Journal of Information Processing Vol.23 No.5 655–663 (Sep. 2015)

[DOI: 10.2197/ipsjjip.23.655]

Regular Paper

HTTP-GET Flood Prevention Method by Dynamically Controlling Multiple Types of Virtual Machine Resources

Mizuki Watanabe[1,a)] Ryotaro Kobayashi[1] Masahiko Kato[2]

Received: November 28, 2014, Accepted: June 5, 2015

Abstract: Currently, Web services are widely utilized to disclose company information, and offer online services and e-commerce. As these services have become an essential part of our everyday lives, the public is greatly inconvenienced when they are disrupted. Denial of service (DoS) attacks exert adverse influences on Web services. We focus on HTTP-GET Flood attacks, which are manually operable DoS attacks. It is possible to simply block manually operable DoS attacks such as F5 attacks on the server side; however, such measures could be noticed by the attackers. Therefore, to prevent the attacker changing their method of attack, it is possible to overcome the attack by redirecting the attack to another system, for which a previous study has proposed a feasible technique located in the service provider. The previous study assumes a correlation between the CPU resource and the request error rate. However, the Web Server actually has multiple resources. Therefore, it is important to be able to control the server resources rather than the CPU and the memory. The operational implementation of the proposed method and the evaluation experiments confirm the effectiveness of the proposed method.

Keywords: network, DoS, virtual machine, HTTP-GET flood

1. Introduction

Currently, web services are widely utilized to disclose company information and offer online services and e-commerce. As these services have become an essential part of our everyday lives, the public is affected when they are disrupted. Denial of service (DoS) attacks exert significant adverse effects on Web services. If a DoS attack floods a targeted system with requests, the service of the system becomes unavailable to its intended users.

DoS attacks can be classified into one of the two following categories:

(1) Human controlled DoS attacks
(2) Bot-based DoS attacks

In a human controlled DoS attack, a human controls the attack by entering keyboard commands or executing an attack program on the attacker's computer, while monitoring responses from the targeted server. By contrast, in a bot-based DoS attack, a client computer is infected with a virus that forcibly overflows the network or the targeted server unbeknown to the owner of the infected computer. In our study, we propose a prevention system for the former.

According to another classification standard, DoS attacks can be divided into some classes including the followings.

- Attacks that utilize system vulnerabilities
- Attacks that waste network resources
- Attacks that waste server resources

Attacks utilizing system vulnerabilities attempt to carry out security attacks by exploiting existing system vulnerabilities resulting from design errors or ambiguities. These types of attacks can be alleviated by building new software that incorporates knowledge about security, and verifies and corrects all vulnerabilities in the systems. Attacks that waste network resources attempt to render a running service unavailable to its users by sending the network a large number of large-sized packets. In this case, if the system cannot supply sufficient network resources to process the vain packets, the network bandwidth becomes inundated, indicating a successful attack. These types of attacks are difficult to address on the server side. Thus, they should be prevented on the network side. Although it is important to protect network resources, this paper focuses on the attacks which are prevented on the server side. Attacks wasting server resources attempt to prevent the server from responding to its users by sending a mass of extraordinary packets that saturate server resources such as memory, CPU time, and disk space. These can often be prevented on the server-side.

Examples of attacks that waste server resources include SYN Flood attacks, Connection Flood attacks, and HTTP-GET Flood attacks. A SYN Flood attack repeatedly starts a TCP three-way handshake but does not complete it by responding to the server with an ACK code, resulting in increasingly large numbers of half-open connections that eventually prevent the server from establishing new connections with legitimate traffic. A well-known countermeasure for such an attack is to either shorten the regulation time for a time-out decision while waiting for an ACK or use defense functions (e.g., SYN cookies). A Connection Flood attack occupies sockets of the server by establishing a large amount of connections, which remain open for a long time. Some well-

[1] Faculty of Engineering, Toyohashi University of Technology, Toyohashi, Aichi 441–8580, Japan
[2] Internet Initiative Japan Inc., Chiyoda, Tokyo 101–0051, Japan
[a)] watanabe.mizuki@ppl.cs.tut.ac.jp

known countermeasures have been proposed, one of which is to increase the socket open-state of the server or queue assignment for the TCP, while another is to shorten the regulation time for a time-out decision of a connection. An HTTP-GET Flood attack sends a large amount of GET requests to the server, forcing it to process them, and saturating the server to prevent the service. A number of well-known countermeasures include enabling KeepAlive, decreasing the capacity of the Web page, and introducing load balancing.

In this paper, we focus on F5-Attacks, which are manually operable types of HTTP-GET Flood attacks. The F5 key is assigned to the function of updating the Web page (resending a GET request) in a variety of browsers. An attacker simply continues to push the F5 key to send a large number of GET requests. The attacker can only confirm whether the attack is performed successfully by browsing the affected web pages.

Several prevention methods are available to prevent DoS attacks. One prevention method is to raise server performance; however, this method requires surplus server resources even when there is no attack, degrading the cost performance of the system. Moreover, if the attackers judge the amount of attacks as insufficient, they can merely increase the severity of attacks; thus, the administrator needs to reinforce server performance endlessly to manage any attacks. Another prevention method is to filter and/or restrict the bandwidth in the firewall or the router [2]. However, if these methods are applied to the victim server, the attackers may notice that their attacks are being prevented and subsequently change their method of attack [3].

