Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 10 (2012) 432 – 441

Procedia Computer Science

The 3rd International Conference on Ambient Systems, Networks and Technologies (ANT-2012)

Tackling Application-layer DDoS Attacks

Hakem Beitollahi and Geert Deconinck

Katholieke Universiteit Leuven, Electrical Engineering Department, Kasteelpark Arenberg 10, Leuven, Belgium
{hakem.beitollahi and geert.deconinck}@esat.kuleuven.be

Abstract

In application-layer distributed denial of service (DDoS) attacks, zombie machines attack the victim server through legitimate packets such that packets have legitimate format and are sent through normal TCP connections. Consequently, neither intrusion detection systems (IDS) nor victim server can detects malicious packets. This paper proposes a novel scheme which is called ConnectionScore to resist against such DDoS attacks. During the attack time, any connection is scored based on history and statistical analysis which has been done during the normal condition. The bottleneck resources are retaken from those connections which take lower scores. Our analysis shows that connections established by the adversary give low scores. In fact, ConnectionScore technique can estimate legitimacy of connections with high probability. To evaluate performance of the scheme, we perform experiments on Emulab environment using real traceroute data of ClarkNet WWW server.

Keywords: Application-layer DDoS attacks, HTTP flood, Connection score

1. Introduction

Distributed denial of service (DDoS) attacks are categorized into two classes: network-layer DDoS attacks and application-layer DDoS attacks. In network-layer DDoS attacks, attackers send a large number of bogus packets (packets with bogus payload and invalid SYN and ACK number) toward the victim server and normally attackers use IP spoofing. In network-layer DDoS attacks, the victim server or IDS can easily distinguish legitimate packets from DDoS packets. In contrast, in application-layer DDoS attacks, attackers attack the victim server through a flood of legitimate requests. In this attack model, any zombie machine has to establish a TCP connection with the victim server, which requires a genuine IP address; otherwise, the TCP connection cannot be established. HTTP flood is a well-known example of application-layer DDoS attacks.

Most well-known DDoS countermeasure [1, 2] techniques are against network-layer DDoS attacks. Those techniques cannot handle application-layer DDoS attacks. CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) puzzles [3, 4] are one of the proposed techniques against application-layer DDoS attacks. However, CAPTCHA puzzles suffer from some challenges which we discuss them in the next section. This paper proposes a novel and systematic technique against these attacks which is called ConnectionScore technique. Our goal is to design a technique such that tackles

1877-0509 © 2012 Published by Elsevier Ltd. Open access under CC BY-NC-ND license.
doi:10.1016/j.procs.2012.06.056

application-layer DDoS attacks without using CAPTCHA puzzles or uses minimum level of CAPTCHA puzzles.

ConnectionScore technique proposes that during normal conditions, any server can measure various statistical attributes for its users and their traffic. The statistical attributes represent behavior and characteristics of normal users. A server can keep the statistical attributes as a reference profile. Now, when an attack occurs against the server, the server assigns scores to the connections based on the reference profile. It retakes bottleneck resources from those connections which have gotten low scores. The key point is that the connections which have been established by the attackers get low scores because they cannot have statistical attributes of the normal users. Dropping suspicious connections is adjusted based on the current level of the overload of the server. To evaluate the performance of the scheme, we perform experiments on the Emulab environment using real logged data of ClarkNet WWW server as a case study. Our experiments show that ConnectionScore scheme can precisely detect malicious connections and retake the bottleneck resources from them.

2. Related work

In this section, we do not review countermeasure techniques against network-layer DDoS attacks. Two useful surveys that collect these techniques are [1, 2]. The most promising technique against application-layer DDoS attacks is CAPTCHA puzzles. A CAPTCHA puzzle [3, 4] is a type of challenge-response test used in computing as an attempt to ensure that the response is generated by a human not by a machine. Any user must successfully solve a CAPTCHA puzzle before establishing a TCP connection with the server. However, the CAPTCHA solution has the following challenges. (1) Patience of the users: several reports [5, 6] show that these tests annoy the users and they are not user-friendly. Since many users have little patience to solve a CAPTCHA test and wait for response, a site that uses CAPTCHA may drive away legitimate users. (2) Breaking techniques: today, several image recognition techniques have been proposed to break CAPTCHAs [7]. (3) Insecure implementation: some CAPTCHA protection systems can be bypassed without using OCR (Optical character recognition) simply by re-using the session ID of a known CAPTCHA image. (4) Labor attack: some reports [8, 9] indicate that there are free or cheap 3rd party human labor to break CAPTCHAs. The above challenges encourage us to find a solution for application-layer DDoS attacks that either do not need CAPTCHA test or uses it at the minimum level. Due to challenges which we enumerated for the CAPTCHA techniques, researchers have attempted to solve application-layer DDoS attack without using CAPTCHAs. Here, we discuss some of them.

