Research on Counter Http DDoS Attacks based on Weighted Queue Random Early Drop

Guo Rui[1] , Chang Guiran[1] , Hou Ruidong[2], Baojing Sun[2], Liu An[2] and Bencheng Zhang [2]

[1.] College of information science and engineering, Northeastern University, Shenyang China

[2.] Electronic scouting and commanding department, College of Shenyang artillery Shenyang, China

Abstract. This paper proposes a new approach, called Weighted Queue Random Early Drop admission control, which protects small and medium online business Web sites against HTTP DDoS attacks. Weighted Queue Random Early Drop is used to compute dropping probability to avoid bursty traffic. Weighted Queue scheduler is adopted to implement access rate limit. The feasibility and effectiveness of our approach is validated by measuring the performance of an experimental prototype against a series of attacks. The advantages of the scheme are discussed and further research directions are given.

1 Introduction

Denial-of-Service (DoS[1]) attacks use legitimate requests to overload the server and make it hang, crash, reboot, or do useless work. The target application, machine, or network spends all of its critical resources on handling the attack traffic and cannot attend to its legitimate clients. Both DoS and DDoS are a huge threat to the operation of Internet sites, but the DDoS[2,3] problem is more complex and harder to solve.

There are mainly three defense approaches: traceback[4]—with the increase of zombies this approach will be invalidated rapidly. Filtrate[5]—because this method requires the communication company and lots of routers to participate, the filter must be open at all times, and the approach is too costly. Throttle[6]—legitimate data stream will be limited because too many data streams converge at a central point. Thus, based on these three methods, three distinct defense approaches emerge: gateway defense, router defense and computer defense. However, with the developing of the DDoS technology, there is increasing room for malicious attackers, such as HTTP DDoS, which appear like normal users and thus nefariously competes for limited critical resources, which crash the server.

Though DDoS defense approaches are inclined to be based on routers and gateway, especially for some international websites, this method is cost ineffective for mini websites. How, then, should mini websites be protected from roboticized attacks? We propose an innovate new method: combining the Turing test with Weighted Queue

Rui G., Guiran C., Ruidong H., Sun B., An L. and Zhang B. (2007).
Research on Counter Http DDoS Attacks based on Weighted Queue Random Early Drop.
In *Proceedings of the 5th International Workshop on Security in Information Systems*, pages 73-81
DOI: 10.5220/0002355000730081

management while filtering connection attempts that do not successfully pass our test and examination.

2 Admission Control Strategy

2.1 Turing Test

In our test, we consulted the English mathematician Turing and referenced the Turing test [7] (figure 1). As we all know, a similar Turing test validation code is widely applied in BBS and e-mail registration in order to effectively differentiate the host computer between people and zombies. Although computerized recognition technology is not as reliable as a live human operator, we can predict that the Turing test mechanism will perform well in the defense of DDoS.
When the IP address fails to pass our Turing test, we will restrict the user within 10mins and immediately place the IP on our running blacklist. This solves the problem of IP address substitution by worms or other automatic attacking gadgets that put actual users out of service from the internet.

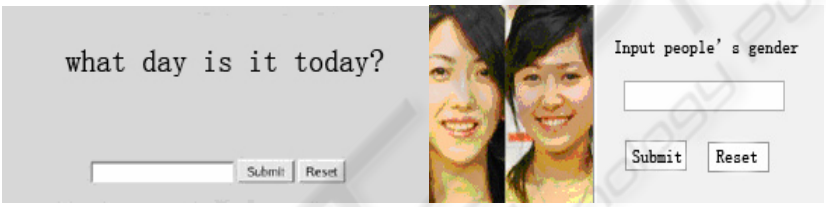

Fig. 1. Turning test.

2.2 The Basic Ideas for the Testing and Restricting Mechanism

Under most circumstances, we do not apply the Turing test except for any IP addresses that are on the blacklist. When an unknown request comes, we simply check the IP address with against our blacklist. If there is a match, we drop the request and cut off the response within 10 minutes. If the IP address is not on the blacklist, we allow the user to proceed on to the admission control phase. Additionally, we set threshold parameters for the server under certain categories, such as CPU, EMS memory, hard disk, interface utilization rate, etc. Once any of these bottleneck resources reaches the threshold, the server will be set into the admission control state, and the Turing test will be started up to ensure authentication. The client will be ordered to correctly pass the Turing test for establishing connection. The number of failed attempts will also be recorded to check for zombies.

The authentication will mainly be ordered when the server parameters overload the default threshold value. Legitimate users will either answer the questions accurately on the first try or manually apply for another Turing test, while worms and zombies cannot answer the questions appropriately, but will continually send requests. From differentiating between these two distinct behaviors, the server will accurately

distinguish live people from zombies sending linking requests.