To overcome this problem, Takahashi et al. [1] proposed a prevention method (referred to in this paper as the conventional method) that allows the server to continue providing a service to its users in the presence of an F5-attack. This method prepares two types of virtual machines: one for its users and the other for the attackers. The F5-attack delays the response time to GET requests on the attacker virtual machine. If the response time to a GET request is longer than the timeout time, the request becomes an error. A moderate rate of request errors on the attacker virtual machine makes it difficult for attackers to notice that their attacks are being prevented. In order to stably control the rate of request errors, despite the varying amount of F5-attacks, the CPU resource of the server is dynamically controlled according to the monitored server response. However, this study focuses only on the CPU resources for stable control of the request error rate. Therefore, if there is a low, or no, correlation between the CPU resources and the request errors [4], [5], [6], the request error rate is difficult to control.

In order to solve this problem, by focusing on both the CPU and memory resources, we propose a method that dynamically selects one of the two resources closely correlated with the request errors caused by the F5-attacks and controls the selected resource to control the request errors. An unfavorable request to the server side is regarded as an attack, and if the request frequency from a client's IP address is over a certain threshold, the client is judged as an attacker. In this paper, we implement our proposed method and evaluate the effectiveness of the system.

The remainder of this paper is organized as follows. In Section 2, we discuss the conventional method proposed by Takahashi et al. [1]. In Section 3, we describe related research, and in Section 4, we state our ideas and the proposed method. Section 5 illustrates the implementation of our method, and an evaluation of our method is presented in Section 6. Our discussion of the effectiveness of our method is given in Section 7, and Section 8 concludes our paper.

2. Conventional Method

2.1 Attacker Assumed in the Conventional Method

The conventional method assumes that the attackers perform the F5-Attack to send GET requests to the Web Server and monitor the responses. If the request error rate is very low, the attackers judge that their attacks are insufficient and increase the amount of attacks. If the request error rate is very high, they judge that their attacks are simply being prevented by some kind of countermeasure and change their method of attack. If the request error rate is moderate, they judge their attacks as successful and continue to sustain their attacks.

2.2 Idea of the Conventional Method

The conventional method takes measures considering that the prevention has a minimal effect on the normal users and is hardly noticeable by the attackers.

In the conventional method, the Web Server is separated into a **Decoy Machine** and a **Normal Machine**: the Decoy Machine is a Web Server for the attackers, while the Normal Machine is a Web Server for the intended users. As mentioned in Introduction, virtual machines operate as the Normal and Decoy Machines. Attacker's requests are forwarded to the Decoy Machine, and normal user's requests are forwarded to the Normal Machine by the administrative domain. In this way, the separate resources used in the conventional method protect the normal users from attacks.

The Decoy Machine offers the same web service as the Normal Machine so that the attackers are unaware the computer resources are being separated for attack prevention. However, a Decoy Machine alone is insufficient. In order to create the situation in which the attacker believes the attacks are successful, it is necessary to stably control the request error rate. Therefore, the conventional method controls the CPU resource of the Decoy Machine while the Decoy Machine is running so that the request error rate reaches a target value set by the administrator.

2.3 Problem of the Conventional Method

The conventional method assumes that a correlation exists between the CPU resource and the request error rate. In this case, if the CPU resource increases or decreases, the request error rate decreases or increases, respectively. The conventional method focuses on the above correlation to control the CPU resource and realize the stable control of the request error rate.

However, the Web Server actually has multiple resources; thus, determining which resource has a practical correlation with the request error rate differs from one case to another. Therefore, if there is a low, or no, correlation between the CPU resource and the request errors [4], [5], [6], it is difficult for the conventional method to stably control the request error rate. Similarly, if the

request error rate is too high or too low, it cannot be controlled by the CPU resource control.

In this paper, we assume the Web Server that requires either the CPU resource control or the memory resource control for the stable control of request error rate. Which resource has the practical correlation depends on the service supplied by the Web Server.

In the Web Server that requires the memory resource control, the attacks make the Web Server generate many processes, each of which requires the large memory space for dynamic contents on the web page [7], [8]. As a result, the memory resource is saturated. In this case, the conventional method does not work well.

There are many types of server resources, major ones of which are as follows: CPU, memory, disk I/O, and network I/O. In this paper, we focus on the CPU and the memory. Although it is also important to be able to control other server resources, we leave it to future work.

3. Related Work

Xie et al. [9] proposed a technique for DDoS attack prevention, but this is outside the scope of this paper. Their method identified whether a "flashcrowd," a situation in which a large volume of users accesses a popular website on the internet, is generated by a DDoS attack. DDoS attacks are detected through a learning model based on principal component analysis (PCA), independent component analysis (ICA), and a hidden semi-Markov model (HsMM). If this method is introduced, it may be possible to prevent bot attacks from hindering our proposed method to deal with F5-Attacks.

Khor et al. [10] also proposed a DDoS prevention method that constructed a DDoS Mitigation-as-a-Service (DaaS), which supplies a service to protect servers against DDoS attacks. DaaS drops bot traffic, using its own resources only for legitimate clients. DaaS has inbuilt defense functions for three types of DDoS: network-layer DDoS, application-layer DDoS, and economic DDoS (eDDoS). This defense function is realized using the following operations: (1) The DaaS framework facilitates utilization of idle internet resources from existing or future systems/services without modification, and (2) a function termed, self-verifying Proof-of-Work (sPoW), defends against eDDoS attacks by performing increases in resource requirements for DDoS attacks and decreases in resource consumption required for effective defense. PoW is an economic method used to deter DoS attacks and other service abuses such as spam on a network by requiring additional work from service requesters. In addition, sPoW has an internal function to modify its own operation. If the server is prepared to protect against DDoS independently, our method can exclude bot attacks from objects controlled under this network.

Wang et al. [11] proposed making schedulers on both the virtual machine host and the virtual machine guest which operate cooperatively to improve resource management and application performance. In fuzzy-modeling based resource management, workload characteristics are extracted and the virtual machine modeling is improved based on the guest-layer application knowledge. The scheduler on the virtual machine host feeds back a resource allocation decision and takes account of the application configuration on the virtual machine guest. This method is applied to virtualized databases with dynamic and complex resource usage behaviors. In this application, this method characterizes query workloads and efficiently allocates resources to the database virtual machines to improve application performance.