Yatagai, et al. [10] propose two simple ideas: (1) when there are attacks from compromised clients with the same virus or bot, the server can observe the same browsing order of pages continually at the server, (2) attackers browse a web page for a shorter time than normal users; thereby if a user browses a web-page shorter than a threshold time, it is considered as malicious. The first idea does not work as the attacker can set zombie machines to send requests for random pages. The second idea also does not work as the attacker can browse a web-page for a longer time and simply pass the threshold. Thapngam et al. [11] classifies attack rate into two categories: predictable rate and nonpredictable rate. Predictable rates include constant rate, monotonically increasing rate and periodical rate. However, nonpredictable rate has no classification. The author then proposes a Pearson correlation coefficient theorem to detect predictable rates for all three classes. However, they have no solution when attackers send requests at random and unpredictable ranges. Xie and Yu [12] assume that normal users always access pages sequentially based on hyperlinks organization, while attackers do not follow this organization and access random pages directly using their URL. Then authors recognize attackers through the entropy test. Although, the first assumption is true, but the second assumption which is the base of the algorithm is not always true. We note that an attacker can easily design a tool and ask zombie machines to follow pages based on the hyperlink organization. In this case, the entropy value of attackers and normal users locate in the same range and the server cannot detect zombie machines.

3. Description of the ConnectionScore scheme

The basic idea of the ConnectionScore scheme is as follows. A server can measure various statistical attributes for its users and their traffic during the normal condition when there is no attack against it. Undoubtedly, the measured statistical attributes represent behavior and characteristics of the normal users of the server. These attributes are site-dependent which means that an outside attacker cannot be aware of such statistical attributes. The server can consider the measured statistical attributes as a reference profile and use it during the attack time as a judgment reference point. When an application-layer DDoS attack occurs against the server, the server assigns scores to connections based on the reference profile. As can be seen below, the connections which get lower scores are more probable to be the connections which have been established by the attackers. In fact, the ConnectionScore scheme predicts legitimacy of connections with high probability. In the next step, in a feedback-control process, the server retakes bottleneck resources from the connections which have lower scores until its current load reaches below a threshold.

3.1. Attributes

A server can consider various attributes for users and their traffic. In this paper, we introduce some of them, though other attributes can be discussed.

Request rate and download rate

The request rate represents the number of requests that a user sends to the server in a specific interval time. The download rate represents the number of bytes that a user downloads from the server during a specific interval time. The server measures request rate and download rate for different random users during different random times a day, different days, different weeks, etc. Then the server can find the cumulative distribution function (CDF) for both request rate and download rate of normal users.

Uptime and downtime

The uptime represents a time that a user starts communication with the server until he terminates the communication with the server. The downtime represents the interval time from the time that a specific user disconnects from the server until the time he connects again to the server. The server randomly selects users during different times (days, weeks, etc.) and calculates their uptime. Although the downtime attribute cannot be measured for all users as some users may either never connect again to the server or again connect to the sever too late, surely some users reconnect every few days or several times a day. In this case, the server can measure the downtime for such users. Next, the server can obtain CDF for both uptime and downtime of normal users.

Browsing behavior

The browsing behavior of users of a web-site depends on two main factors: a) the structure of the website and b) the behavior of users. Normally, any web-server composes of many numbers of web-pages that have been organized hieratically through hyperlinks. The behavior of users indicates that which pages are more favorite for users (page popularity). How much fraction of hyperlinks in a typical page is clicked by normal users? And some other behaviors that we discuss them below.

Page classification based on type: in several servers, pages can be virtually or logically classified based on their types. For instance, a news web agency can classify its web-pages based on the type of the news: politics, economics, culture, sport, etc. Then the server can measure request rate of each user for each category. The collected data of different users of different times assists the server to depict CDF for each category. Similarly, the CDF helps the server to consider a threshold rate for each category such that during the attack time if a user has request rate more than threshold rate of a category, he gets negative score.

page access rate and page popularity: surely, any page has different access rate. Some pages are frequently requested by the users and some have very few requests. The result of web mining [13] shows that in most cases, about 10% of the pages of a website draw 90% of the access. This attribute can significantly help the server to detect malicious connections as attackers are not aware of the popularity of pages and thereby access to the pages randomly. The server can measure access rate for any page during a time interval. Then the server can measure popularity for each page as the following formula: $p_i^t = \frac{a_i^t}{\sum_{j=1}^{N} a_j^t}$. where p_i^t shows popularity of page i during time interval t; N is the number of pages and a_i^t is the access rate for page i during the time interval t.

