Regular Paper

Defense Method of HTTP GET Flood Attack by Adaptively Controlling Server Resources Depending on Different Attack Intensity

Ryotaro Kobayashi[1,a) Genki Otani[1] Takuro Yoshida[1] Masahiko Kato[2]

Received: December 3, 2015, Accepted: June 2, 2016

Abstract: The Internet currently provides a multitude of services, which have become essential for everyday life such as disclosure of company information, online services, and e-commerce. Therefore, interruptions to these services greatly inconvenience the public. A denial of service (DoS) attack affects regular users' access to a network resource. DoS tools usually include a function for monitoring the status of the targeted server that allows the attacker to confirm the effectiveness of the current attack and the defense activities of the server, and thus plan further attacks. By observing the effectiveness of the current attack, the attacker can adjust the attack intensity to match the server's status. Depending on the defense response, the perpetrator can judge whether their attack is being mitigated using certain techniques. If the attacker observes a defensive response to the attack, the attacker can respond by changing the attack method, abandoning the attack, or targeting a more vulnerable server. We propose a method that allows the server to maintain its service to users relatively unaffected by the attacks, responds optimally to each attacker, and impedes the attacker's ability to detect defensive responses. In this paper, we implement our proposed method and evaluate the effectiveness of the system.

Keywords: HTTP GET Flood, DoS, Adaptive Control, virtual machine

1. Introduction

The Internet currently provides a multitude of services that have become essential for everyday life, such as disclosure of company information, provision of online services, and e-commerce. Any interruptions to these services cause major inconvenience to the public.

Denial of service (DoS) is a type of attack designed to block ordinary users' access to a network resource. The attack side and the defense side engage in a kind of arms race, in which the defense side raises countermeasures to block the attacks, and the attackers develop methods of evading them. DoS attacks and countermeasures are continuing to increase in both number and complexity.

In this paper, we focus on HTTP GET Flood attacks, controlled manually in real time using a DoS tool designed to degrade a service. An HTTP GET Flood attack is a serious application-level DoS attack that exhausts server resources by flooding the targeted system with a deluge of HTTP GET requests [1]. DoS tools, which are available to the public via the Internet, are one type of weapon used in HTTP GET Flood attacks [2].

Service-degrading attacks result in slower average service to regular users but do not lead to total service disruption [3]. To degrade a service, the perpetrator controls the attack directly from their computer step by step: launching the attack, monitoring the targeted server status, adjusting the attack parameters, and applying other strategies.

DoS tools usually include a function for monitoring the status of the targeted server [4] that allows the attacker to confirm the effectiveness of the current attack and the defense activities of the server, and thus plan further attacks. By observing the effectiveness of the current attack, the attacker can adjust the attack intensity to match the server's status. Depending on the defense response, the perpetrator can judge whether their attack is being mitigated using certain techniques such as filtering or rate-limiting [5]. If the attacker observes a defensive response to the attack, the attacker can respond by changing the attack method [6], [7], [8], [9], abandoning the attack [3], or targeting a more vulnerable server [3].

In this paper, we propose a defense method that 1) allows the server to maintain its service to users relatively unaffected by the attacks, 2) responds optimally to each attacker, and 3) impedes the attacker's ability to detect a defensive response. 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 explain how we define and classify DoS attacks. In Section 3, we show the problem of the previous studies. In Section 4, we state an attack scenario and our idea. Section 5 describes the implementation of our proposal. In Section 6, we perform an evaluation of our method. In Section 7 we show and

[1] Faculty of Engineering, Toyohashi University of Technology, Toyohashi, Aich 441–8580, Japan
[2] Internet Initiative Japan Inc., Iidabashi Grand Bloom, Chiyoda, Tokyo 102–0071, Japan
[a)] kobayashi@ppl.cs.tut.ac.jp

discuss the results. In Section 8, we discuss the feasibility and the limitations of our proposal. Section 9 concludes our paper.

2. DoS Attack and Defense

In this section, we survey DoS attacks and defense mechanisms. We begin our survey by setting our definitions and thereafter focus on instances that fall within the scope of our study.

2.1 Classification of DoS Attacks

Figure 1 shows our classification of DoS attacks. The categories are not always exclusive, since an attack can correspond to one or more of the classified items. As seen in Fig. 1, we explain each of the classes and specify the scope of this study.

