情報保護學會論文誌
第 21 卷 第 1 號, 2011. 2

사용자 의도 기반 응용계층 DDoS 공격 탐지 알고리즘

오 진 태,[1†] 박 동 규,[2] 장 종 수,[1] 류 재 철,[3‡]
[1]한국전자통신연구원, [2]순천향대학교, [3]충남대학교

A Novel Application-Layer DDoS Attack Detection Algorithm based on Client Intention

Jin-tae Oh,[1†] Dong-gue Park,[2] Jong-soo Jang,[1] Jeacheol Ryou[3‡]
[1]ETRI, [2]SoonChunHyang University, [3]Chungnam National University

요 약

서버의 응용계층에 대한 DDoS 공격은 매우 적은 량의 패킷으로 효과적인 공격이 가능하며, 공격 트래픽이 정상 트래픽과 유사하여 탐지가 매우 어렵다. 하지만 HTTP 응용계층 공격 트래픽에는 사용자 의도에 의한 특성이 있음을 찾았다. 정상 사용자와 DDoS 공격자는 동일하게 TCP 계층에서 세션을 맺는다. 이후 최소 한번의 HTTP Get 요구 패킷을 발생한다. 정상적인 HTTP 요구는 서버의 응답을 기다리지만 공격자는 Get 요청 직후 세션을 종료한다. 이러한 행위는 사용자 의도에 의한 차이로 해석할 수 있다. 본 논문에서는 이러한 차이를 기반으로 응용계층 분산서비스 거부 공격 탐지 알고리즘을 제안하였다. 제안된 알고리즘은 정상 네트워크와 봇 기반 분산서비스거부 공격 툴에서 발생한 트래픽으로 실험되었으며, 거의 오탐 없이 HTTP-Get 공격을 탐지함을 보여 주었다.

ABSTRACT

An application-layer attack can effectively achieve its objective with a small amount of traffic, and detection is difficult because the traffic type is very similar to that of legitimate users. We have discovered a unique characteristic that is produced by a difference in client intention: Both a legitimate user and DDoS attacker establish a session through a 3-way handshake over the TCP/IP layer. After a connection is established, they request at least one HTTP service by a Get request packet. The legitimate HTTP user waits for the server's response. However, an attacker tries to terminate the existing session right after the Get request. These different actions can be interpreted as a difference in client intention. In this paper, we propose a detection algorithm for application layer DDoS attacks based on this difference. The proposed algorithm was simulated using traffic dump files that were taken from normal user networks and Botnet-based attack tools. The test results showed that the algorithm can detect an HTTP-Get flooding attack with almost zero false alarms.

Keywords: Application-layer DDoS attack; layer-7 attack HTTP-Get flooding; CC attack; Botnet

I. INTRODUCTION

Various Internet services are available to users due to the development of communication network and electronic technologies. However, such development has also introduced various hacking tools. These tools can disturb a system's provision-related services or cause the system to malfunction. Hacking tools have been advanced to provide various types of

접수일(2010년 8월 26일), 게재확정일(2010년 9월 27일)
† 주저자, showme@etri.re.kr
‡ 교신저자, jcryou@home.cnu.ac.kr

attacks for economic profit. One of the well-known cyber attacks is a distributed denial-of-service (DDoS) attack[1]. A DDoS attack causes a victim to malfunction and no longer supply normal service. DDoS attacks have been growing stronger through the abuse of botnets[2], which are network groups of zombie.

Several DDoS detection algorithms have been developed to detect an attack. However, most of these algorithms have been limited to detecting and blocking network-level DDoS attacks such as SYN flooding[3][4]. But DDoS attacks targeting at the network-layer are giving way to sophisticated application-layer (layer-7) attacks. DDoS attacks are increased against application-layer services, particularly HTTP Web services. In one instance, an online merchant employed the "DDoS Mafia" to launch an HTTP flood on his competitors' Web sites after a regular SYN flood failed to bring the sites down.

DDoS attacks against an application-layer disturb servers providing application-layer services. Typically, DDoS prevention systems employ a rate limit to reduce the number of packets arriving at the Web server. However, a rate limit can result in false alarms. Some DDoS attack packets are still input into a related server by a false negative. A detection system's false positive, i.e., legitimate packets that are interpreted as malicious and hence dropped, is detrimental to real users of the system. In other words, DDoS prevention technologies cannot protect a related server precisely due to such false alarms. Signature-based packet filtering is also used to protect against DDoS attacks. However, such filtering cannot prevent an attack in real time because attacker can change their payload easily.

In previous works, the research starts from the assumption that an attacker's traffic does not have unique characteristics that can be distinguished from legitimate traffic. However, we have found that attack traffic has a unique characteristic that is generated slightly within legitimate HTTP traffic. Therefore, a new application-layer DDoS detection algorithm is proposed that can detect an HTTP-Get flooding attack with almost zero false alarms.