The key point is that the server classifies pages based on their popularity into five major categories: very low, low, medium, high and very high popular pages. Then, it classifies any of the above major categories into some smaller classes such that pages which have similar or nearly popularity locate in one class. Then, for each class, the server depicts CDF of the percentage of requests of users for pages of each class. Next, the server determines a threshold rate for each class based on CDF. During the attack time, if the percentage of requests of a user exceeds the threshold rate for a class, he gets a negative score as explained below.

Hyperlink fraction click: suppose a page has k hyperlinks. What fraction of hyperlinks of a page is clicked by a user? During the normal condition, a server can extract for each page a fraction of hyperlinks on which a user clicks with a probability. Then the server can define a threshold for each page based on its observation from normal users such that the higher deviation from the threshold, the higher the probability to be a malicious user.

Hyperlink depth: let us explain this attribute with this question: how many sequential interlink pages a user requests for? The depth (D) a user proceeds in hyperlink pages is an effective attribute that a server can extract for its normal users. Similarly, a server can compute CDF for this attribute.

There are also some other attributes in this category such as “out of time pages”, “repetitive pages” and “ sequential-hyperlink pages” which we omit to represent them here due to page limit. Find them in [14].

Source IP address distribution

In most cases, source IP distribution of legitimate users is different from source IP distribution of attackers. While, distribution form of source IP addresses of legitimate users is more uniformly scattered across the Internet; the distribution form of source IP addresses of attackers is more cumulative in some places. This is because an adversary can catch several zombie machines in the same LAN or same area. For instance, in a university or in a company, most users rely on central firewall that has been installed on the gateway (they, themselves, do not care about installing firewalls or regularly update the tool); thereby, if an adversary could break the firewall’s rules, he can capture several zombie machines from the same LAN. Figure 1 shows an example of this attribute where the left image shows source IP address distribution of a server just prior an DDoS attack (the right image). In fact, in some attacks, we can see the creation of several clusters of source IP addresses; while before the attack there were no such clusters. The IP addresses within the range of clusters are more suspicious to be the IP address of zombie machines. However, we should have in mind that source IP distribution always cannot help, as the source IP distribution of zombie machines in some attacks may be uniformly distributed across the Internet.

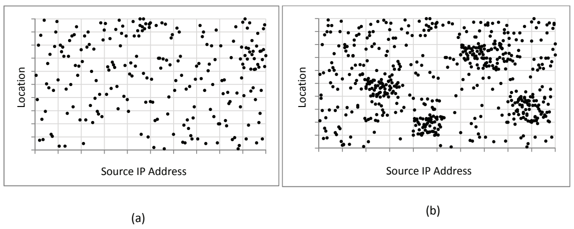


Fig. 1: Source IP address distribution: a) just before attack, b) after attack

Arrival distribution rate of users

In any server, the arrival rate of users is different during different times a day. A server can measure arrival rate of users for different times a day, during different days a week, month, etc. Hence, a server can predict the arrival rate for different times for future days. Normally, arrival rate of users follow the Poisson distribution. The Poisson distribution theory indicates that if the expected arrival rate in an interval is λ, the probability that k users connect the server during that interval is: $f(k;\lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$. For example if any second, two users connect the system ($\lambda = 2$), then the probability that 20 users connect to the server during a second is 5.8×10^{-14}. In a DDoS attack, a large number of attackers simultaneously or in a short time connect the server. According to the above probability, we can predict that most of the users that connect to the system during that particular time are attacker's machines.

3.2. Analysis of Attributes

Let us discuss the abilities of attributes and show that whether attackers can bypass attributes. An attacker can thwart some attributes, but he encounters some difficulties and limitations. For instance, an attacker can send request rate in the order of legitimate users or he can have download rate in the order of legitimate users as well. But, the problem is that in such cases, the attacker should have numerous zombie machines to be able to set up an effective attack (i.e., meek attack, see below).

The attribute of "source IP address distribution" is somewhat in contradiction with attributes of "request rate" and "download rate". If an attacker tries to avoid creation of clusters of source IP addresses, then he should use smaller number of zombie machines. In this case, the attacker can choose a set of zombie machines from the pool of zombie machines that he has in such a way that IP address of zombie machines are evenly distributed across the globe. But, the point is that in this case, the attacker should use a high rate for request rate and download rate to have an effective attack. On the other hand, if the attacker wishes that all zombie machine have send/receive rates in the order of legitimate users, then he should use numerous number of zombie machines for an effective attack (meek attack). In this case, creation of clusters of source IP addresses is inevitable!