We intend to apply the Turing test into the HTTP connection with a certain probability (for instance, once the CPU usage rate exceeds 70%) when the web server is running under the "admission control state." Under such circumstances, only if the Turing test question is correctly answered will the service be continued. Otherwise, the IP address will be put back on the blacklist. (Our implementation limits users to only ten attempts at authentication through the Turing test).

2.3 The Server Accessing Strategy under Admission Control Mode

Under admission control mode, we apply a Turing test to inspect all incoming HTTP requests. By amending the TCP/IP Protocol stack[8,9,10] (Figure 2), we guarantee that no resources are allocated before the clients respond correctly. As the picture shows, two packages will be sent, including one SYN ACK and an HTTP request data package. Instead of setting up a socket to finish the TCP 3 time-handshakes, the server will ignore the request and combine a Turing test with one FIN, waiting for the client to respond. The Turing test will be sent again if the response does not contain the exact answer. On the other hand, once the question is accurately answered, the TCP connection will be established shortly after. A cookie will be given only when the following three conditions are met: (1) the user correctly solves the Turing test (2) the user returns the answer to the requested server (3) the server checks out the responder information and ensures that it was engendered within three minutes. The client can use the authorized cookie to visit resources in the server, and the server will validate both the hash and the limited time of each cookie. The server side also controls the number of HTTP request connections from the cookie under a preset maximum amount (in our implementation, we have set this number at 10). It would be inconvenient to inspect each HTTP request from legitimate users once the connection has been established, so an authorized cookie will be delivered that allows a legitimate user to visit the server without any authentication check. (We set it to 20 minutes in the implementation)

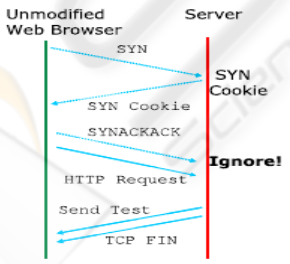

Fig. 2. Modified TCP protocol stack.

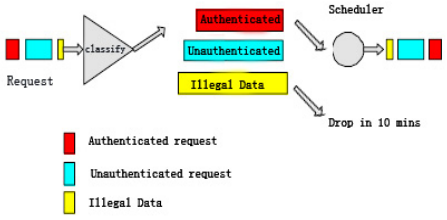

Fig. 3. Testing and restricting mechanism.

3 The Implementation of Admission Control

Admission control is the core of DDoS defense. Primarily, we hypothesize that the attacker would overload all of the resources, ultimately leading to a denial of service. We improved the Weighted Queue management Algorithm method based on the former algorithm. We used Weighted Queue Random Early Drop (WQRED), which enhanced throughput noticeably. In the former disposal, we dropped legitimated connections and suspect requests with the same drop rate. This indiscriminate method places subjects all users—legitimate and illegitimate—to the same limitation of resources. Weighted Queue Random Early Drop (WQRED) effectively resolves this irrationality (Figure 3).

3.1 Weighted Queue Random Early Drop Management Algorithm

The fundamental idea is this: the server makes use of a connection recorder to differentiate different data streams. Requests that cannot provide a correct answer to a Turing test after ten attempts are dropped. Within the ten validation attempts, the placement within the queue will be lowered. If a request passes the authentication, then the connection requests will be placed in another queue. The position will be higher in this authenticated queue. The server then sustains two queues to restrict data stream and resource utilization rates with different weights.

We set a recorder for every connection. When a new request is received, we handle it in three different ways:

1) If the IP is on the blacklist, we drop the requests that do not correctly solve the puzzle, cutting off their service within 10 minutes.

2) If the connector can answer correctly, we use a random dropping algorithm.

We setup a queue for legal data stream and make use of a weighted moving average algorithm to figure out the length of the average queue (L). Once a segment receives and calculates the length of the average queue, it is compared with the two thresholds: the maximum length and minimum lengths of the average queue. When the recalculated average length is smaller than the minimal parameter, the arrived data packages will be inserted at the tail of the list, and the discarding-package probability p will become 0. When it is bigger than the maximum parameter, the package will be discarded, and the probability p will be set to 1. The received packages will also be discarded with a certain probability p whenever the average list length is between the Min and Max (p is the linear function of the average list length). Figure 4 shows the relationship between the authenticated package drop rate *p* and the two thresholds.

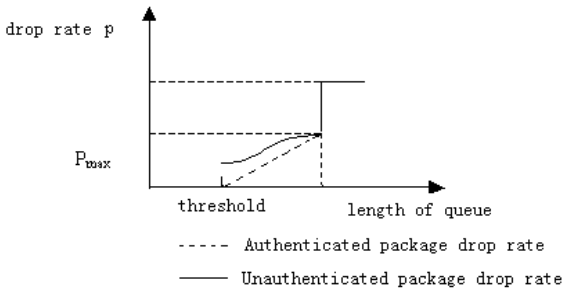

Fig. 4. Request package drop rate under admission control state.