The remainder of this paper is organized as follows. Related works are reviewed in section II. In section III, our proposed algorithm, which detects HTTP-Get flooding attacks based on client intention, is introduced. Results of a performance evaluation are shown in section IV. The implementation and test results are given in section V. And finally, we offer a conclusion including a brief summary of our proposal in section VI.

II. RELATED WORK

In prior works, simple threshold-based application-layer DDoS attack detection methods have been proposed. These proposed methods detect HTTP-Get flooding attacks that increase Web server load. They limit traffic bandwidth whether it comes from a legitimate user or an attacker[5][6][7][8]. However, false alarms can be generated by a bandwidth limit.

Another detection algorithm detects a client as an attacker when the same page is requested more than a threshold number during a given time[9]. The threshold number was determined using historical data, and was compared with current data to improve the detection rate[10]. However, all detection methods based on a threshold have false positives and false negatives depending on the threshold value and operational conditions.

A pattern classification method has also

been used to detect application-layer DDoS attacks. An HTTP-Get flood detection method based on a Web page access behavior analysis was proposed[11]. The proposed method used two detection algorithms. The first algorithm detects an HTTP-Get flood when clients request pages same sequence order. Second, it detects a client as an attacker when the pages are browsed very briefly. However, attackers can change their page requests very easily, and thus the proposed methods cannot detect attacks precisely.

A Web-browsing pattern-based attack detection method was also proposed[12]. The method uses Hidden Semi-Markov Models (HSMM) to describe Web-browsing patterns and detect HTTP-Get flooding attacks. The method uses a large number of legitimate request sequences to train an HSMM, and then calculates the interval of likelihood for legitimate sequences against the norm. After obtaining a normal likelihood interval, the model can be used in detecting an HTTP-Get flooding attack. However, it creates false alarms because the reasonable interval of the average log likelihood can contains the abnormal user's traffic.

A DDoS-Shield was proposed to detect DDoS attacks based on a counter-mechanism[13]. This paper considered sophisticated attacks that were protocol-compliant, non-intrusive, and utilized legitimate application-layer requests to overwhelm system resources. The method consists of a suspicion assignment mechanism and DDoS-resilient scheduler. A suspicion assignment mechanism used a session history to assign a suspicion measure to every client session, and a DDoS-resilient scheduler decided which sessions were allowed to forward requests depending on the scheduling policy and scheduler service rate. However, the proposed method also provided mitigation against attack traffic. It did not provide precise protection methods. Therefore, a new application-layer DDoS detection algorithm is needed to protect a server from an attack with almost zero false alarms.

III. PROPOSED DETECTION ALGORITHM

The detection researches in previous works start from the assumption that traffic from a layer-7 attacker does not have unique characteristics that could be distinguished from legitimate traffic. A layer-7 attack based on the TCP protocol cannot fake the attacker's IPs because it has to establish a session through a 3-way handshake. An attack can be blocked by the ACL when the attacker's IP is detected. For this reason, we focused on the session, where we found that the difference of access intention will be expressed. Attack traffic has a unique characteristic that is rarely generated in legitimate HTTP user traffic. The characteristic was discovered in various DDoS tools including NETBOT, Black-Energy, Fungwon, and zombie codes that provoked the DDoS attack in Korea on the 7th of July, 2009. An algorithm was proposed to find the application-layer DDoS attack. Usually, a false alarm is generated by the probability distribution of the detection model. In our algorithm, entire sessions were mapped into two states based on the client's access intention. The proposed algorithm is the first approach to detect an application-layer DDoS attack by classifying sessions through user intention. The proposed algorithm is shown in section 3.1. The difference in client intention between a legitimate user and an attacker is explained in section 3.2 through the viewpoint of socket programming. The proposed algorithm is reviewed in section 3.3 based on the HTTP protocol standard.

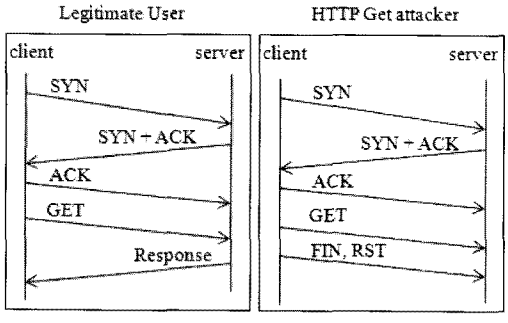
3.1 Proposed Layer-7 DDoS attack detection algorithm based on client intention

To start HTTP service, both a legitimate user and DDoS attacker establish a session through a 3-way handshake over the TCP/IP layer. After a connection is established, they request at least one HTTP service by a Get request packet. The procedures are exactly the same up to this point. The legitimate user waits for the server's response. After the server replies, the next step is processed. However, an attacker tries to terminate the existing session right after the Get request. The different action can be interpreted as a difference in client intention. A legitimate user's intention is to be provided a service from the server. But an attacker's intention is to disturb the service. The attacker is not interested in data from the server, so the attacker does not need to wait for such data. The difference between these sessions is shown in [Fig. 1].