Tackling the attribute of "arrival rate distribution" is not easy for the attacker. For instance, suppose in a meek attack, the attacker uses 20000 zombie machines to bring down a server. If he organizes the attack in such a way that every second, 10 zombie machines establish connections, then about 34 minutes are required for completing the attack scenario. Moreover, the attribute of "uptime" is in contradiction with this scheme because those zombie machines which have established connection earlier encounters the limitation of "uptime" and get negative scores of uptime.

To bypass attributes of "hyperlink fraction click" and "hyperlink depth", an attacker encounters a serious challenge because these two attributes are in contradiction. First, we note that any page has a particular threshold rate for "hyperlink fraction click". Second, an attacker is not aware of these threshold rates. However, an attacker may try to click small fraction of hyperlinks of each page to guarantee that he remains below the threshold rate for each page. In this case, first, an attacker does not know to select which particular rate and does not know the selected rate is how much large or how much small regarding to the threshold rate of different pages. The second issue which is much important is that if an attacker chooses a small fraction rate for click on each page, then he should proceed in depth. We note, in this case, the attacker get negative score of attributes of "hyperlink depth".

We believe an attacker cannot frustrate the attributes of "uptime", "downtime", "page classification based on type" and "page popularity" because these attributes are completely site-dependent and thus more difficult for an outsider attacker to collect such information. In [14], we show that attackers cannot bypass the attribute of "page popularity". However, due to page limit, we ignore to bring proof in this paper.

3.3. Computing scores

In this section we formulate the notion of score for each attribute and subsequently for each connection. An established connection c has a set of attributes A_i^c, where A_1^c could be the request rate, A_2^c, the download rate, etc. Let $S(A_i^c)$ be the score of connection c associated with attribute A_i. We then calculate the total score for connection c as the sum of scores of all attributes: $S(c) = \sum_{i=1}^{n} S(A_i^c)$; where n is the number of attributes. Now, let us explain how the score of a connection is calculated for an attribute such as A_i. The score for attributes of "request rate", "download rate", "uptime", "page popularity", "page classification based on type", "hyperlink fraction click", "hyperlink depth" and "arrival rate" is calculated as follows. Suppose $y = f_i(x)$, where f_i shows cumulative distribution function (CDF) for attribute A_i in the reference profile. Assume that the control unit has determined a pair of (x_b, y_b) as a reference baseline; so, $y_b = f_i(x_b)$. In the next section, we explain how the control unit through a feedback-control process selects the appropriate baseline point. Assume that the value of the connection c for attribute A_i is shown by x_i^c. The score of a connection for mentioned attributes is calculated as the following formula:

$$S(A_i^c) = \begin{cases} 0, & \text{if } x_i^c \leq x_b \\ -1 \times k^{div(\frac{x_i^c - x_b}{\Delta x})} \times \frac{x_i^c - x_b}{\Delta x}, & \text{if } x_i^c > x_b \end{cases} \tag{1}$$

where k is a geometric constant value (e.g., 1.2), $div(m/n)$ shows quotient m per n and Δx is a constant scale factor. As can be seen, the higher the deviation from the base-line value (i.e., $x_i^c - x_b$), the lower the score which is decreased in a semi-geometric progression. Any server can select k and Δx appropriately based on the volume of attack rate. The score for attribute of "downtime" is calculated as for the above attributes, but in the reverse direction. The score for attributes of "source IP address distribution" is a constant value: if a connection has been established from locations where we guess are the location of attackers, the connection gets a constant score, for example -1[1]; otherwise the connection gets zero.

3.4. Control unit

The goal of the control unit is to prevent exhaustion of bottleneck resources and, as a result, prevent that the server goes down during the attack. To achieve this goal, the control unit defines three thresholds for each bottleneck resources and defines three strategic states: red, yellow and green. For each bottleneck resources, we define threshold1, threshold2 and threshold3 as 90%, 80% and 60% of total capacity of the bottleneck resource, respectively[2]. When the load of a bottleneck resource exceeds **threshold1**, we say so the situation of the bottleneck resource is in the red state. Whenever, traffic is controlled and the load of the bottleneck resource returns below **threshold2**, but still is above **threshold3**, we say so the situation of the bottleneck resource is in yellow state and finally, when the load of the bottleneck resource returns below **threshold3**, we say so the situation of the bottleneck resource is in green (normal) state.