Das et al. [12] proposed an effective method that detects HTTP-GET Flood attacks. They assumed three different situations in which HTTP-GET Flood attacks occur. As the patterns of attacks are hardly identifiable (e.g., the slashdot effect) this method prepares the right request patterns in advance and calculates nonconformity between the right request patterns and the current request patterns that are dynamically captured. If the nonconformity, which is calculated based on the arrival rate of the GET requests, exceeds a predetermined threshold configured by a user, the GET requests are judged as attacks. The evaluation results showed that their proposed method can detect attacks with an accuracy rate of 99%.

4. Proposal

4.1 Main Idea

Figure 1 shows the example of the construction of the proposed method. In this paper, we assume the attackers described in Section 2.1 and that the Web Server requires either the CPU resource control or the memory resource control to stably control the request error rate. However, the Web Server actually has multiple resources; thus determining which resource has the practical correlation with the request error rate depends on the service supplied by the Web Server. This paper focuses on two server resources: CPU and memory. It is also important to be able to control other server resources, and this will comprise future work.

Based on this situation, the conventional method does not work well. For example, when the Web Server requires the memory resource control, the request error rate changes according to the amount of attacks, even if the conventional method controls the CPU resource for the target error rate. This cannot control the stable request error rate and fails to convince attackers that their attacks are successful.

To address the problem, we propose a method that stably controls the request error rate for varying amounts of F5-attacks, regardless of which resource has a correlation with the request error rate. This method selects whether to control the CPU or memory resources in advance and controls the selected resource to stably control the request error rate to make it close to the target value

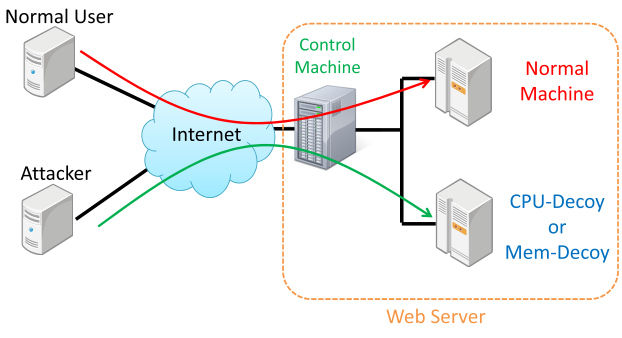


Fig. 1 Main Idea.

which is set by the administrator even when the amount of F5-attack varies. This stable request error control falsely shows the attackers that their attacks are going to succeed in saturating the server resources.

If the change of server resource affects the response time the most, the server resource contributes to the response time [13]. Hence, in order to select the resource to be controlled, our proposed method changes the CPU and memory resources individually to confirm the change in the request error rate, and selects the resource that contributes more to the request error rate control.

4.2 Criteria of the Attack and the System Load

DoS attacks are considered requests with the intention of disrupting Web services. A request without such an intention cannot be called a DoS attack, even if it puts enough load on the Web Server to affect the service. However, it is difficult for the service providers to detect whether a request is made with the intention of disrupting the service. Therefore, in this paper, an unfavorable request to the server side is regarded as an attack, and if the request frequency from a client's IP address is over a certain threshold, the client is judged as an attacker. According to the above definition, the requests are separated into normal and attack requests. The normal/attack request separation is assumed to be perfectly performed.

Similar to the conventional method, this study uses the request error rate as the evaluation criteria for the system load of the server. We assume that an HTTP communication succeeds if a client sends a GET request to a server and receives a response, even if the response is negative, for example, the request page is not found on the server. The response time is the time between the sending of the GET request and the receipt of the response. F5-attacks put a load on the server, which slow the response time. Therefore, if the response time is longer than the timeout time, the request is processed as an error.

4.3 Components

4.3.1 Separation of Computer Resources

In the proposed method, we separate the computer resources into three different types of machines: the Decoy Machine, the Normal Machine, and the **Control Machine**, in order to keep the attackers from having an impact on normal users and to provide a control mechanism for the proposed method. In the conventional method, the term "Decoy" is used as the general term. In the proposed method, we define three types of Decoy Machines in Section 4.3.2: Init-Decoy, CPU-Decoy, and Mem-Decoy.

Since it is not realistic to prepare multiple, costly physical machines for momentary attacks, and it is difficult to control the resources of a physical machine operating as a Decoy, we use virtualization software to implement multiple units of virtual machines on a physical machine [14], [15].

When the clients communicate with the virtual machines, all the packets go through the host OS, which forwards the requests from the clients to the virtual machines depending on the types of clients and machines. Attackers' requests are forwarded to the Decoy and normal user requests are forwarded to the Normal Machine. This forwarding prevents normal users from being affected by any attacks because the resources for the attackers and those for normal users are kept separate.

4.3.2 Operations of Control Machine and Web Servers

The Control Machine performs the selection of the resource to be controlled, the judgment of the attackers, the forwarding of the requests, and the control of the selected resource. The details of them are described in the next section.

Normal Machine and Decoy operate as Web Servers. If there is no attack, they offer the service for normal users. If F5-attacks launch, Normal Machine and Decoy offer the same service for normal users and attackers, respectively. Before the resource selection, any resource is not controlled. After the resource selection, either the CPU resource or the memory resource of the Decoy is controlled to stably control the error rate of the GET requests. A Decoy is called an Init-Decoy if any resource is not controlled. A Decoy is called a CPU-Decoy if the CPU resource is controlled and a Mem-Decoy otherwise.