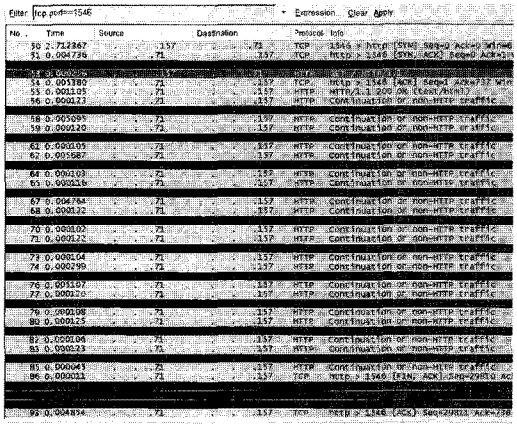
The left box of [Fig. 1] is a sample of a legitimate user's session. They established a session by a 3-way hand shaking. They then generated an HTTP-Get request packet, and waited until the server downloaded the requested pages. The session was terminated by the server or client after the server's response packets. This behavior appears when they have the intention of receiving a service from the server. The right box of [Fig. 1] shows a sample of an HTTP-Get attacker's session.

The session states were exactly the same as in legitimate sessions until the HTTP-Get request. However, an attacker terminated the session with a Fin or Reset packet right after the Get request packet because they did not need the service.

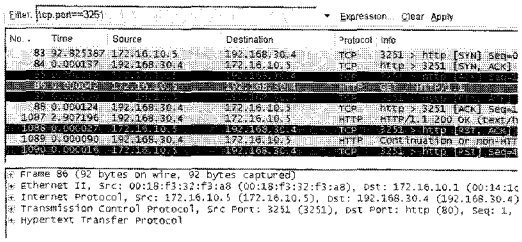
Examples of a packet dump are shown in [Fig. 2] and [Fig. 3]. The traffics in the figures were captured with a WIRESHARK packet dump program. The legitimate user's traffic dump in [Fig. 2] was captured in a real network. The addresses were erased for the sake of privacy. Packets with the same IP pair and a 1546 port number were selected and displayed. The session began with a Syn packet by a client. The session was established through a 3-way handshake. The client in the forth column generated an HTTP Get request packet. The server replied with 21 packets as a response to the Get request. The server started a session demolish sequence by sending a Fin packet, which is shown as sequence number 86 in [Fig. 2]. It is legitimate user behavior to wait until the request receives



[Fig. 1] The session state of a legitimate user and DDoS attacker.



[Fig. 2] The traffic dump of a legitimate user's HTTP-Get request.



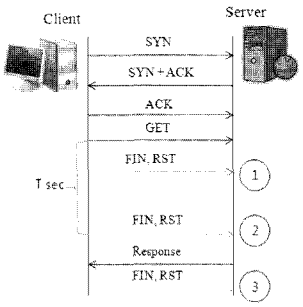
[Fig. 3] An HTTP Get flooding traffic dump from a NETBOT attacker.

a result from the server.

The dump file in [Fig. 3] was captured in our DDoS test network, while a NETBOT attacker generated attack traffic against a server.

The attacker had 172.16.10.5 as its IP address. The server's IP address was 192.168.30.4. The IP addresses were not hidden because the dump file was not taken from a real network. Port 3251 was selected from a number of sessions of the original traffic dump. The first column of the selected packet was a Syn packet to initiate a session by an attacker. The fourth packet was an HTTP Get request and the contents were displayed in the bottom box of the dumped image. The Fin packet was generated in the fifth sequence. The time gap between the HTTP Get request and Fin packet was only 30 microseconds. The server responded with an "HTTP/1.1 200 OK" packet after 2.9 seconds even though the attacker had started a termination sequence with the Fin packet. The packet was the server's answer packet for the previous HTTP Get request.

The session was closed normally by the reset sequences. Other DDoS attack tools that we collected had a similar packet sequence, and one of the traffic dump files is shown in this paper. The above explanation clearly shows that the DDoS attack session had a unique characteristic compared with the legitimate users. We call this a difference in client intention. We proposed an algorithm to detect an application-layer DDoS attack using this difference in client intention. Our algorithm is very simple. It checks whether the client terminates an established session before the server's response packet has been observed, and the source IP is then detected as an attacker. The proposed DDoS detection algorithm works properly for HTTP 1.x protocols. In this paper, we want to focus on HTTP Get flooding attack detection. The detection algorithm based on client intention can be expanded further.