The control unit periodically checks status of the bottleneck resources of the server (e.g., bandwidth, TCP/IP stack, CPU cycles, memory, etc.). Whenever the load of one of the bottleneck resources of the server exceeds threshold1, the server goes to the freeze mode and control procedure is started. As long as the server is in the freeze mode, it does not accept new connections. The next task of the red state is that the control unit assigns scores to the established connections and then drops suspicious connections until the load of bottleneck resource(s) has(ve) returned below threshold2. When the state of the system exits the red state, the server exits the freeze mode and accepts new connections. When the system is in the yellow state, whenever a request for a new connection (i.e., SYN packet) arrives, the control unit first checks whether the source IP address of the SYN packet is in the blacklist or not (see below). If it is in the blacklist, then its previous score is summed up with the score of the "downtime" attribute. If it is not in the blacklist, only score of "downtime" attribute is calculated for it. Then the score of the new connection is compared with the score of established connections; if the lowest score of the system is lower than the score of the new connection, the connection with the lowest score is dropped and the new connection is appropriately established.

The control unit always follows the following rules: **(Rule1:)** the connections which get zero score are not dropped. **(Rule 2:)** if score of a connection is equal or greater than a threshold (e.g., -10), the control unit should send a CAPTCHA test for the user. If the user solves the test, the connection is not dropped; otherwise, the connection is dropped. Let us call this threshold the "drop threshold". **(Rule 3:)** if the score of a connection is smaller than the drop threshold (e.g., -10), the connection is candidate for dropping without need to test CAPTCHA puzzle with it. The control procedure is as follows.

1. The control unit initializes the baseline point for each attribute to a specific value. The decision about initial value of baseline points can be made in the pre-attack stage. Normally, initial values are selected such that minimum amount of false positive occurs; thereby they will be initialized to the maximum amount such that false positive at the beginning are zero.

[1]The value of constant score can be discussed and it is possible that a server could extract a suitable value by its experience.
[2]These thresholds can be varied from a server to another server.

2. The control unit monitors all established connections for duration of so called "score interval" (e.g., one minute) and then computes score for each established connection based on the baseline point for that interval.
3. The control unit starts dropping connections from the connection which have the lowest score. It continues dropping connections until either no connections with negative scores remain (considering the rules) or the load of bottleneck resource(s) returns below threshold2.
4. If no bottleneck resource is in the red state, the server exits the freeze mode. If the state of at least one bottleneck resource is in the yellow state, the control unit still calculates scores for the connections and waits for new connections.
5. If the state of all bottleneck resources returns to the green state, the control procedure is terminated.
6. If at least one of the bottlneck resources is in the red state and there are no connections with negative scores smaller than drop threshold (e.g., -10), the control unit changes the baseline point appropriately. For all attributes except "downtime" and "source IP distribution", the baseline point is changed as follows: suppose in the CDF function of an attribute, (x_b, y_b) and (x'_b, y'_b) shows the current and next baseline points, respectively. For the next base-line point, the control unit decreases y_b by a Δy. So, we have $y'_b = y_b - \Delta y \Rightarrow x'_b = f^{-1}(y_b - \Delta y)$. The amended baseline point, i.e. the next base-line point would be $(f^{-1}(y_b - \Delta y), y_b - \Delta y)$. The amount of Δy is considered appropriately, for instance, 0.1 or 0.05. For attribute of "downtime" the calculation is similar, but the direction is opposite. In other words, for this attribute, we have $(x'_b, y'_b) = (f^{-1}(y_b + \Delta y), y_b + \Delta y)$. For attribute of "source IP distribution", there is no baseline point and the scores are calculated as before.
7. The algorithm returns to step 2. This loop is continued until the condition of step 5 is succeeded.

Here, we address some additional issues of the control unit. (1) It is not required that at each iteration of the loop, baseline points for all attributes should be changed. Sometimes, the control unit may only change baseline points for some attributes and not for all of them. (2) When a connection is dropped, the IP address of the connection within its score is recorded in a list which is called blacklist. (3) A server can select a suitable "drop threshold" before the attack time based on the maximum absolute value of negative scores that legitimate connections may get.

4. Analyzing attributes for a case study

This section studies and analyzes the nature of the distribution of the mentioned attributes for a real case-study in the Internet. Our case study is the real-life Internet traces collected from the traffic archive of ClarkNet WWW server. The traces contain two week's worth of all HTTP requests to this web server[3].