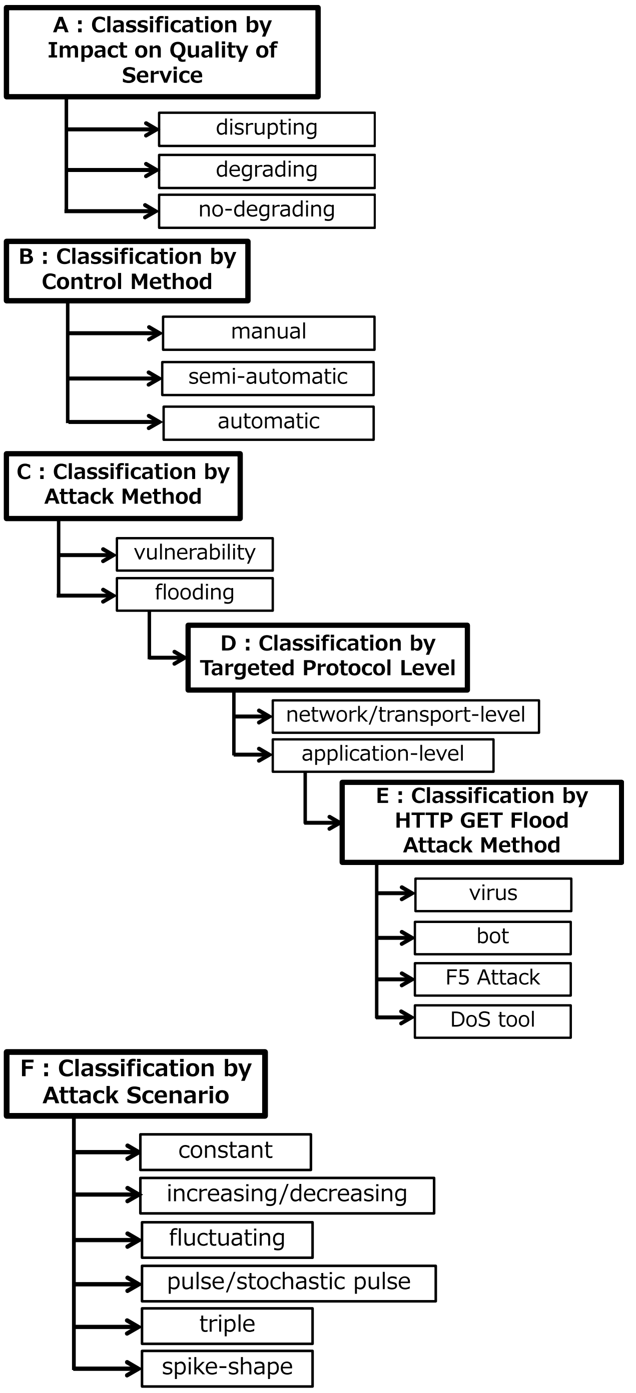

A : Classification by Impact on Quality of Service
disrupting
degrading
no-degrading
B : Classification by Control Method
manual
semi-automatic
automatic
C : Classification by Attack Method
vulnerability
flooding
D : Classification by Targeted Protocol Level
network/transport-level
application-level
E : Classification by HTTP GET Flood Attack Method
virus
bot
F5 Attack
DoS tool
F : Classification by Attack Scenario
constant
increasing/decreasing
fluctuating
pulse/stochastic pulse
triple
spike-shape

Fig. 1 Attack classification.

A: Impact on Quality of Service

DoS attacks negatively affect users' access to services by disrupting or degrading the quality of service [10], [11], [12]. "Disruptive attacks" completely block users from accessing the service. "Degrading attacks" substantially degrade the service without completely blocking it, causing dissatisfaction on the part of its users. Degrading attacks are more difficult to detect than disruptive attacks [3], [12], [13] since, unlike a totally disrupted service, a degraded service remains partially available. We propose the term "non-degrading attacks," indicating that the attacks do not entirely degrade the service, to be able to classify "energy DoS attacks" [13], [14], [15] from the viewpoint of their impact on quality of service. "Energy DoS attacks" aim to drain the energy of the server while keeping the attacks as stealthy as possible. Since energy DoS attacks do not always degrade the service [15], they can be classified as either degrading or non-degrading attacks. In this paper, for reasons of simplicity, we do not describe non-degrading attacks.

B: Control Method

There are three kinds of DoS attack control methods: Manual, Semi-Automatic, and Automatic [10]. In manual control, a person directly controls the attack step by step, for example, by determining the targeted server and the attack method, starting/finishing the attack, monitoring the server's status, adjusting the attack's intensity, and fending off any defense mechanisms. With automatic control, on the other hand, a program controls the attack. A program control-based attack involves the use of compromised clients running malicious software (malware) that surreptitiously penetrates the targeted server. In a semi-automatic attack, a person and a program manage the attack together.

As noted in Section 1, our study focuses on proposing a system that prevents human control-based DoS attacks.

C: Attack Method

DoS attack methods are generally divided into vulnerability attacks and flooding attacks [1], [3], [4]. Vulnerability attacks exploit any weaknesses in the target server that have resulted from design errors or ambiguities. Flooding attacks, which are the most common, send a deluge of attack packets that exhaust the server's resources to the point where they negatively impact service utilization.

D: Targeted Protocol Level

Flooding attacks can be classified into two categories based on the targeted protocol level [1]. An attack can belong to one or both categories. "Network/transport-level attacks" attempt to exert a negative effect on the quality of service by exhausting bandwidth, router processing capacity, or network resources. "Application-level attacks" attempt to negatively affect the quality of service by exhausting a server's resources such as its CPU, memory, sockets, disk/database bandwidth, I/O bandwidth, and disk space. Unlike network/transport-level attacks, this type of attack is preventable on the server side.

In this paper, we focus on HTTP GET Flood attacks, which are one type of application-level flooding attack. In this type of attack, a large number of GET requests are sent to the server, placing a heavy load on it by forcing it to process them.

E: HTTP GET Flood Attack Method

The methods used for HTTP GET Flood attack are shown in [2]. It should be noted that most of these methods can also be used for launching other types of attacks.

Viruses: a computer virus running on a compromised client manages the attacks by selecting a victim, setting the parameters, launching the attacks, and so on. The attack behavior is pre-programmed into the computer virus. This type of attack is thus automatically controlled.