[Fig. 4]. Possible Session Terminating scenarios by client

The [Fig. 4] shows possible session terminating point by clients. The point 3 that terminates the session after receiving the server's response packet is normal. But attackers can terminate the session at this point, then they have dependence on server.

If the server does not send response packet, then attack program can terminate the sessions after time expiration. Usually the attack program uses multi-thread to generate many requests. The point 1 is already explained. The session termination at point 2 is explained as the attacker programmed the attack code to wait certain time until session termination. In this case, if the server can not response quickly enough, then the result is same as point 1.

The hacker has no room to evade the session termination method unless he waits the session termination by server. But the waiting causes session exhaustion of the client, which limits attack packet generation against victim.

3.2 An analysis of intention from a socket programming viewpoint

Our proposed algorithm was introduced in the previous section. In this section, the difference of session behavior will be analyzed based on the viewpoint of socket programming that occurs from the discrepancy of user intentions. Most communication programs use a socket. The socket is an API (Application Programming Interface) that connects a TCP/IP layer with an application layer. Some OS (Operating Systems) allow accessing a network layer only through a socket due to security concerns. For communication programs using a socket, it is necessary to create a session for data communication in the application layer. After the end of data communication, the sessions have to be terminated.

A computer has many manageable resources such as a CPU, Memory, IO, Network, and so on. A socket is one such resource. Usually, a personal computer can generate around three-hundred sockets simultaneously. If the number of sockets created approaches a limitation, no more sockets can be created. If a legitimate code is run from the machine, it can utilize enough resources until all of the services are finished. However, a zombie code is a type of illegal program. It can use limited resources within a limited time to avoid being awakened by a user at the host. Since a zombie code is written as a socket, it is necessary to terminate the socket after requesting service in order to maximally utilize the socket resources. The session termination right after a GET request does not exhaust the socket resources of the machine. If new sockets are continuously created without termination, it is impossible to create a new socket very quickly because the socket resources of the client become exhausted.

3.3 An abnormal state analysis by the HTTP protocol standard

The difference in user intention was analyzed based on the viewpoint of socket programming in the previous section. In this section, an abnormal state termination will be reviewed using the HTTP protocol standard. The HTTP 1.0 standard is described as follows. In applications other than experimental programs, a client must establish a connection to the server before transferring a request message, and must terminate the connection to the server after the server transmits a response. The client and server must be aware that the connection may be terminated by a user operation, an automatic time out, or a program error. Further, the client and server must have the capability to perform a proper operation when the connection is terminated. A terminated connection means the deletion of a current request even though the connection is terminated by either or both sides. The standard defines a normal service termination as terminating a session after a server ends data communication. It also defines three abnormal service terminations.

1) Session termination by a user operation
2) Session termination by an automatic time out
3) Session termination by a program error

During our investigation with several DDoS attack tools, it was discovered that the time gap between a Get request and session termination by a client was very short. The time interval is only several tens of microseconds (μsec). There is very little chance to terminate a session by a legitimate user operation as mentioned in the first abnormal termination. If a user wants to terminate a session right after a GET packet, he/she must close the Web-browser or hit the ESC key to cancel the request within several tens of microseconds. However, it is almost impossible for a user to close a Web browser within such a short period of time.

In the case of a session termination by an automatic time out, a time out is generally set to be several seconds in length. Compared to several tens of microseconds, the automatic time out is very long. Therefore, it is almost impossible to terminate the session within several tens of microseconds by an automatic time out.

Finally, a session termination through a program error can be clearly analyzed as such, and is sometimes excluded from assumptions defining an abnormal session termination.

The above analysis shows that a session termination event by normal users does not happen frequently, and thus DDoS attack sessions are distinguished from legitimate users.

IV. PERFORMANCE EVALUATIONS

Our proposed algorithm was analyzed from various viewpoints in sections 3.1 to 3.3. We have confidence through this evidence that our intention-based detection algorithm produces almost zero false alarms. But to further prove the integrity of our algorithm, we need simulation results with real traffic, implementation, and test results. The possibility of a false alarm is shown using a real traffic dump analysis described in section 4.1. The performance evaluation results are shown in section 4.2 using a real traffic dump.

4.1 The possibility of a false alarm in a real network

To evaluate the proposed algorithm, the probability of a false alarm has to be calculated based on the mathematical model of the algorithm. Our proposed algorithm has two groups of sessions that are classified by user intention. We call the groups S0 and S1. The S0 group is a set of sessions that await service until the server provides it. The S1 group is a set of sessions that abandon a service request before receiving the server's first response packet.