4.4 Comparison with Conventional Method

In the conventional method, the Web Server has Normal Machine and Decoy which has only CPU resource control. In order to control both CPU and memory resources, we extend the Decoy in the conventional method to propose the new method. In the proposed method, unlike the conventional method, the term "Decoy" is the general term and we introduce three types of Decoys: Init-Decoy, CPU-Decoy, and Mem-Decoy. When the Web Server starts to run, a Decoy starts to run as an Init-Decoy. An Init-Decoy does not have any resource control. Based on the resource selection result, an Init-Decoy is changed to a CPU-Decoy or a Mem-Decoy. A CPU-Decoy in the proposed method corresponds to a Decoy in the conventional method. A Mem-Decoy is introduced for memory resource control which can't be performed in the conventional method.

5. Implementation

5.1 Web Server

Figure 2 shows an example of the Web Server using the proposed method. The Web Server consists of the control machine and two Web Sites. These sites are implemented by using virtual hosting, which allows a single server to host multiple Web Sites. Each website consists of a Normal Machine and a Decoy. The Decoys shown in Fig. 2 have been already judged to be either a CPU-Decoy or a Mem-Decoy by the Control Machine. On the client side, every user will appear to transmit GET requests to Web Site A or B, while on the server side, the Normal Machines and the Decoys supply the actual services.

We use Xen as virtualization software to prepare the privileged virtual machine called Dom0, which runs as the Control Machine, and the unprivileged virtual machines, called DomU, which run as the Normal Machines and the Decoys. The Control Machine can capture and forward the request packets from the clients to the virtual machines and control the server resources of the virtual machines contained in the Web Sites. We write dedicated software, called Controller, in a Python language and a script language to implement the operations of the Control Machine. Controller runs on the Control Machine.

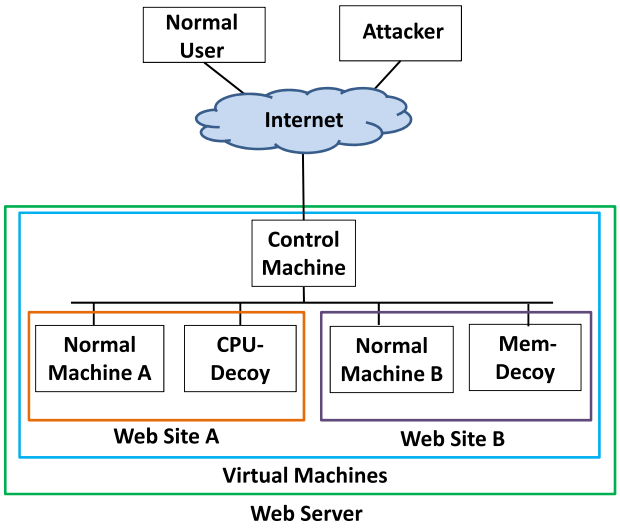


Fig. 2 Example of system configuration.

5.2 Resource Control Mechanism

The following parameters are related to the CPU and memory resources of DomU:

- the number of virtual CPU
- mapping from physical CPU to virtual CPU
- threshold of CPU utilization
- CPU scheduling priority
- memory size
- maximum memory size

In this paper, only the threshold of CPU utilization, called a cap, and the memory size are used.

In order to control the CPU resource, we use an xm command and a sched-credit option to set the cap of a DomU to allocate CPU time to the DomU. The cap is the maximum amount of CPUs that can be consumed by a DomU. The cap can be set from 0 to 100 per one physical CPU. The cap values between 1 and 100 indicate the maximum CPU utilization rate of a DomU. On the other hand, cap value 0 has a different meaning and indicates that there is no constraint on CPU utilization. For example, if the cap is 50 and one physical CPU is allocated, the maximum CPU utilization rate is 50%. In addition to the cap, a CPU scheduling priority can affect the CPU time; however, it does not affect it in our implementation because the CPU scheduling priorities of all the DomUs are the same.

In order to control the memory resource, we use an xm command and a mem-set option to set the memory size of a DomU.

5.3 Request Forwarding Mechanism

We use a NAT function of iptables on the Control Machine to forward the GET requests from the clients to the Normal Machines and the Decoys. The controller executes an iptables command to create a NAT mapping between a client and a DomU, where the DomU is either a Normal Machine or a Decoy. The selected DomU changes from case to case. Based on the database containing the NAT mappings, the Control Machine forwards the requests from the clients to the Normal Machines and Decoy Machines.

When there are no attacks, mappings are created between normal users and Normal Machines and between normal users and Decoys Machines, depending on the hash value of the client's IP address.

When attacks are being made, mappings are created between normal users and Normal Machines, while separate mappings are created between attackers and Decoy Machines.

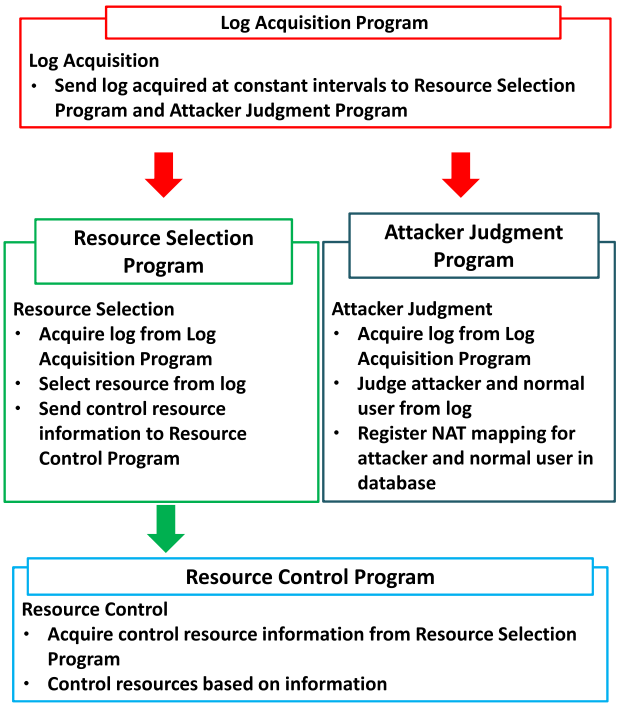


Fig. 3 Structure of controller.