We use a weighted moving average algorithm to calculate the average list length *L* and the package dropping probability p, and the maximum dropping rate is p_{max}. The following shows the computing method of L:

$$L = (1-\delta) * (\text{old } L) + \delta * (\text{current queue}) \quad (1)$$

δ is a number between 0 and 1. Once δ becomes small enough, instead of being impacted by the short time burst data, L will depend on the permanently changing tendency of the length of the queue.

There are some improvements for the calculation of dropping probability p in order to homogenize the package dropping alteration to a higher degree. This is achieved by computing the transitional dropping probability temp with the previously introduced algorithm in the first place:

$$temp = p_{max} \times (L - Min) / (Max - Min)$$

The dropping probability is then computed based on the following equation:

$$p = temp / (1 - count \times temp) \quad (2)$$

Count is a variable representing the number of data packages that enter into the list. Generalizing from this equation, we can conclude not only that the dropping probability p relates to the average queue length, but that it also increases with the enlargement of preserved data packages (count) in that queue. Doing so effectively avoids the congregation of data abandonment and precludes the possibility of DoS when the attack packages accumulate to a certain degree. The algorithm for the dropping probability is as follows:

for each packet arrival calculate the average queue size L

```
if min < =L < =max
          calculate probability p  base on (2)
          with probability p drop the arriving packet
else if max< =L
          drop t h e arriving packet
```

The avoidance of data abandonment (including by authorized requests) once the server is burdened with a huge number of attacks is the essential goal for our algorithm. The algorithm drops packet at fairly evenly spaced intervals. It effectively dismisses the global synchronization of legal users and enhances efficiency as well as resource availability, even without DDoS attacks.

3) For any suspicious data requests that do not fit under the previously mentioned categories, the initial step is to find the legal dropping probability p. Based on p, calculate suspicious data dropping probability p by the server's load with the following formula:

$$p=p_{max}/2 \ (\ \sin(x-\prod/2)+1 \ ) \tag{3}$$

The basic idea of this formula is that when the average queue size is close to the minimum threshold (meaning the server doesn't load too much) p is lowered to boost throughput. When the average queue size is close to the maximum threshold (meaning the server nearly overload) p is raised to boost throughput, avoiding overloading. Following this method provides different PRI for legal and illegal requests. We can thus get steady performance and lower drop rates.

The following shows the computing algorithm:

```
If Min<=avg<=Max
          If pass turingtest
          calculate probability p  base on 2)with probability p
                                        drop the arriving packet
          Elseif  during turingtest
                    calculate probability ps  base on 2)3)with
                       probability ps drop the arriving packet
          Else    turingtest>10 times
                  drop t h e arriving packet  without puzzle
                   turning test in 10 mins
Else avg>= Max
          drop t h e arriving packet ;
```

In any case, the drop rate of unauthenticated packages is much higher than that of authenticated packages. At least, it doubles the drop rate of authenticated packages. This avoids biases and assigns authenticated and unauthenticated packages with different PRI.

The admission control strategy of DDoS attack is supposed to be consisted of three independent algorithms: 1. An algorithm calculating the average length of the queue. 2. An algorithm for computing the drop probability of those packages. 3. An algorithm for adjusting the drop probability of those unauthenticated packages.

The first algorithm determines the degree of burstiness that will be allowed in the server. The second algorithm allows for packets to be dropped at fairly evenly spaced intervals in order to avoid biases and global synchronization. It also allows for packets to be dropped with sufficient frequency to control the average queue size. The third algorithm causes the server to assign resources accordingly based on suspicious request validation, thus enhancing performance.

4 Performance Evaluation and Comparison

The web server is a standard Pentium IV 3 GHz Linux machine with 1GB of memory and 100Mbps local Ethernet running on top of a modified Linux 2.4.10 kernel. We modified the kernel code, mostly in the TCP/IP protocol stack, which accomplished

the DDoS protector. For evaluating our system, we launch attacks from 100 different zombie machines using different port ranges to simulate multiple attackers per zombie. Each node simulates up to 1000 zombies, which results in a total of 100,000 attack clients. At the same time, we use Webbench, published by Lionbridge, to test server performance under attack. Webbench can test the performance of different services under controlled hardware conditions, as well as of the same service running on different hardware. Therefore, it is easy to tell the difference in performance between a base server and a WQRED server. We use WebBench static standard test suites to test throughput and response time (Figure 5), and we use dynamic standard test suites to test CPU utilization rate.

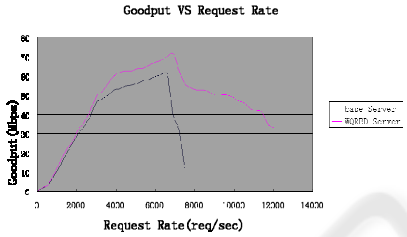

Fig. 5. Throughput VS Request Rate.