Bots: Command-and-control (C&C) servers are used by attackers to maintain communications with compromised systems within the target network and to send simple commands to bots infected with malware. The bots then launch attacks based on these commands. The parameters of the attacks can be configured using the commands. To hide communication between a C&C server and bots, the number of communications is minimized as needed. In this type of attack, a semi-automatic control method is used.

F5 attack: F5 attacks are performed manually by someone holding down the Web browser reload key (F5) on a keyboard. The reload function sends HTTP GET request packets to the URL currently being viewed in the browser. The F5 attacker confirms whether the attack is successful by browsing the relevant Web pages. An F5 attack can issue up to 30 GET requests per second [16].

DoS tools: A DoS tool attack, described above in Section 1, is carried out manually by someone who wants to attack the target server. The attacker runs a DoS tool and utilizes the functions included in the tool to mount the attack step by step. Unlike an F5 attack, the DoS tool attack can adjust the issue rate of GET requests without being limited by I/O devices, and can more precisely monitor the server status based on the responses received by the tool.

In this paper, we focus on DoS tool attacks that are performed manually by someone who is monitoring the server status.

F: Attack Scenario

There are different types of attack scenarios that are used for evaluating defense mechanisms or are used by DoS tools. The name of each scenario indicates the control method of the attack rate. In an HTTP GET Flooding attack, the attack rate is the number of GET requests per unit time.

Examples of these scenarios are shown in Fig. 1. In the "Constant scenario" [2], [7], [10], [17], [18], the attack rate is constant. This scenario is often used with DoS tools (e.g., MyDoom, BlueCode, Netsky, and BlackEnergy). The "Increasing scenario" [7], [10], [17] applies a rising attack rate. The small rate of increase of the attack rate delays detection of the attack. On the other hand, the "Decreasing scenario" [17] applies a declining attack rate. The "Fluctuating scenario" [10] controls the attack rate according to a schedule written into the malicious code or in response to the targeted server's behavior. The "Pulsed scenario" [12], [17] regularly switches the attack rate between a constant value and zero; and the "Stochastic pulsed scenario" [7] continuously and randomly changes the parameters forming each pulse. The "Triple scenario" [17] alternates two phases: increasing and decreasing of the attack rate; and the "Spike-shaped" scenario [18] maintains a spike-shaped attack rate.

2.2 Classification of Defenses

As shown in the following itemization, defense methods can be classified into preventive methods and reactive methods. The reactive methods can be further classified into attack detection methods and attack mitigation [*1] methods [3].

- Preventive methods
- Reactive methods
  - Detection methods
  - Mitigation methods

The preventive method pre-emptively eliminates the risk of DoS attacks. Examples include the installation of new software with improved security levels; or to alleviate vulnerability attacks, remedial action can be taken to eradicate existing errors.

On the other hand, the reactive method alleviates the influence of attacks that have already occurred by detecting and mitigating them. Detection is the first step and mitigation is the second step. The detection method identifies the attackers or the attack packets, and the mitigation method works directly on the attack packets to prevent them from affecting the service. The attackers cannot detect the level of activity of the detection method, but they do notice that mitigation is being applied.

Some studies focus on proposing an advanced analysis method to detect attacks (e.g., pattern detection and anomaly detection). Below are examples of detection methods for HTTP Flooding attacks. Yatagai et al. [2] propose a detection method which focuses on the order of accessed pages and the correlation between the information content of individual pages and the average time taken by genuine human users to browse them. Xie et al. [7] propose a detection method that utilizes a learning model based on principal component analysis (PCA), independent component analysis (ICA), and a hidden semi-Markov model (HsMM). Stevanovic et al. [19] propose a detection method which classifies clients into malicious or non-malicious by means of unsupervised neural network learning algorithms: a self-organizing map (SOM) and modified adaptive resonance theory 2 (Modified ART2). Das et al. [20] studied the detection of different types of attacks and propose a method based on a computation employing rate of arrival of HTTP requests (HAr), computation and comparison with the legitimate pattern disagreement (PD) value, and the clustering of the packet header datasets of HTTP requests.

Filtering and rate-limiting in the firewall or the router are examples of attack mitigation methods [5], [21], [22]. The filtering method simply drops the packets that have been identified by the detection method. Rate-limiting involves two strategies: "Traffic shaping" and "Traffic policing." Both limit the maximum bandwidth of the attacks. The difference between these strategies is that traffic shaping uses a dedicated buffer to delay the transport of extra packets that exceed the maximum bandwidth, whereas traffic policing just drops the extra packets. Traffic shaping holds the

[*1] Mirkovic et al. use the term "response" instead of the term "mitigation" for expressing a certain method. However, in this paper, the term "response" is used to describe communication from the server to the client. Therefore, we use the term "mitigation" instead of the term "response" to avoid unnecessary confusion.

extra packets in a buffer, and thus requires memory space. When the buffer space is full, due to excess attacks, the extra packets are just dropped, in the same way as for traffic policing. This means that traffic shaping will not work effectively if the traffic constantly exceeds the maximum bandwidth. Garg et al. [23] propose a mitigation method that performs both "Rate Control" and "Window Control." Rate control limits the flow rates of the attacks in rate-limiting fashion, and window control limits the number of requests that consume a resource simultaneously. The extra requests are simply dropped.