5.4 Controller

Controller is software that runs on the Control Machine and consists of a log acquisition program, an attacker judgment program, a resource selection program, and a resource control program (**Fig. 3**).

5.4.1 Log Acquisition Program

The log acquisition program repeats a series of operations at regular intervals to acquire the access log from output of tcpdump and send it to the attacker judgment program and the resource selection program to clear log. Tcpdump is a software tool that can capture the contents of all or certain packets for network traffic analysis.

5.4.2 Attacker Judgment Program

According to the log sent from the log acquisition program, the attacker judgment program calculates the number of requests each client makes at one time. If the number of requests exceeds a predetermined threshold, the client is judged an attacker; otherwise, the client is judged a normal user. As a result, all the clients are classified as either normal users or attackers. Based on the classification, this program registers the NAT mappings for each client in the database.

5.4.3 Resource Selection Program

The resource selection program sends GET requests to the Init-Decoys for the resource selection and performs the following operations. First, according to the log sent from the log acquisition program, the request error rate of each Init-Decoy in the case of

no change in resources is calculated. Second, the CPU resource and the memory resource are individually changed and the request error rates of each Init-Decoy after changes in the CPU resource and the memory resource are calculated, respectively. Lastly, if the change in the request error rate due to the CPU resource is larger than the change due to the memory resource, the CPU resource is selected for the resource control. Otherwise, the memory resource is selected.

5.4.4 Resource Control Program

The resource control program performs the following functions for each Decoy:

- calculation of the request error rate
- modification of the selected resource

In this program, each Decoy is either a CPU-Decoy or a Mem-Decoy. The first function sends a GET request per second to the Decoy and records the time when a GET request becomes an error. Based on the record, the request error rate is calculated per second by dividing the number of request errors by the number of GET requests in the last 60 seconds. In the last 60 seconds, the number of request errors varies depending on the operation of the Web Server but the number of GET requests is always 60 because the first function sends a GET request per second. The second function compares the last request error rate with the target request error rate and modifies the selected resource of the Decoy to make the next request error rate closer to the target request error rate.

5.5 System Operation

In this subsection, we explain the system operation, which is divided into two modes: normal mode and prevention mode. If there is no attack, the system selects normal mode; otherwise, it selects prevention mode. The presence or absence of attacks is checked at regular intervals by the Control Machine.

5.5.1 Normal Mode Operation

The system operation of normal mode is shown in **Fig. 4**. In this figure, it is assumed that the construction of the Web Server is the same as that shown in Fig. 2; there are no attackers, normal users A and B send requests to Web Site A, and normal users C and D send requests to Web Site B.

The Control Machine selects the CPU and memory resources to be controlled for Web Sites A and B, respectively. Based on the hash value of each normal user's IP address, the NAT mappings related to the normal users are registered in the database. According to the database, the Control Machine forwards the requests from the normal users to the Normal Machines and the Decoys, as shown in Fig. 4.

5.5.2 Prevention Mode Operation

Figure 5 shows the system operation of the prevention mode. The Web Server in Fig. 5 is the same as that shown in Fig. 4 with the exception of the operation mode. In Fig. 5, it is assumed that attacker A and normal user A send requests to Web Site A, and attacker B and normal user B send requests to Web Site B.

The Control Machine judges that attacker A and attacker B are attackers based on the NAT mappings relating to the normal users and the attackers, which are registered in the database. According to the database, the Control Machine forwards the requests from the normal users to the Normal Machines and those from the attackers to the Decoys, as shown in Fig. 5. The Control Machine controls the CPU resource of the CPU-Decoy and the memory resource of the Mem-Decoy to make the request error rates of the CPU-Decoy and the Mem-Decoy close to the target value.

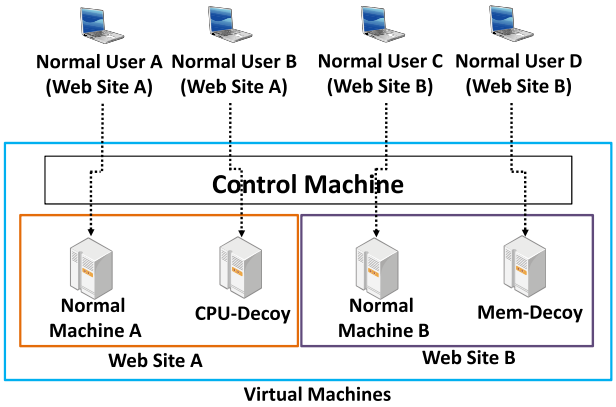


Fig. 4 Normal mode operation.

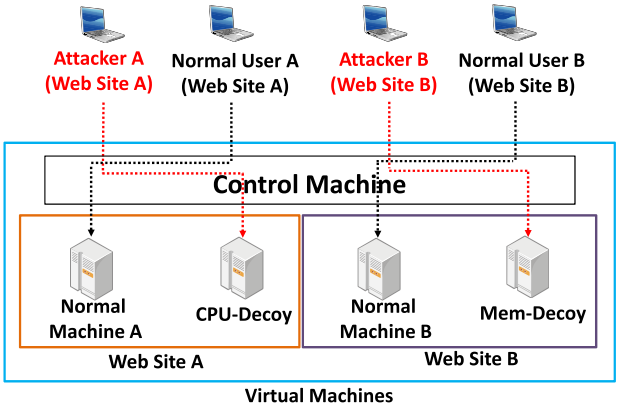


Fig. 5 Prevention mode operation.