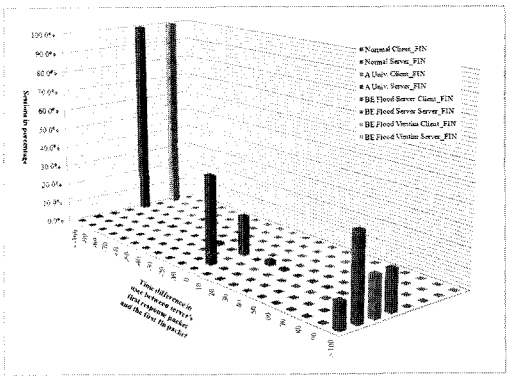
S0: a set of sessions that await server's response
S1: a set of sessions that abandon a request before receiving the server's first response packet

Legitimate users mainly generate S0 sessions. However, a legitimate user can generate an S1 session by hitting the ESC key or closing the Web browser when they can no longer wait. A Web user's waiting tolerance for information retrieval is generally around 2 seconds[14]. Most users abandon a request when the Web server does not download the requested pages within a tolerable waiting time. Also, S1 sessions for normal users were found in real traffic dump files. This means a false positive is not zero in a real network. However, S1 events on average do not exceed a hundred instances per year for the normal user. The detection model has to be

made using IP pairs because the proposed algorithm works on a session. The number of S1 sessions for a specific user and server pair is too small to compare with the number of S0 sessions. A reasonable probability distribution function cannot be made with a small number of samples. However, no user abandons the same page twice during a one second period because a legitimate user waits for the specific Web server to reply until the end of a tolerable waiting time. Therefore, we enhanced the proposed algorithm to detect a host as an attacker when it generates an S1 set twice in a one second period, and based on this, an almost zero false positive detection rate is possible. A false negative can be generated when an attacker uses an S0 session. When an attacker gives up an S1 session, its zombies consume more resources causing easy detection at the host.

4.2 Simulation results with real traffic

Two types of dump files were analyzed to evaluate the proposed algorithm. One was captured from a university backbone as a normal sample, and the other was captured from a DDoS zombie network as an attack sample. When a client generated an HTTP-Get request on an established session, the session was traced and the time gap between the server's first response and the session termination request by the client or server was logged. If the server's first response preceded the session termination request on both sides, then the time gap had a positive value. On the other hand, the time gap had a negative value when the session termination request packet was observed before server's first response packet. A false positive would only be generated when a user hits the ESC key or exits the browser before the first request



(Fig. 5) Analysis results of normal and attack traffic.

arrives from the server.

The analysis results are shown in (Fig. 5). The x axis displays the time difference in usec between server's first response packet and fin or reset packet from both sides. The y axis displays the number of sessions in percentage. In the (Fig. 5), eight colored lines are showed. The front 4 lines show the analyzed results of legitimate user traffic. These results mainly had positive delay values, and thus the server's first response preceded the session termination request. The rear four lines indicate the results of traffic generated from a Black-Energy DDoS tool. The test results had only negative delay values. The session termination requests by the client always come earlier than the server's first response. However, a few negative values were generated from the legitimate user traffic. False positive alarms were generated by these sessions. Even though their behaviors cannot be completely known, they abandoned their service request before the server's response.

V. THE IMPLEMENTATION AND TEST RESULTS

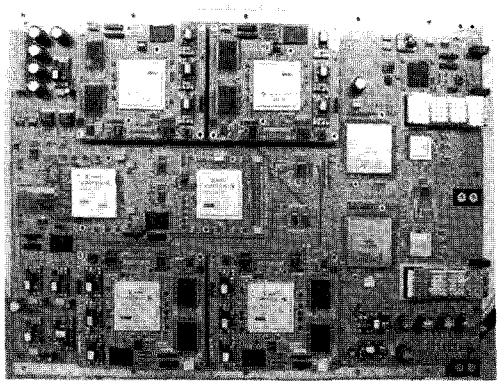
In this chapter, the implementation results are described for our proposed de-

tection algorithm. The test results are presented using real attack tools and replaying the dumped data. The implementation environments are described in section 5.1. The implementation results of a 20Gbps Anti-DDoS system using the proposed algorithm are shown in section 5.2. Finally, performance comparisons and discussions are given in section 5.3.

5.1 The implementation environments

A 20Gbps Anti-DDoS system named ALADDIN was developed at the end of 2009. The developed system board is shown in [Fig. 6].

The board was configured using two 10Gbps fiber optic interfaces with a MAC layer, which are shown in the right part of [Fig. 6]. It has one load balance chip to distribute the input packets to the DDoS detection engines. The load balance chip is shown in the center in the figure. The detection engines each process 5 Gbps of data per second. The detection engines were implemented as a daughter board type. The load balancer and detection engines were implemented using a grade-1 speed Vertex-5 Xilinx FPGA. Each engine had 4 external SRAMs for session, flow, and ACL tables. The chips were implemented using the verilog HDL language. The interfaces between the load balance and detection engines were implemented using 64-bit bus widths and a 100Mhz clock speed.



[Fig. 6] The implemented 20Gbps Anti-DDoS system board.