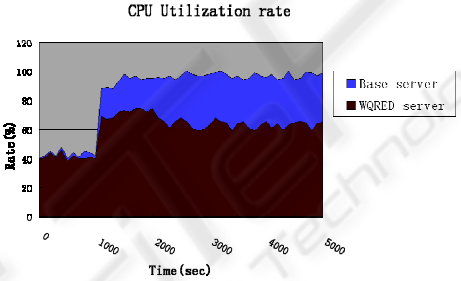

Fig. 6. CPU utilization rate and Time.

Figure 6 shows two servers running for both normal requests and attackers. It displays the throughput of legitimate users at different attack rates for both a base server and WQRED version. For the base server, when the throughput is very small, throughput and request rate have a linear relationship. The delay increases very slowly, but after the Knee is crossed, the throughput increases slowly, and the delay increases quickly. After the cross cliff, throughput decreases dramatically, the average response time experienced by legitimate users increases dramatically, and the server is overloaded and crashes. By contrast, the average response time of users accessing DDoS protector server is unaffected by the ongoing attacks. Only at the beginning do both of the servers have the same curve character. After crossing the cliff, the DDoS protector server won't crash, but it will begin to filter attack requests while continuing to serve the authenticated client.

Usually, the CPU can dispose of much faster than request submissions, but the CPU has to authenticate the client, so the CPU is a bottleneck.

The CPU utilization rate = throughput*service time

so our DDoS protector keeps the CPU utilization rate below 75% under attack. We can keep the CPU utilization rate below 60% after starting up the authentication mechanism.

5 Conclusions

The defense mechanism of DDoS attacks, particularly the multi-based, multi-approached and diversified flow method of offensive artifice, simulating the competition of legal users, inhabits a keystone and difficulty in the internet security arena, especially for the mini websites. This dissertation discusses and implements the Counter HTTP DDoS Attacks based on Weighted Queue Random Early Drop.
Our mechanism is characteristically distinct from current methods:

(1) Utilizes few resources and does not require participation from other routers. In general, requires nothing from the internet or the management services of ISP.

(2) Allows for simple and convenient updating of the Turing test. A few shares of restriction codes as well as the amendment of protocol stacks are the only renovations needed for withstanding DDoS without any negative impact on the clients.

(3) Optimize the web flow. Enhance the server's efficiency by precluding and dismissing the overall current abruptness of ordinary flow,

All in all, allocating the server's resources to both the validation and service components with more efficiency, and applying the Turing test to larger websites for DDoS defense are voids we are seeking to fill in this sector of internet security.

References

1. Jelena Mirkovic, Sven Dietrich, Internet Denial of Service: Attack and Defense Mechanisms, Prentice Hall PTR, December 30, 2004,1 - 400
2. Siris VA, Application of anomaly detection algorithms for detecting SYN flooding attacks In: Regency H, ed. Global Telecommunications Conf. (GLOBECOM 2004). Dallas: IEEE, 2004. 2050-2054.
3. Li W, Wu LF, Hu GY. Design and implementation of distributed intrusion detection system NetNumen. Journal of Software, 2002,13(8):1723-1728
4. Sung M, Xu J. IP traceback-based intelligent packet filtering: A novel technique for defending against Internet DDoS attacks. IEEE Trans. on Parallel and Distributed Systems, 2003, 14(9):861-872.
5. A. Chandra and P. Shenoy. Effectiveness of dynamic resource allocation for handling Internet, University of Massachussets, TR03-37, 2003.
6. Liang F, Yau D. Using adaptive router throttles against distributed denial-of-service attacks. Journal of Software, 2002,13(7): 1120-1127
7. Morein, W.G., Stavrou, A., Cook, D.L., Keromytis, A.D., Misra, V., Rubenstein, D.: Using Graphic Turing Tests to Counter Automated DDoS Attacks Against Web Servers. In: Proceedings of the 10th ACM International Conference on Computer and Communications Security (CCS). (2003) 8-19.
8. S. Kandula, D. Katabi, M. Jacob, and A. Berger. Botz-4-sale:Surviving organized DDoS attacks that mimic flash crowds. In USENIX NSDI, May 2005.

9. Thomas R, Mark B, Johnson T, Croall J. NetBouncer: Client-Legitimacy-Based high-performance DDoS filtering. In: Wermer B, ed. DARPA Information Survivability Conf. and Exposition 2003. Washington: Institute of Electrical and Electronics Engineers, Inc., 2003. 14-25.
10. 10 Yu Chen, Kai Hwang, Yu-Kwong Kwok, "Filtering of Shrew DDoS Attacks in Frequency Domain," lcn, pp. 786-793, The IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05)l, 2005