6. Evaluation Method

In this section, we describe the evaluation environment and method. **Figure 6** shows the evaluation environment, which includes multiple machines: Attack server 1, Attack Server 2, Web Server, and Measurement Server. Web Server is included the Control Machine described in Section 4.3.2. The servers are the host machines, which run Xen to implement multiple VMs, denoted as VM_n, where n is the number of VMs. **Table 1** shows the specifications of the Web Servers and VMs.

To operate four Web Servers on a single machine, we prepare four VMs: VM1-4. We evaluate two prevention methods: the conventional method and our proposed method. In the evaluation of the conventional method, VM1-4 are operated as Normal Machines and Decoys. It should be noted that only the CPU resource of each Decoy is controlled in the conventional method. On the other hand, in the evaluation of the proposed method, VM1-4 are operated as Normal Machines, a CPU-Decoy, and a Mem-Decoy. Attack Server 1 and Attack Server 2 are each equipped with 10 attacks, VM1-10, which launch HTTP-GET Flood Attacks against the Web Servers, making 20 attackers.

An attacker launches HTTP-GET Flood Attacks using a Java application called JMeter, which is a commonly used a load test-

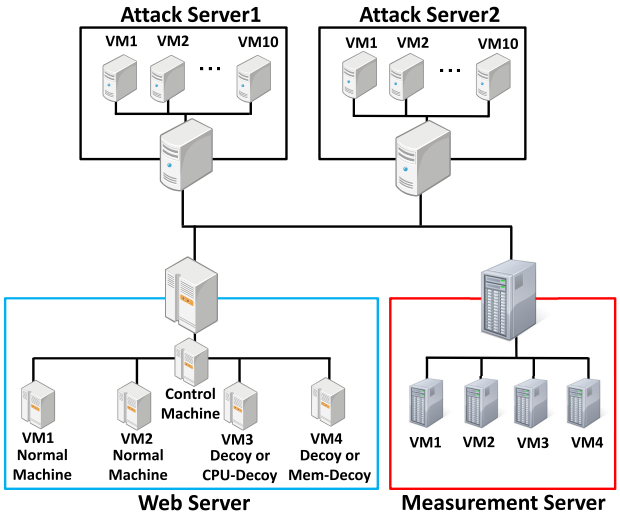


Fig. 6 Evaluation environment.

Table 1 Specification of servers and virtual machines.

(a) Web server specification.

OS	openSUSE 11.3 64 bit (2.6.34.7-0.5-xen)
CPU	Intel Xeon L3426 (1.87 GHz)
Memory	8 GByte
VM	Xen 4.0.0 21091 06 0.1

(b) Attack server specification.

OS	openSUSE 11.3 64 bit (2.6.34.12-xen)
CPU	AMD Phenom II X6 1055T (2.8 GHz)
Memory	8 GByte
VM	Xen 4.0.0 21091 06 0.1

(c) Virtual machine specification on Web server.

OS	openSUSE 11.3 64 bit (2.6.34.7-0.3)
CPU	1 Unit
Memory	512 MByte or 2 GByte

(d) Virtual machine specification on attack server.

OS	openSUSE 11.3 64 bit (2.6.34.7-0.3)
CPU	1 Unit
Memory	512 MB

ing tool. We use the JMeter parameters, thread delay interval and the number of issuable threads, to control the amount of attacker requests per second, which are set to 180, 200, 220, 240, and 260 [unit/sec]. These values are determined as shown below. The amount of attacker requests per second is calculated by the product of the number of attackers and the number of attacker requests per attacker per second. As described in the 3rd paragraph in Section 6, 20 attackers are prepared. The numbers of attacker requests per attacker per second are set to 9, 10, 11, 12, and 13 [unit/sec], which are determined based on the observation result which is acquired by actually pushing F5 key on the keyboard by a finger on several occasions.

The Measurement Server enables JMeter to measure the request error rate on VM1-4 without being affected by the attackers. The mapping information used for the request is registered in the NAT table and referred to by iptables on the Control Machine. The request error rate is measured once per second for calculation. The duration of each evaluation is 30 minutes.

The initial value of the cap and memory of each VM is set to 0 and 512 MB to allow resources to be equally distributed among

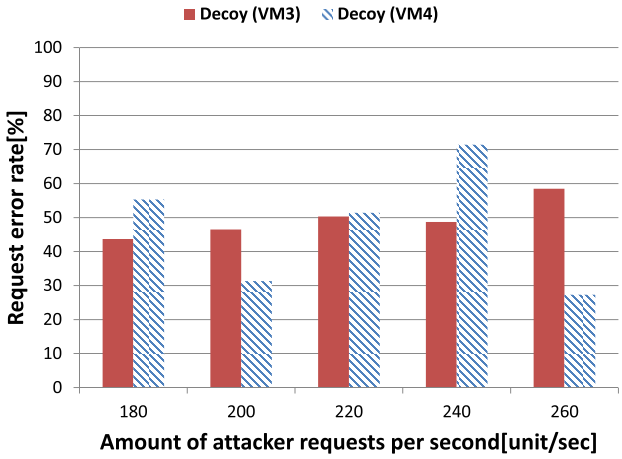


Fig. 7 The request error rate of the conventional method.

