus, the purpose of this research work is to allow a tier-1 stateless device to acquire significant application layer information and detect application layer attacks. Lastly, we did the experiments on traffic trace was captured in October 2017, in our well-established laboratories by CISCO on network security. The traffic capture was performed on a router belonging to the IP backbone network. The acquire results are confirmed the efficiency ESP methods to detect the application layer attacks in terms of accuracy, memory usage, and time response.

ACKNOWLEDGMENT

This paper is submitted as part of research project “Efficient Detection of DDoS Application Layer Attacks” is sponsored and funded by Department of Science and Technology (DST), Science and Engineering Research Board (SERB), New Delhi. I would extend my sincere thanks to DST for supporting this research.

REFERENCES

[1] T. Peng, C. Leckie, and K. Ramamohanarao, “Survey of network-based defense mechanisms countering the DoS and DDoS problems” ACM Computer Survey , 39(1), pp: 1-42, 2007.

[2] Y. Xuan, I. Shin, M. T. Thai, and T. Znati, “Detecting application denial-of-service attacks: A group-testing-based approach”, IEEE Trans. Parallel Distributing systems, 21(8), pp: 1203–1216, 2013.

[3] M. Srivatsa, A. Iyengar, J. Yin, and L. Liu. “A middleware system for protecting against application level denial of service attacks”. Springer Middleware, pp: 260–280, 2006.

[4] S. Ganguly, M. N. Garofalakis, R. Rastogi, and K. K. Sabnani “Streaming algorithms for robust, real-time detection of ddos attacks”. In 27th IEEE International Conference on Distributed Computing Systems (ICDCS 2007). IEEE Computer Society, 2007.

[5] P. Clifford and I. A. Cosma. “A statistical analysis of probabilistic counting algorithms”. Scandinavian Journal of Statistics, Vol: 39, pp:1-14, 2011.

[6] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. Journal of Computational System. Science, Vol: 31, pp:182–209, 2013.

[7] Yousra Chabchoub, Raja Chiky and Betul Dogan “How can sliding HyperLogLog and EWMAdetect port scan attacks in IP traffic” EURASIP Journal on Information Security, Vol 5, pp:1-11, 2014.

[8] E. Cohen. “Size-estimation framework with applications to transitive closure and reachability” Journal of Computational System Science 55(3), pp: 441–453, 2007.

[9] L. Feinstein, D. Schnackenberg, R. Balupari, and D. Kindred. Statistical approaches to ddos attack detection and response. In DISCEX (1), pp: 303–314, 2013.

[10] J. Lumbroso. An optimal cardinality estimation algorithm based on order statistics and its full analysis. In Analysis of Algorithms (AofA) pp: 489-504, 2010.

[11] C.Balarengadurai, S Saraswathi, “Fuzzy logic-based detection of DDoS attacks in IEEE 802.15.4 low rate wireless personal area network*”* Int. J. Trust Management in Computing and Communications, Vol. 1, Nos. 3/4, pp: 243-260,2013.

[12] C.Balarengadurai, S Saraswathi, “Fuzzy based Detection and Prediction of DDoS attacks in IEEE 802.15.4 low rate wireless personal area network*”* International Journal in Computer Science, Vol:10, pp:293-310 ,2013.

[13] C.Balarengadurai, S Saraswathi, “Detection of Exhaustion attacks over IEEE 802.15.4 MAC Layer using Fuzzy systems*” IEEE Conference on ISDA,* pp: 527-532,2012.

[14] Chabchoub Y, Hebrail G, “Sliding HyperLogLog: estimating cardinality in a data stream over a sliding window” ICDM workshop on large-scale analytics for complex instrumented systems (LACIS), Sydney, 13, 2010.

[15] Durumeric Z, Wustrow E, Halderman JA: ZMap, “Fast internet-wide scanning and its security applications” 22nd USENIX security symposium, Washington, D.C., USA, pp: 14–16, 2013.

[16] C.Balarengadurai, S Saraswathi,, 'A Fuzzy Logic System for Detecting Ping Pong Effect Attack in IEEE 802.15.4 Low Rate Wireless Personal Area Network' in proceedings of Advances in Intelligent Systems and Computing-Springer Verlag, Vol: 182, pp: 405-416,2012.

[17] C.Balarengadurai, S Saraswathi 'Comparative analysis of detection of DDoS attacks in IEEE 802.15.4 low rate wireless personal area network' in proceedings of Elsevier Procedia Engineering ,Vol.38, pp.3855-3863,2012.