In the light of the above, in this paper we focus on a reactive method for performing both detection and mitigation. Although detection is also important, we place more emphasis on mitigation, since we focus on the attackers' behavior of monitoring the targeted server during the attacks. Improvement of the detection step awaits future work.

2.3 Targeted Server Monitoring

DoS tools provide a function for monitoring the status of the targeted server. Xin et al. [4] analyzed 80 DoS tools including 60 DoS programs and 20 DoS scripts, and categorized the DoS attack functions provided as part of these DoS tools. They classified 29 DoS attack functions into six categories. One of these categories is the server monitoring function, which receives the response data from the targeted server to confirm the targeted server status and enable further attacks. The above classification of DoS attack functions does not mean that every DoS tool has a monitoring function, or that every attacker monitors the targeted server status. Targeted server monitoring is simply one option in the DoS attackers' toolkit.

If the attackers monitor a targeted server, they can confirm the impact of the current attack on the quality of service and detect any activation of defense mechanisms [13]. During degrading attacks, it is likely that the attackers will confirm the service impact, since it is necessary to adjust the attack intensity to a level at which the service is not disrupted but causes dissatisfaction on the part of its users. Even in disruptive attacks, if the attackers aim to perform cost-effective attacks [12], it is a good strategy to gradually increase the attack strength while confirming the impact on service up to the point where service disruption occurs. It is also important for the attackers to detect any defense activities, since it is pointless to continue attacks that have already been mitigated by the defense mechanism.

Kuzmanovic et al. [24] investigated the response time under overload without any defense activities. Their results showed that when the server is overloaded, the response delay time is randomly determined, and as the load increases, the throughput decreases. It therefore appears that the attackers monitoring the server status expect the above response during degrading attacks.

However, once the defense mechanism mitigates the attacks, the pattern of responses changes. For example, if a filtering method is activated to mitigate the degrading attacks, all the attack packets timeout. In another example, if rate-limiting is activated, a proportion of the attack packets are responded to normally and the others are dropped.

There is a clear contrast between the situations before and after mitigation. This means that if the attacker tries to degrade the service and monitors the server status, defense in the form of filtering will be noticed.

Considering the above, we focus on targeted server monitoring.

2.4 Our Previous Work

In our previous work [25], [26] we proposed mitigation methods for F5 attacks, which is a type of HTTP GET Flooding attack. An F5 attacker can confirm whether the attack is successful by browsing the requisite web pages. If a defense method such as filtering or rate-limiting drops all or part of the attack packets, the attackers may notice that their attacks are being defended against and will therefore attempt to respond accordingly [6], [7], [8], [9].

In our first paper [25], we focused on the above point and proposed a defense method that would allow the server to continue providing a service to its users while hiding its defensive response from the F5 attacker. With this method, the Web Server is partitioned into a decoy machine and a normal machine. The decoy machine is used by the attackers, while the normal machine is used by the legitimate users. In an F5 attack, the attack rate is constant and determined by the limitations of I/O devices, and the targeted server is meanwhile monitored using a Web browser. Therefore, this method detects the F5 attacker simply by checking whether the attack rate exceeds a certain threshold, and assumes each attacker aims for almost the same target error rate of HTTP GET requests, with the error rate being used as an index of the impact of the attacks on the service. The attackers' requests are then forwarded to the decoy machine, while normal users' requests are forwarded to the normal machine. This approach minimizes the inconvenience to legitimate users. To deceive the attackers into believing their attack is successful, the resources of the decoy machine are controlled during runtime so that the error rate of requests reaches a target value that is set by an administrator.

Our first paper focused on the use of the correlation between CPU resources and request errors to maintain CPU resources at the targeted error rate. However, which resource the request error rate is related to depends on the Web Server. If the sole correlation is between memory resources and request errors, however, the request error rate cannot be controlled using our first proposal. In our second paper [26], we therefore focused on both the CPU and memory resources and proposed a method that dynamically selects and controls one of the two resources.

3. Problem

In this paper, we focus on tool attacks that neither of the proposals in our previous studies shown in Section 2.4 can be assumed to mitigate. Both the previous methods supply multiple attackers with the same provision because the F5 attacker's behaviors are limited by the I/O devices, leading to the same and constant attack rate. However, unlike F5 attackers, attackers who launch DoS tool attacks apply different attack rates [17], [27] and, in addition, can adjust the attack rate to match different targeted error rates while monitoring the current server status [4], [13]. This makes it difficult for the methods proposed in our previous studies to deceive DoS tool attackers into believing their attacks are successful.

Similarly conventional studies other than our previous studies can not deal with the attacker monitoring. One example is as follows.

A. Sardana and R.C. Joshi [28] propose a defense method, which is called Autonomous Dynamic Honeypot Routing (ADHR). They focus on the DDoS attacks on FTP servers, where the attack rate is constant that is one of the attack scenarios shown in Section 2.1. ADHR includes detectors, honeypots, and servers. The detectors perform entropy-based attack detection. The detected attacks are redirected to the honeypots and the legitimate clients are redirected to the FTP servers. This isolation of the attacks protects the legitimate clients from the bad influence of the attacks. Moreover, this method dynamically provides the adequate number of honeypots, depending on the attack load. For example, if the load of attack changes from low to high, the number of honeypots changes from low to moderate. This dynamic operation aims to get information for trace back by allowing the attacks to interact with honeypots.