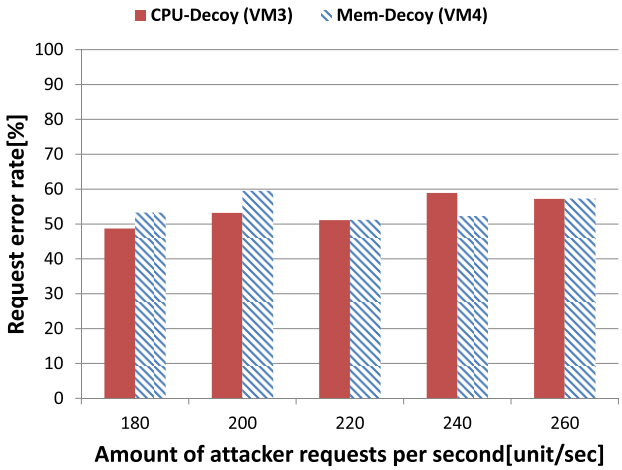


Fig. 8 The request error rate of the proposed method.

the four VMs on the Web Server. The cap or the memory of each VM is adjusted by the resource control program in order to make the request error rate approach the target value of 50% that is used in Ref. [1].

7. Result

We evaluate the request error rate with varying amounts of attacker requests per second. **Figure 7** shows the results of the conventional method and **Fig. 8** shows the results of the proposed method. In the figures, "Decoy (VM3)," "Decoy (VM4)," "CPU-Decoy (VM3)," and "MEM-Decoy (VM4)" are used as graph legends to specify the correspondences of a Decoy to a VM.

Since the request error rate of the Normal Machines is 0%, the request error rates in each graph are omitted. Figure 7 shows the request error rates in the conventional method, described in Section 2, with varying amounts of requests per second. Figure 7 shows that each group of two bars from left to right indicate the request error rates of Decoy (VM3) and Decoy (VM4), respectively. Only the CPU resource of each Decoy is controlled. For Decoy (VM3), we can make the error rate close to the target value of 50% regardless of the amount of attacker requests per second, while for Decoy (VM4), when the amount of attacker requests per second is 240, the request error rate is higher than the target value of 50%. Conversely, when the amount of attacker requests per second is 260, the request error rate is lower than the tar-

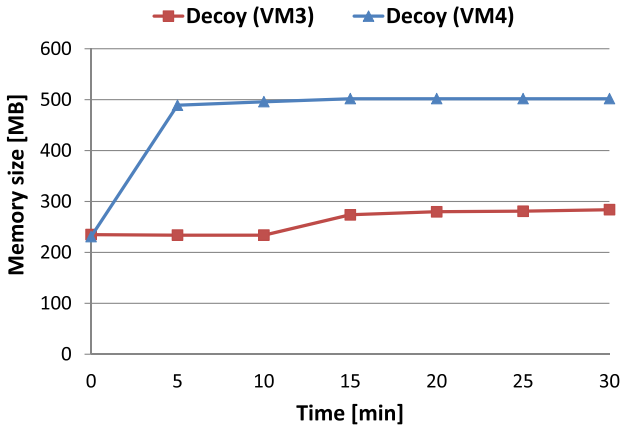


Fig. 9 Memory size of the conventional method.

get value, meaning that the conventional method is unsuitable for Decoy (VM4). In addition, if the Decoy (VM4) resource is controlled in the conventional method, the request error rate is up to 22.7% far from the target value. One of the reasons for this is the memory resource usage of Decoy (VM4). **Figure 9** shows the variation of memory size of the Web Servers in the above measurement. The vertical axis shows the memory size, and the horizontal axis shows the elapsed time. The memory size of Decoy (VM3) is relatively stable, while that of Decoy (VM4) is significantly increased between the start and the end of the measurement. Therefore, if the request error rate control focuses solely on CPU resources, it cannot adapt to variations of memory size and fails to control the request error rate stably on Decoy (VM4); thus affecting the result.

Figure 8 shows the request error rate when the amount of requests per second varies in the proposed method described in Section 4. Figure 8 shows that each group of two bars from left to right indicate the request error rates of CPU-Decoy (VM3) and Mem-Decoy (VM4), respectively. As for both CPU-Decoy (VM3) and Mem-Decoy (VM4), we can make the request error rate close to the target value of 50% regardless of the amount of attacker requests per second. Since we don't improve the conventional method in terms of adjusting the CPU resource, there is no functional difference between "Decoy (VM3)" in the conventional method and "CPU-Decoy (VM3)" in the proposed method. However, in the evaluation results, they are close but not the same in the request error rate even if the amount of attacker requests per second is fixed. The reason for this is as follows. The request error rate depends on the operation of the Web Server. However, the operation of the Web Server is not stable when the request errors occur. Therefore, the request error rate varies depending on each time of evaluation. In the proposed method, we control memory resources of Mem-Decoy (VM4), the memory usage of which varies widely with the resource selection program. In addition, if Mem-Decoy (VM4) resources are controlled in the proposed method, the request error rate is at most 9.5% far from the target value. Therefore, the request error rate is improved by 13.2% in the proposed method.

The above results show that memory resource control is needed to control the request error rate stably on Mem-Decoy (VM4) and that the mechanism of the proposed method enables more stable control by changing control resources depending on the Web Servers.

The proposed method extends the conventional method to increase the kinds of Decoys and add the functionality to Controller. In order to evaluate the influence of the extension on the system load, we run a top command for 30 minutes to measure the average CPU time of Controller on the Control Machine when the amount of attacker requests per second is 180 [unit/sec]. The results show that the average percentages of the CPU time in the conventional method and the proposed method are 8.2% and 7.1%, respectively. However, the CPU time changes depending on each time of evaluation. Considering the change in the CPU time, the results confirm that there is no significant difference in the CPU time between the conventional method and the proposed method. Unlike the conventional method, the proposed method needs to select the resource to be controlled in advance. We use a top command to measure the average CPU time of Controller during the resource selection, the time needed for which is 2 minutes. The evaluation result shows that the average CPU time of the resource selection is only 1.0%.