Figure 2, parts a, b, c and d, shows cumulative distribution function for request rate, download rate, uptime and downtime, respectively. **Request rate:** as can be seen more than 50% of users have request rate per second less than 0.066. It means that more than 50% of users have sent a request to the server every 15 seconds. Moreover, more than 80% of users have request rate per second less than 0.25. As figure shows about 10% of users have sent more than one request every two seconds. For this case study, we suggest $(x_b, y_b) = (0.5, 0.9)$ as an initial base-line point. **Download rate:** as the figure shows, more than 50% of normal users have a download rate of less than 1000 bytes per second. More than 80% of users have a download rate less than 3000 bytes per second. Moreover, only 10% of users have a download rate more than 6000 bytes per second. We suggest $(x_b, y_b) = (6000, 0.9)$ as an initial base-line point of this attribute for this case-study. **Uptime:** as can be seen about 54% of users have stayed online for less than 1.5 minutes. About 80% of users have an uptime less than 5 minutes and only 4% of users have an uptime more than 16 minutes. The point of $(10, 0.9)$ can be considered as an initial base-line point for attribute of "uptime" in this case-study. **Downtime:** as can be seen more than 50% of those normal users have downtime more than 8 hours and more than 80% of them have downtime more than 4 hours. Our analysis shows that only 0.3% of users have downtime less than one hour. The point of $(3, 0.1)$ can be considered as an initial base-line point for this attribute.

[3]http://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html

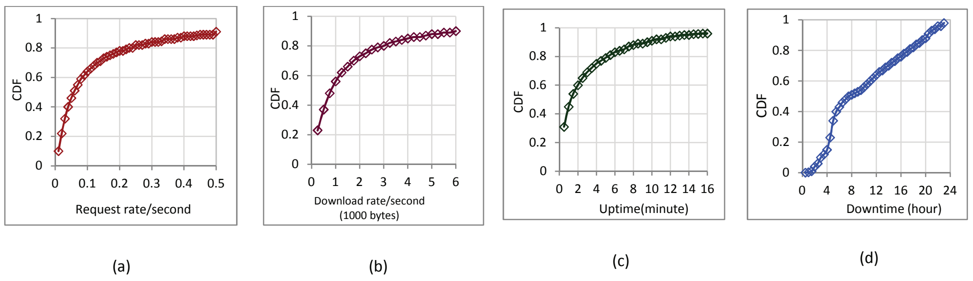


Fig. 2: Cumulative distribution function for (a) request rate/second, (b) download rate/second, (c) uptime and (d) downtime

Figure 3.a shows distribution of page popularity for day 3 between 11:00 to 14:00 o'clock. As discussed above, we divide pages based on their popularity into five major categories: very low, low, medium, high and very high popular pages. Next, for more accuracy, any of the above major categories may divided into some smaller classes. Figure 3 shows that pages of class 1 compose very low popular class; pages of classes 2 and 3 compose low popular class; pages of classes 4 to 12 compose medium popular class and finally pages of classes 13 to 17 and 18 to 19 compose high and very high popular classes, respectively. In Figure 3, popularity of class i is more than class j when $i > j$. Figure 3.b shows threshold rate (percentage) for different classes based on CDF of 90% to 95% (depending on the class) for four different days. As can be seen, the threshold rate of a specific class is in a similar range for different days. So, we can determine an upper bound and fixed threshold rates for classes independent from days.

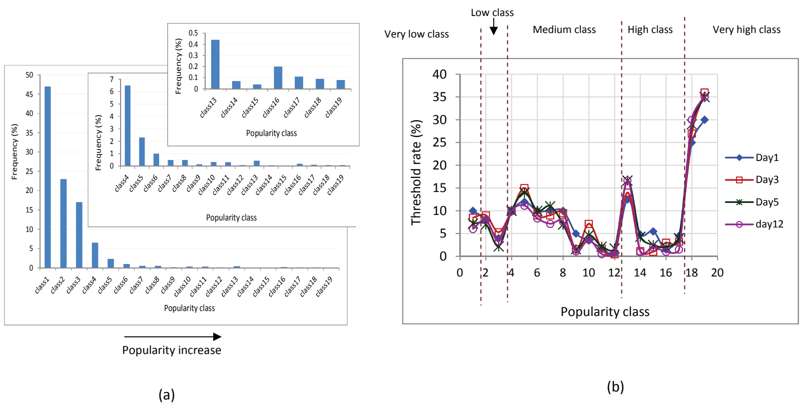
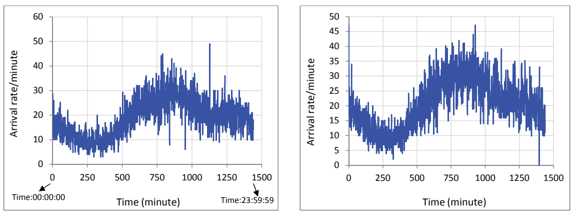


Fig. 3: Page popularity: (a) Frequency rate for different classes of pages, (b) threshold rate for different classes of page popularity

Figure 1.a shows source IP distribution for duration of 5 minutes of day 6. As can be seen, users are scattered uniformly across the Internet. Figure 4 shows arrival distribution rate for day 1 and day 12. As can be seen arrival rate is different for different times a day. In both days, the arrival rate after mid-night between 01:00 to 07:00 is lower than other parts of day. The arrival rate between 11:00 to 16:00 is maximum. The average arrival rate for this period of the day is about 27 users per minute. As can be seen in the maximum case less than 50 users have connected to the server during a minute; in other words, less than one user per second.