The detection engine processes packets using 32-bit bus widths internally. The engine was synthesized successfully at a clock speed of 178Mhz. The implemented SRAMs were operated at 150Mhz. However, 200Mhz SRAM is also available. The maximum performance of the detection chip is 5.69Gbps. A Synplify pro 9.4 synthesis tool was used for hardware synthesis. A Modelsim PE 6.4 simulator was used for the logic simulation. An Xilinx ISE 10.1 was used for mapping logic and routing the resources in the FPGA.

5.2 The implementation results

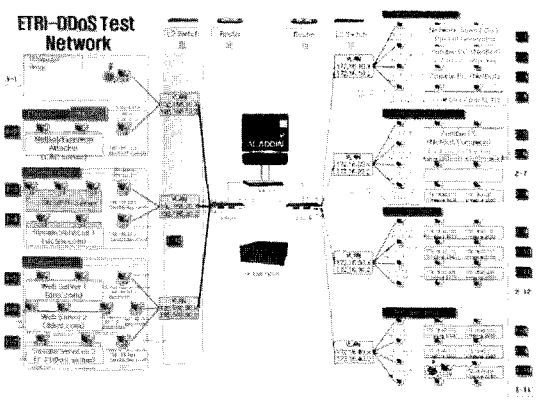
Our system was implemented using the proposed algorithm. The system also has a network-layer DDoS detection function, but this is beyond the scope of this paper. When an application-layer DDoS is detected, a proper response mechanism is needed to protect from the attack. Because the application-layer attacks cannot fake their source IPs, the proposed system uses an access control list (ACL) for the response. The ACL was implemented using an SRAM with a half-million entries. The complete ACL of the system included 2 Million entries that came from 4 different engines. Each ACL entry has a 6-bit counter that can count the number of abandoned requests during a one second period. And each entry has control bits to drop packets or log an event. The time interval and event threshold can be configured. The time interval is set to one second by default. The default event threshold is set to 2. The first event only activates an ACL entry, but the second event on the same flow will generate

an alarm and block the client's access to the server. However, the entry will be erased when no more events are generated during the time interval. A session tracer was implemented using 8 million simultaneous entries that trace the sessions in order to detect network-layer and application-layer DDoS attacks.

Our system was tested in a local DDoS testing network using various DDoS tools and test equipment including BreakingPoint. Our test environment is shown in [Fig. 7].

The network was configured with two racks, 23 servers, several layer-2 gigabit switches, a 10 gigabit switch with a 24-gigabit interface and two 10Gbit interfaces, and two routers. First, the system was tested using real DDoS attack tools including NETBOT under 8 Gbps of background traffic. Second, the system was tested using zombie codes that invoked the DDoS attack in Korea on the 7th of July, 2009. The system was tested using BreakingPoint equipment through amplified NETBOT traffic for large scale zombie attack scenarios. Also, a real traffic dump replay was used for checking false alarms.

At the beginning of the test, the counter threshold of the ACL was set to 1, which



[Fig. 7] The configuration of our DDoS test network.

detected a single attack session. Our system detected HTTP-Get and CC attacks without false negatives generated by the real attack tools and testing equipment. The system was tested during a period of more than a couple of months. We had one false positive alarm during a test with an ACL counter threshold of 1. This false positive alarm made us realize how false positives can be generated by a legitimate user. A Web server was attacked with a NETBOT DDoS attack tool through the network. However, our system could not protect the attack properly due to a bug in the hardware at that time. We tried to check the server's condition through Web browsing, but the server had already been overwhelmed by the DDoS attack. While waiting for the browser to answer during several seconds, an engineer hit the ESC key to cancel the request. A FIN packet was generated by the browser to terminate the session. Finally, the session termination request was detected as an attack by our system. Normal traffic load was needed to check for false positives in our system within a real network environment. Instead of a real reference site, a packet replay system was used to replay Pcap format traffic dump files. Traffic of 107 GB in size was dumped at a university in Korea from 2005.07.29, 14:03:38, to 2005.07.31, 02:43:52. It had a total of 322,540,183 packets with 1,296,654 established sessions. The dumped packet size was limited to 500 bytes. Our implemented system detected 19,856 HTTP-Get attacks that were false positives, and the false positive rate was 1.53%. The ACL counter threshold was set to 2to remove false positive alarms. The 107 GB of dumped packets were replayed again. This generated no more false positive alarms in our system. Our modified algorithm could detect an application layer attack that gen-

erated more than two HTTP-Get request packets in a one second period.

To prove the scalability of our system, a test for detecting 1.8 million zombies was successfully completed using BreakingPoint testing equipment. The NETBOT attacker-generated packets were captured and multiplied by the test equipment. All of the attackers' IPs were blocked in real time by our system.