The system implemented in the proposed method deploys multiple Web Servers of which one server is designated as a Decoy Machine. The experiment shows that the dynamic resource selection between the CPU and memory resources makes the resource control more effective. We focus on two types of resources; however, other resources also affect the performance of Web Servers, which should be considered in future work.

8. Conclusion

In this paper, we focused on HTTP-GET Flood attacks, which are a type of DoS attack. Our previous work proposed a method to make Attackers believe their attacks are successful. The conventional method assumes there is a correlation between the CPU resource and the request error rate. However, the Web Server actually has multiple resources. The resource which has a practical correlation with the request error rate differs from case to case. As a result, even if the request error rate is too high or too low, it cannot be controlled by the CPU resource control. To solve this problem, we proposed a method that stably controls the request error rate for varying amount of F5-attacks, regardless of which resource has a correlation with the request error rate. Furthermore, we have evaluated our method and shown its effectiveness. In this study, we focus solely on memory and CPU resources; nevertheless, as many resources affect the performance of the Web Server, the subject of future work will be to study the effect of the other resources.

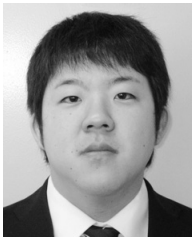
Acknowledgments The authors wish to thank Tsuyoshi Mikami and Shoma Yoshida for technical assistance and helpful discussions that they provided when they were master students at Toyohashi University of Technology.

References

[1] Takahashi, T., Taguchi, G., Kobayashi, R. and Katoh, M.: HTTP-GET Flood Provision by Dynamic Resource Control to Virtual Machine, *IEICE Trans. IaS.*, Vol.J94-D, No.12, pp.2058–2068 (2011) (in Japanese).

[2] @police: The results of the internet observation from July to Septem-

ber in 2012, available from ⟨http://www.npa.go.jp/cyberpolice/detect/pdf/20121114.pdf⟩.
[3] IPA: The report of investigation into the DoS Attack Prevention, available from ⟨http://www.ipa.go.jp/security/fy22/reports/isec-dos/index.html⟩.
[4] Chandra, A., Gong, W. and Shenoy, P.: Dynamic Resource Allocation for Shared Data Centers Using Online Measurements, *Proc. 11th International Conference on Quality of Service*, pp.381–398 (2003).
[5] Ramamurthy, P., Sekar, V., Akella, A., Ander, B.K. and Shaikh, A.: Using Mini-flash Crowds to Infer Resource Constraints in Remote Web Servers, *Proc. 2007 SIGCOMM Workshop on Internet Network Management*, pp.250–255 (2007).
[6] Srivatsa, M., Iyengar, A., Yin, J. and Liu, L.: A Middleware System for Protecting Against Application Level Denial of Service Attacks, *Proc. ACM/IFIP/USENIX 2006 International Conference on Middleware*, pp.260–280 (2006).
[7] Lee, S., Jung, C. and Pande, S.: Detecting Memory Leaks Through Introspective Dynamic Behavior Modelling Using Machine Learning, *Proc. 36th International Conference on Software Engineering*, pp.814–824 (2014).
[8] Ben-Yehuda, O.A., Posener, E., Ben-Yehuda, M., Schuster, A. and Mu'alem, A.: Ginseng: Market-driven Memory Allocation, *Proc. 10th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pp.41–52 (2014).
[9] Xie, Y. and Yu, S.: Monitoring the Application-layer DDoS Attacks for Popular Websites, *IEEE/ACM Trans. Netw.*, Vol.17, No.1, pp.15–25 (2009).
[10] Khor, S.H. and Nakao, A.: DaaS: DDoS Mitigation-as-a-Service, *Proc. IEEE/IPSJ International Symposium on Applications and the Internet*, pp.160–171 (2011).
[11] Wang, L., Xu, J. and Zhao, M.: Application-aware Cross-layer Virtual Machine Resource Management, *Proc. 9th International Conference on Autonomic Computing*, pp.13–22 (2012).
[12] Das, D., Sharma, U. and Bhattacharyya, D.K.: Detection of HTTP Flooding Attacks in Multiple Scenarios, *Proc. 2011 International Conference on Communication Computing Security*, pp.517–522 (2011).
[13] Liu, X., Sha, L., Diao, Y., Froehlich, S., Hellerstein, J.L. and Parekh, S.: Online Response Time Optimization of Apache Web Server, *Proc. 11th International Conference on Quality of Service*, pp.461–478 (2003).
[14] Rao, J., Wei, Y., Gong, J. and Xu, C.: DynaQoS: Model-free Self-tuning Fuzzy Control of Virtualized Resources for QoS Provisioning, *Proc. 19th International Workshop on Quality of Service*, pp.31:1–31:9 (2011).
[15] Perez-Botero, D., Szefer, J. and Lee, R.B.: Characterizing Hypervisor Vulnerabilities in Cloud Computing Servers, *Proc. 2013 International Workshop on Security in Cloud Computing*, pp.3–10 (2013).



Mizuki Watanabe received his B.E. degree in Information from Toyohashi University of Technology in 2011. He is now a master student at Toyohashi University of Technology. His research interests include network security.

Ryotaro Kobayashi received his B.E., M.E., and D.E. degrees from Nagoya University in 1995, 1997, and 2001, respectively. He had been a research assistant in Nagoya University from 2000 to 2008. He is currently a lecturer at Toyohashi University. His research interests include computer architecture, parallel processing, and network security.

Masahiko Kato recieved his B.Sc. and M.Sc. degrees in engineering from Toyohashi University of Technology in 1993 and 1995 respectively. He is now working for Internet Initiative Japan Inc. since 1998. He is currently interested in network security and cloud computing security.