We show the important differences between the ADHR and our method as follows. In the study of ADHR, ADHR adaptively controls the number of honeypots in accordance with the attack strength to maintain connection between the attackers and the honeypots, and each attacker launches the constant attack independently of the FTP server status. In our study, the Control Machine controls the CPU resources of each Decoy Machine to achieve the target error rate, and each attacker gradually increases the attack strength until the error rate reaches the target while monitoring the server status.

The advantage against the above ADHR is that our proposal can suppress the attack strength of the degrading attacker which uses the monitoring function, resulting in less CPU resource being wasted by the attack. The attacker gradually increases the attack rate while monitoring the server status until the attack rate reaches the target. On the other hand, our method controls the resource of the Decoy to show the target error rate to the attacker in an early time. Therefore, the attacker stops to increase the attack rate while the attack strength is weak. The weak attack does not substantially flood the server resource, allowing us to reduce the server resources needed for defense from attack.

4. Proposal

4.1 Attack Scenario

This paper focuses on the HTTP GET Flood attacks controlled by a human operator using a DoS tool for degrading the service. The attack rate depends on each tool attacker [17], [27], and the attackers can control the attack rate for the different targeted error rates while monitoring the targeted server status [4], [13]. Therefore, in this section, we assume an attack scenario suitable for the above attack. In this paper, the attack scenario is as method of how to control the attack rate.

At first, we focus on the constant and increasing scenarios among the ones shown in Section 2.1. The constant scenario is used in the realistic DoS tools and for the evaluation of the defense methods. On the other hand, the increasing scenario can delay the detection by gradually increasing the attack rate, going well with the goal of the degrading attack. However, both scenarios lack the ability to monitor server status and the attack rate is configured independently of the server status.

To address the above problem, we introduce a new scenario which repeats the following steps:

1) First, launches the attack with the constant attack rate and monitors the server status for a while

2) Then, it increases or decreases the attack rate if the monitored server status has not reach the targeted error rate.

This combines the features of the constant and increasing scenarios and also incorporates an adaptive attack rate based on the server monitoring results. The attacker can monitor the response of the targeted server via the functions of the DoS tool. However, since the response is generally not stable, especially for an overload status, it takes time to accurately judge the server status. Therefore, this scenario defines a time period for the determination of the current server status.

In this scenario, the attack rate is constant in the short term, and if the attack rate at the start is low enough to avoid the defense activity, the attack rate is increasing in the long term. Hence, an attacker who monitors the server status will expect that the variance in the response delay time is random and that the error rate of the server will tend to increase as the attack rate increases. If the current server status is different than expected, then the attacker will notice some defensive activities, such as packet dropping.

We confirm here the definition of a DoS attack. DoS attacks are requests with an intention to disturb services. A request without such an intention cannot be called a DoS attack, even if the request puts a load on the server great enough to affect the web service. However, it is difficult for service providers to detect whether a request intends to disturb the Web Service. Therefore, in this paper, we regard an unfavorable request to the server side as an attack, and the client who launches the attack is judged to be an attacker, even though it is unclear whether the attack has an intention to disturb services.

4.2 Main Idea

We propose a defense method for HTTP GET Flood attacks that 1) allows the server to maintain quality of service for normal users with little or no influence from attacks, 2) provides each attacker with a different expected server status, and 3) hinders the attacker's operation in order to determine a defense mechanism that will effectively mitigate their attack.

Figure 2 illustrates the set-up and operation of the proposed

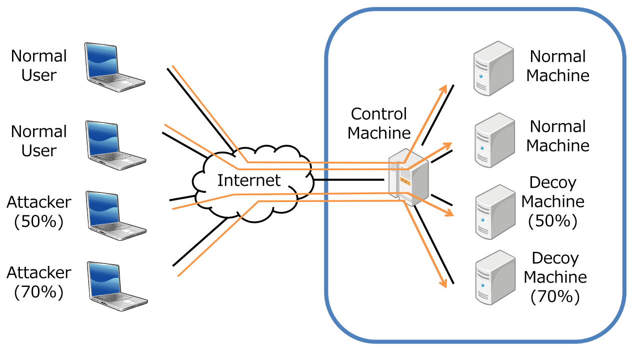


Fig. 2 Main idea.

method. As per the figure, the Web Server contains a Control Machine, Normal Machines, and Decoy Machines. The Control Machine is set between the connection to the Internet and the other machines, and it performs operations for attack detection and mitigation. The Normal Machines supply normal users with the service, while the Decoy Machines fool the attackers. This arrangement ensures that normal users are barely affected by attacks because resources for normal users and resources for attackers are separate. The Decoy Machines generate different error rates through CPU resource control, presenting different targeted error rates to the attackers. In the figure, the percentage in brackets of each client or machine is their targeted error rate. The CPU resource control essentially places each Decoy Machine into an overload status so the server response is random and tends to decline as the load increases, making it appear to attackers as if the attack is successfully degrading the service without being detected.

The defense operation is outlined as follows. In the first step, the Control Machine classifies the clients as either normal users or attackers. Moreover, it determines for each attacker whether it is in a transition period, in which it increases the attack rate for the long-term targeted server state. In the second step, the Control Machine forwards normal users to the Normal Machines, and attackers to the Decoy Machines: an attacker with a targeted error rate of $x\%$ is forwarded to a Decoy Machine with the same targeted error rate, $x\%$. Simultaneously, the Control Machine controls the server resources for each Decoy Machine, so that they will each arrive at the targeted error rates.