5.3 Performance comparison and discussion

Even though the chance of abandoning a service request by a legitimate user is quite rare, we had a false positive alarm during our test. Also, false positives were found in our traffic dump analysis. Even though the false positive rate was very low, the automatic defense could not be enforced due to the false positives. A legitimate user does not terminate a session continually in such a short period. If someone terminates the same requests more than twice in a second, he/she has no intention to receive service from the server. Even though the server is overwhelmed by a DDoS attack, a legitimate user usually awaits a response from the server during a tolerable waiting time. Eventually, he/she closes the Web browser without receiving a result from the server. This situation is detected as a first session termination event by our system. If the user's intention is to be provided service, he/she will not terminate the session again within a one second period. If a second termination occurs, it should be interpreted as an attack. [Table 1] shows the comparison results for the performance differences of various detection methods.

All of the prior researches assumed that an attacker's traffic does not have unique characteristics that can be distinguished from legitimate traffic. As seen in the data in [Table 1], the previous methods were implemented by hardware with difficulty. It was also reported that the BOTNETs had more than a couple million zombies. However, previous methods did not mention the scalability.

[Table 1] Performance comparison of detection methods.

Detection methods	Model	Implementation	Scalability	False alarm
Detection of HTTP-Get flood attack (Takeshi Yatagai et al. 2007)	Correlation with browsing time	SW	limited	10%
Web based traffic anomaly (Jun Lv et al., 2007)	GLR	SW	limited	2.6%
	WGLR			0.69%
	EPD			0.35%
HTTP flooding detection method (Wei Zhou Lu et al., 2006)	HSMM	SW	scalable	6.6%
Proposed algorithm	Classify sessions by users' intention	HW or SW	scalable	not found yet

We suggest that the sessions be classified into two groups based on the intention of the service request. Our test results showed that our method did not produce a false alarm during various tests. High-speed implementation was possible using a very simple architecture, and the 1.8 million zombies detected shows the scalability of the algorithm.

VI. CONCLUSIONS

An application-layer DDoS attack can efficiently overwhelm a server with small amounts of traffic. However, detection is difficult because the traffic types are very similar to that of legitimate users. Previous researches have proposed detection algo-

rithms based on various approaches, but they did not find a false-positive-free solution. A unique characteristic that arises from the differences in client intention was found, and this characteristic is seldom generated in legitimate HTTP traffic. In this paper, an algorithm was proposed to detect an application-layer DDoS attack based on client intention. The proposed algorithm can reduce the uncertainty of the mathematical calculation of prior works. Client-intention-based DDoS attack detection can distinguish attack sessions from legitimate users with almost zero false alarms. A 20Gbps anti-DDoS system was implemented with our proposed algorithm. The possibility of high-speed hardware implementation was shown in this paper. The system was tested using existing DDoS attack tools in 8Gbps of background traffic. The system detected HTTP-Get flooding attacks including CC attacks with few false alarms during our test. An ACL with an event counter was implemented to remove false alarms. The modified algorithm did not generate false alarms during the testing procedure. The test results for detection of 2-million simultaneous zombies show the scalability of the proposed algorithm. A test in a real network will be prepared in the near future.

ACKNOWLEDGMENT

"This research was supported by the MKE(Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency)"(NIPA-2011-(C1090-1031-0005))

"The last author of this research was supported by the MKE(Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency)" (NIPA-2011-(C1090- 1031-0005))

REFERENCES

[1] Kargl, F., Maier, J., and Weber, M. "Protecting web servers from distributed denial of service attacks," In Proceedings of the 10th International World Wide Web Conference, pp. 130-143, 2001.

[2] Honeynet "Know your enemy:tracking botnets," Whitepaper. The Honeynet Project & Research Alliance. Feb. 2005. www.honeynet.org/index.html.

[3] Wang, H., Zhang, D., and Shin, K. G. "Detecting SYN flooding attacks," In Proceedings of IEEE Infocom 2002, pp. 1530-1539, 2002.

[4] Tuncer, T. and Tatar,Y. "Detection SYN Flooding Attacks Using Fuzzy Logic." Proc. Int. Conf. Information Security and Assurance (ISA'08), Washington, DC, USA, pp. 321-325. Apr. 24-26, 2008.

[5] http://www.topology.org/src/bwshare/README.html

[6] http://www.sakura.ad.jp/tanaka/apache/module/mod access limit.tar.gz

[7] Abdelsayed, S., Glimsholt, D., Leckie, C., Ryan, S., and Shami, S. "An efficient filter for denial of service bandwidth attacks," In Proceedings of the 46th IEEE Global Telecommunications Conference (Globecom'03), pp. 1353-1357, 2003.

[8] Gil, T. M. and Poletto, M. "MULTOPS: A data-structure for bandwidth attack detection," In Proceedings of the 10th Usenix Security Symposium. 2001.