5. Experimental results

To evaluate the effectiveness of the ConnectionScore scheme, we set up some experiments on the Emulab environment. In these experiments, we simulate day 6 of Clarknet www server. We generate 3559 pages with different sizes according to distribution of file sizes of day 6 before 14:00 o'clock and then upload them



b) Day 1 a) Day 12

Fig. 4: Arrival distribution rate: a) day 1 and b) day 12

in the server. In the experiments, legitimate clients follow all attributes of the real users of Clarknet www server. We assume that the attack is started at time 14:03. We set the bandwidth of the server to 10^6 bytes per second.

In these experiments, we are interest to see 1) score distribution of connections, 2) how fast the server can recover from the attack, 3) percentage of false positive and false negative, 4) percentage of candidate connections for false positive and 5) percentage of the legitimate connections that get negative scores (R_L) and also percentage of the malicious connections that do not get negative scores (R_A). We evaluate the scheme against two types of attacks: 1) common attacks and 2) meek attack. For the common attack, we follow a real scenario to model this attack which has been represented in [10]. In this scenario, attackers use three viruses programs: Netsky.Q, Trojan_Sientok and BlueCode.Worm to send HTTP GET requests to the server. These virus programs send the requests in about 300 milliseconds, 250 milliseconds and 137 milliseconds, respectively. In this experiment, 150 machines play the role of attackers. For the meek attack, 600 attackers create TCP connection and follow attributes of request rate and download rate of legitimate clients. In other words, request rate and download rate of attackers are in the order of legitimate users.

Figure 5.a and 5.b show score distribution for established connections when the server is under the common attack and the meek attack, respectively. Red bars show score of malicious connections and green bars show score of legitimate connections. As can be seen, about 25% of legitimate connections get negative score, but with small absolute values (maximum 11). In the common attack, most of malicious connections get very low scores (i.e., negative scores with high absolute values). The reason is that users' high request rate leads to high negative values in most of attributes. We observe that the largest and lowest negative score in the common attack are -7485.65 and $-2.60E + 06$, respectively. In the meek attack, all malicious connections do not get negative scores and moreover, the absolute value of negative scores is much lower than the common attack. We observe the largest and lowest negative score in the meek attack are 0 and -2585.3, respectively.

Figure 5.c shows bandwidth rate occupied by legitimate traffic and attack traffic during the meek attack (for the common attack we have the similar situation). At time 14:03, the attack is started and the server goes to the freeze mode. One minute (score interval) after starting the attack, the control unit drops enough number of connections with negative scores such that bandwidth rate drops below threshold2. At this time (14:04) the server exits the freeze mode and accepts the new connections. The control unit replaces the established connections with lowest scores by new connections until the bandwidth drops below threshold3. Then the duty of control unit terminates. During 14:03 and 14:04, the bandwidth rate of good traffic slowly decreases because some legitimate users normally leave the system.

To calculate the percentage of false positive (FP), false negative (FN), the percentage of legitimate connections for possible false positive (PFP), R_L and R_A, we repeat the experiments 30 times during different times of a day. On average 6.3% of malicious connections do not get negative score in the meek attack; while all malicious connections get negative score in the common attack. So, the percentage of false negative and R_A is 6.3%. On average 24% of legitimate connections get negative score. Finally, on average, 2% of legitimate connections get score lower than -10 (the drop threshold); thereby, 2% of legitimate connections could be candidate for possible drop without asking them to solve the CAPTCHA tests.