It is difficult to directly identify the targeted error rate, $x\%$, of each attacker since each attack rate is gradually increasing until it reaches a target value. Therefore, like the attackers, the Control Machine gradually adjusts how it forwards attackers to Decoy Machines. If the attack rate remains constant, then the attacker continues to be forwarded to the current Decoy Machine. Otherwise, the Control Machine adjusts the forwarding destination from the current Decoy Machine to another Decoy Machine with a higher targeted error rate.

The error rate on the Decoy Machine side is determined by the relationship between the attack rate and the server resources. During the attack, the attacker gradually increases the attack rate. During the initial phase of this increase, the Control Machine begins to control the server resources of the Decoy Machine. When the error rate of the Decoy Machine reaches the targeted error rate, the attacker maintains the attack rate. As a result, the proposed method can limit the attack to a low attack rate.

The advantage points of the proposed method are as follows. Firstly, the proposed method pretends as if the DoS attack is completed to prevent the attackers from becoming aware of the defense activity and conducting more severe or sophisticated attacks. Secondly, the proposed method suppresses the attack strength of the attackers by starting to control the error rate immediately after the attackers start to gradually increase the attack strength.

5. Implementation

5.1 Entire Structure

In the proposed method, we prepare three different types of machines, namely the Control Machine, Normal Machines, and Decoy Machines. However, it is not practical to equip costly physical machines for brief attacks. Thus, instead of physical machines, we render these machines as kernel-based virtual machines (KVM), which are a form of virtualization software, and implement multiple units of virtual machines on a host machine [29].

We present a schematic of the entire structure of the proposed method in **Fig. 3**. Here, we generate four virtual machines on a host machine, and operate the host machine as a Control Machine and the virtual machines as either Normal Machines and Decoy Machines. The Control Machine is set up between the external connection to the Internet and the virtual machines, including the Normal Machines and Decoy Machines. Every client transmits requests to the Control Machine, which then forwards the requests to one of the Normal Machines or Decoy Machines. Note that the Normal Machines and the Decoy Machines can both actually provide the service.

The Control Machine has two roles: load balancing of normal requests and management of defense against attack. This latter role can be further divided into attack detection and attack mitigation, based on the classification shown in Section 2.2. The above mentioned roles are performed by the following components of the Control Machine:

- Load balancing mechanism,
- Forwarding mechanism,
- Resource control mechanism, and
- Main Controller.

The load balancing, forwarding, and resource control mechanisms are implemented using existing techniques. The load balancing mechanism is used for load balancing of normal requests, while the forwarding mechanism and the resource control mechanism are used for attack mitigation. The Main Controller is dedicated software written in Python that is used for both attack detection and mitigation. The remainder of this section explains these

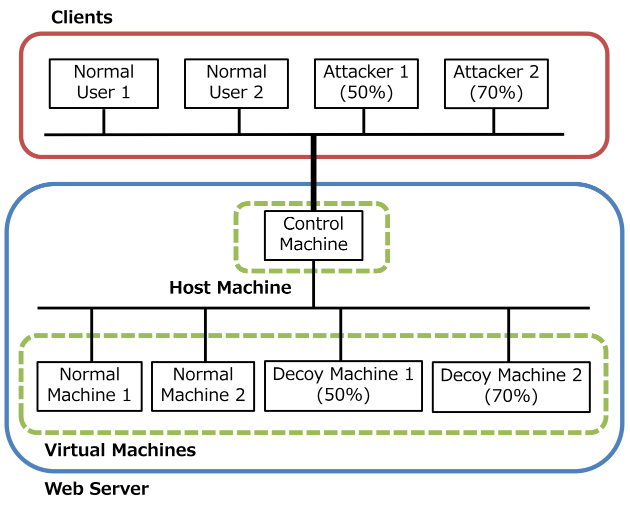


Fig. 3 Example of system implementation.

components in detail.

5.2 Load Balancing Mechanism for Normal Users

Load balancing is a well-known countermeasure for handling heavy website traffic. In this technique, the Control Machine operates as a load balancer for normal users, distributing them to a Normal Machine. If this distribution is performed properly, it is possible to avoid situations in which some of the Normal Machines become overloaded. Consequently, server resources can be more efficiently utilized, while multiplexing server equipment. In addition, if any Normal Machine fails, the service can be sustained using the remaining Normal Machines.

A load balancer can be constructed using dedicated hardware or software. However, since the former is expensive, it is not practical to introduce a hardware-based load balancer to a small-scale web server. Therefore, in this research, we use Apache and some Apache modules, called mod_proxy, mod_proxy_balancer, and mod_headers, on the Control Machine to implement a load balancer. In addition, we use a pending request counting algorithm that distributes a new client to the Normal Machine with the least number of pending requests, as well as a cookie stickiness function that maintains the established session by using a cookie.

The load balancer distributes clients only to servers that are registered in the load-balancing group, which is managed by a BalancerMember directive in mod_proxy. In the proposed method, the load-balancing group holds only the Normal Machines and so the Control Machine distributes normal users to one of the Normal Machines.

Load balancing is not applied to attackers, which are handled by the forwarding mechanism described in Section 5.3.