[9] http://www.zdziarski.com/projects/mod evasive/

[10] Jun Lv, Xing Li, and Tong Li, "Web based Application for Traffic Anomaly Detection Algorithm," Second International

Conference on Internet and Web Applications and Services (ICIW'07), pp. 44-49, 2007.

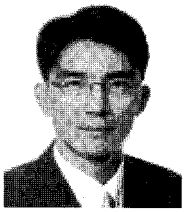
[11] Takeshi Yatagai, Takamasa Isohara, and Iwao Sasase, "Detection of HTTP-GET flood Attack Based on Analysis of Page Access Behavior," Communications, Computers and Signal Processing, pp. 232-235, 2007.

[12] Wei Zhou Lu and Shun Zheng Yu, "A HTTP Flooding Detection Method Based on Browser Behavior," IEEE Computational Intelligence and Security, 2006 International Conference on, vol. 2, pp. 1151-1154, Nov. 2006.

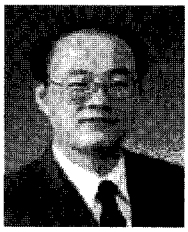
[13] Supranamaya Ranjan, Ram Swaminathan, Mustafa Uysl, Antonio Nucci, and Edward Knightly, "DDoS-Shield: DDoS-Resilient Scheduling to Counter Application Layer Attacks," IEEE/ACM Transactions on networking, vol. 17, no. 1, pp. 26-39, Feb. 2009.

[14] Fiona Fui-Hoon Nah, "A study on tolerable waiting time: how long are Web users willing to wait?," http://sigs.aisnet.org/sighci/bit04/BIT_Nah.pdf

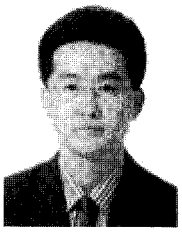
〈著者紹介〉



오 진 태 (Jintae Oh) 정회원
1990년 2월: 경북대학교 전자공학과 졸업
1992년 2월: 경북대학교 전자공학과 석사
2011년 2월: 충남대학교 컴퓨터공학과 박사
1992년 3월~1998년 2월: 한국전자통신연구원 선임연구원
1998년 2월~ 2001년: MINMAX tech. and Engedy Networks 부장
2001년 9월~ 2003년 1월: Winnow Tech. 공동창업, CTO, 부사장
2005년 3월~ 2009년 3월: 한국전자통신연구원 보안게이트웨이 팀장
2003년 3월~ 현재: 한국전자통신연구원 책임연구원
〈관심분야〉 정보보호, 고속 하드웨어 설계



박 동 규 (Donggue Park) 종신회원
1985년 2월: 한양대학교 전자공학과 졸업
1988년 2월: 한양대학교 전자공학과 석사
1992년 2월: 한양대학교 전자공학과 박사
1992년 2월~ 2003년: 순천향대학교 정보기술공학부 부교수
2004년~현재: 순천향대학교 정보통신공학과 교수
〈관심분야〉 네트워크 보안, 유비쿼터스 컴퓨팅 보안



장 종 수 (Jongsoo Jang) 종신회원
1980년 3월~ 1984년 2월: 경북대학교 전자공학과 학사
1984년 3월~ 1986년 2월: 경북대학교 전자공학과 석사
1997년 3월~ 2000년 2월: 충북대학교 컴퓨터공학과 박사
1989년 7월~ 현재: 한국전자통신연구원 책임연구원
2004년 1월~ 2008년 2월: 한국전자통신연구원 네트워크보안그룹 그룹장
2000년 7월~ 2003년 12월: 한국전자통신연구원 네트워크보안구조팀 팀장
2004년 1월~ 현재: 한국정보보호학회 이사(부회장)
2007년 1월~ 현재: 한국정보처리학회 이사
2008년 1월~ 현재: 한국전자공학회 통신소사이어티 이사
2006년 1월~ 현재: OSIA(개방형컴퓨터통신연구회) 이사
2006년 1월~ 현재: 대검찰청 디지털수사자문위원회 위원
2006년 1월~ 2008년2월: 행정안전부 전자정부서비스보안위원회 실무위원
2008년 5월~ 현재: 방송통신위원회 인터넷정보보호협의회 안전인터넷분과 위원
2005년 3월~ 2009년12월: 네트워크시큐리티포럼 임원
〈관심분야〉 정보보호, 전자공학

류 재 철 (Jaecheol Ryou) 종신회원
1988년 5월: Iowa State University 전산학과 석사
1990년 12월: Northwestern University 전산학과 박사
1991년 ~ 현재: 충남대학교 정보통신공학부 교수
1997년 ~ 현재: 한국정보보호학회 이사
2001년 ~ 현재: 국가정보원 정보보호시스템 인증위원회 위원
2003년 ~ 현재: 인터넷침해대응기술연구센터 센터장
〈관심분야〉 IPTV 보안, 인증이론 및 시스템, 유·무선 인터넷 보안