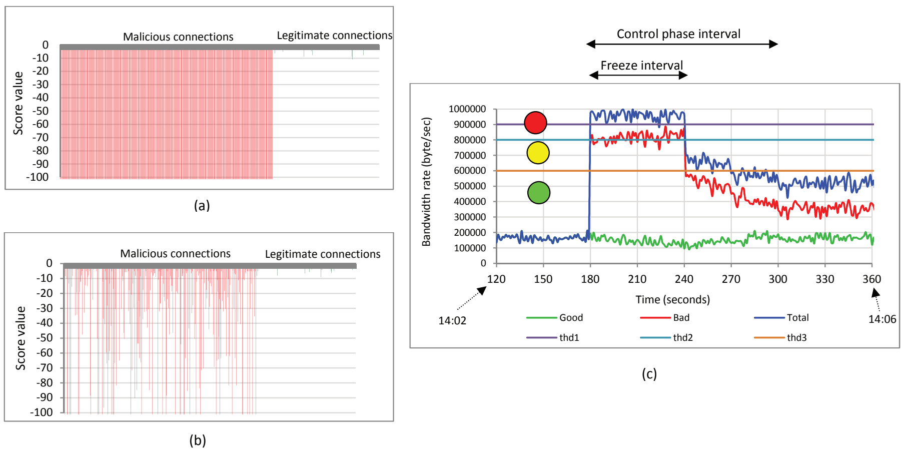


Fig. 5: Experiments: (a) score distribution in a common attack, (b) score distribution in a meek attack and (c) bandwidth rate occupied by good traffic and attack traffic

6. Conclusion

This paper proposes the ConnectionScore scheme against application-layer DDoS attacks. In this scheme, connections get score based on their behaviour. With a high probability, the connections which get lower scores are malicious connections; thereby the server retakes bottleneck resources from them. Our experimental results on Emulab environment indicate that the ConnectionScore scheme can effectively handle application-layer DDoS attacks for both common and meek attacks. We believe that the administrators of web-sites do not need to annoy their users by forcing them to solve CAPTCHA tests. Moreover, CAPTCHA tests are not so reliable. Therefore, the ConnectionScore scheme can be effectively considered as an alternative technique to handle application-layer DDoS attacks.

References

[1] T. Peng, C. Leckie, K.Ramamohanarao, Survey of Network-Based Defense Mechanisms Countering the DoS and DDoS Problems, ACM Computing Surveys 39 (1).

[2] C. Douligeri, A. Mitrokotsa, DDoS attacks and defense mechanisms: classi?cation and state-of-the-art, Computer Network 44 (2004) (2004) 643–666.

[3] W. G. Morein, A. Stavrou, D. L. Cook, A. D. Keromytis, V. Misra, D. Rubensteiny, Using Graphic Turing Tests To Counter Automated DDoS Attacks Against Web Servers, in: Proceedings of the 10th ACM conference on Computer and communications security, Washington, DC, USA, 2003.

[4] J.-F. Podevin, Telling humans and computers apart automatically, COMMUNICATIONS OF THE ACM 47 (2) (2004) 57–60.

[5] L. O. Caum, Why is CAPTCHA so annoying?, http://lorenzocaum.com/blog/why-is-captcha-so-fing-annoying/ (2011).

[6] J. Fraser, Why you should never use a CAPTCHA, online aspect, http://www.blogopreneur.com/2007/04/02/captchas-are-annoying/ (2010).

[7] G. Mori, J. Malik, Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA, in: Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Madison, Wisconsin, 2003.

[8] E. Athanasopoulos, S. Antonatos, Enhanced CAPTCHAs: Using Animation to Tell Humans and Computers Apart, in: Proceedings of Communications and Multimedia Security, 2006, pp. 97–108.

[9] H. D. Truong, C. F. Turner, C. C. Zou, iCAPTCHA: The Next Generation of CAPTCHA Designed to Defend Against 3rd Party Human Attacks, in: Proceedings of IEEE International Conference on Communications, Kyoto, Japan, 2011.

[10] T. Yatagai, T. Isohara, I. Sasase, Detection of HTTP-GET flood Attack Based on Analysis of Page Access Behavior, in: Proceedings of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, 2007, pp. 232–235.

[11] T. Thapngam, S. Yu, W. Zhou, G. Beliakov, Discriminating DDoS Attack Traffic from Flash Crowd through Packet Arrival Patterns, in: Proceedings of 2011 IEEE Conference on Computer Communications Workshops, 2011, pp. 969–974.

[12] Y. Xie, S. Yu, A Large-Scale Hidden Semi-Markov Model for Anomaly Detection on User Browsing Behaviors, IEEE/ACM Transactions on Networking 17 (1) (2009) 54–65.

[13] M. Kantardzic, Data Mining: Concepts, Models, Methods, and Algorithms, second edition Edition, IEEE press, 2002.

[14] H. Beitollahi, G. Deconinck, ConnectionScore: A Statistical Technique to Resist Application-layer DDoS Attacks, Tech. Rep. 01-2012-0130, Electrical Engineering Department, University of Leuven, Belgium, http://www.esat.kuleuven.be/electa/publications/fulltexts/pub_2313.pdf (2012).