5.3 Forwarding Mechanism for Attackers

We use the NAT function of iptables to implement a forwarding mechanism for attackers. If a new attacker is detected and a Decoy Machine is selected as a forwarding destination, then the Main Controller executes iptables on the Control Machine to register a NAT mapping between the attacker and the Decoy Machine in the NAT table. The Control Machine forwards the attackers to the Decoy Machines according to the NAT table.

As shown in Section 4.2, the Control Machine gradually adjusts the forwarding destination of each attacker while confirming the attack rate. Therefore, if the forwarding destination of a registered NAT mapping is changed from the current Decoy Machine to another Decoy Machine with a higher targeted error rate, the Main Controller executes iptables to delete the NAT mapping and add a new NAT mapping between the attacker and the next Decoy Machine.

It should be noted that the NAT function is executed prior to load balancing. Thus, if the NAT table includes a NAT mapping between an attacker and a Decoy Machine, then the attacker is forwarded to the Decoy Machine based on the mapping, regardless of the action of load balancing by the Control Machine.

5.4 Resource Control Mechanism for Attackers

There are a number of server resources, as shown in Section 2. In this paper, in order to focus on the implementation of the proposed technique, we select CPU as a resource to be controlled.

We use the cgroups (control groups) function of a Linux kernel to control the resources of each Decoy Machine. The cgroups function groups processes and controls resources in a group unit. Each Decoy Machine is dealt with as a process group. The CPU utilization threshold of each process group is determined by two parameters: cpu.cfs_period_us and cpu.cfs_quota_us. They indicate time and are specified in microseconds. The cpu.cfs_period_us is the period in which CPU time is allocated to processes in a group. The cpu.cfs_quota_us is the maximum CPU time that all processes in a group can utilize during cpu.cfs_period_us. The cpu.cfs_quota_us can be set from 1,000 to 1,000,000, but a cpu.cfs_quota_us value of -1 has a different meaning, indicating that there is no constraint on CPU utilization. For example, if cpu.cfs_period_us is set to 100,000 and cpu.cfs_quota_us is 50,000, then the CPU utilization rate is restricted to a maximum of 50%.

5.5 Main Controller for Normal Users and Attackers

The Main Controller consists of an attack detection class and an attack mitigation class. We explain these two classes in detail in this subsection.

5.5.1 Attack Detection Class

Tcpdump is a software tool that can capture the contents of packets, which are needed for network traffic analysis. The attack detection class runs the tcpdump command on the Control Machine to capture the packets of clients, thus generating a request log.

In our assumed scenario, the attacker maintains a constant attack rate over a period, but regularly increases it to attain a targeted error rate. Therefore, the attack detection class regards as attackers those clients who, according to the request log, increase their request issue rate in the long term. It sends information about them to the attack mitigation class. If a client is an attacker, then the request issue rate is considered to be the attack rate.

However, the time from transmission of the request to its receipt is not always constant, especially in an overload state. This is similar to the distribution of the response delay time noted in Section 2.3. Although requests may be issued at a fixed interval on the client side, the request will not necessarily arrive at the same interval on the server side. Thus, if the requests follow the attack scenario assumed in our method on the client side, then the request issue rate may be increasing or decreasing in the short term, but will be increasing in the long term.

Considering the above issues, the attack detection class repeats the following steps for each client for every fixed global time period (*Period_global*):

1) Partition the *Period_global* into the multiple fixed local time period (*Period_local*);

2) Calculate the request issue rate for every *Period_local* by dividing the number of requests by the duration of a *Period_local*; [*2].

[*2] Although the second step performs the division, this can be simplified by eliminating the division and having the remaining steps treat the number of requests as the request issue rate. This modification does not affect the outputs of the attack detection class.

3) Find the maximum request issue rate among the above calculated the request issue rates in the *Period_global*; and
4) Regard the client as an attacker increasing the attack rate if the maximum request issue rate is larger than the previous maximum request issue rate by the predetermined threshold (*Th*).

5.5.2 Attack Mitigation Class

In the attack mitigation class, the attackers are classified as one of three types of attacker: a new attacker, a stable attacker, or a transitional attacker. A new attacker is a client that has been detected as an attacker because of an increase in their request issue rate (i.e., their attack rate), but has yet to be registered as an attacker by the attack mitigation class. A stable attacker is a registered attacker, meaning an attacker already registered in the attack mitigation class, which has, in a stable period, maintained a constant attack rate in the long term. Finally, a transitional attacker is a registered attacker in a transition period, who is in the process of increasing their attack rate in the long term.

The attack mitigation class contains two functions: a forwarding function and a resource control function. The forwarding function operates according to the type of attacker, as defined above. If information about an attacker increasing the attack rate is sent from the attack detection class, then the forwarding function checks whether the attacker is registered inside itself. If not, then it is a new attacker; otherwise, it is a transitional attacker. In the case of a new attacker, the forwarding function registers the attacker in itself and uses the forwarding mechanism (explained in Section 5.3) to begin forwarding the attacker to the Decoy Machine with the lowest targeted error rate among all the Decoy Machines. On the other hand, in the case of a transitional attacker, the forwarding function uses the forwarding mechanism to change the forwarding target from the current Decoy Machine to another Decoy Machine with a higher targeted error rate, if there is one. Finally, in the case of a stable attacker, the forwarding function does not perform any operation. However, the forwarding mechanism can forward stable attackers to Decoy Machines, as the forwarding targets of each stable attacker have already been registered in the NAT table.

The resource control function repeats the following operations at regular intervals:
1) Transmit a GET request to each Decoy Machine and receive the response;
2) Calculate the request error rate and compare it with the targeted error rate of each Decoy Machine; and
3) Use the resource control mechanism (explained in Section 5.4) to modify the CPU resources of each Decoy Machine to cause the request error rate to approach the targeted error rate.

5.6 System Operation

The Main Controller carries out attack detection constantly, but only performs attack mitigation if attackers are detected. In this subsection, we explain the two modes of system operation: the normal mode and the mitigation mode.

5.6.1 Normal Mode Operation

If there are no attackers, the system operates in the normal mode, serving only normal users. **Figure 4** shows the system during the normal operation mode. In this figure, the configuration of the system is the same as that shown in Fig. 3. In the normal mode, the Control Machine distributes normal users to the Normal Machines, based on the pending request counting algorithm and the cookie stickiness for load balancing.

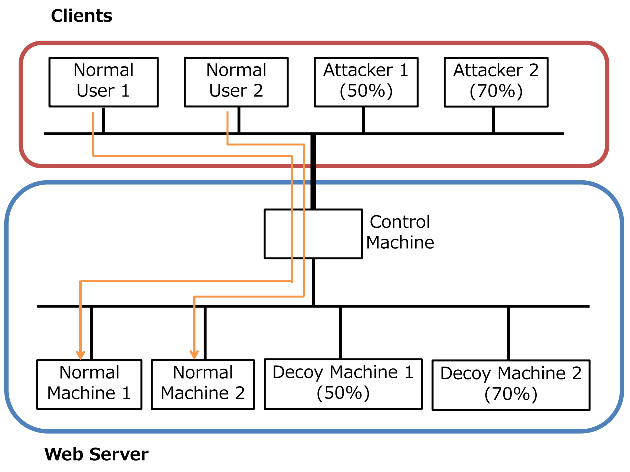


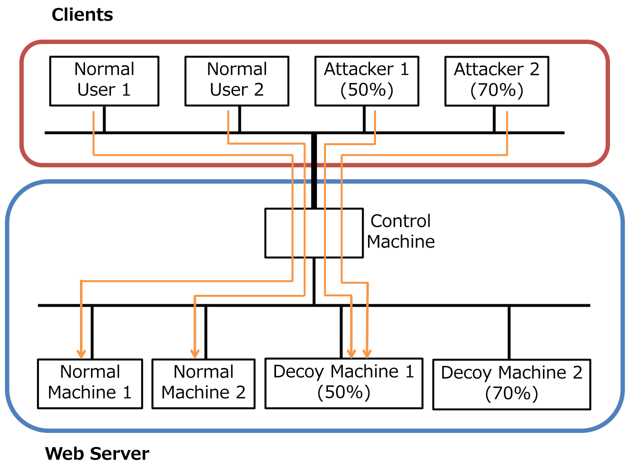
Fig. 4 Normal mode operation.

5.6.2 Mitigation Mode Operation

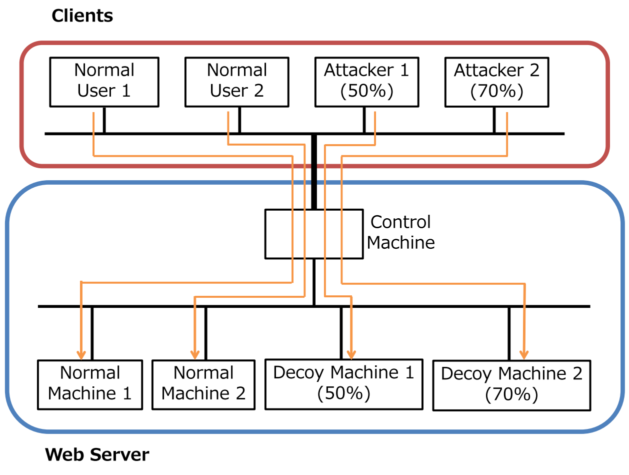
If there are attackers, the system operates in the mitigation mode for both normal users and attackers. **Figure 5** shows the system during the mitigation operation mode. Unlike in Fig. 4, there are attackers that follow the attack scenario outlined in Section 4.1.

Figure 5 (a) shows the operation for the new attackers. There are two normal users and two attackers. The normal users are distributed to the Normal Machines by the load balancing mechanism. The attackers launch their attacks and begin to increase the attack rate. They are detected for the first time and the information about them is sent to the mitigation class by the attack detection class. Since the detected attackers are yet to be registered, the mitigation class judges they are new attackers in order to register them and forward them to a Decoy Machine 1 with a targeted error rate of 50%. In addition, the mitigation class controls the CPU resource of the Decoy Machine 1 to make the request error rate close to the targeted error rate 50%.

Figure 5 (b) shows the operations for the stable attacker and the transitional attacker. After the operation shown in Fig. 5 (a), the Attacker 1 with an attack rate of 50% judges that the error rate has reached the targeted error rate, and thus maintains the attack rate. The Attacker 2 with a 70% attack rate judges the attack rate is insufficient, and increases their attack rate. Therefore, only the Attacker 2 is detected for the second time and the information of it is sent to the mitigation class by the attack detection class. Since the Attacker 2 is registered, the mitigation class judges the Attacker 2 is a transitional attacker, and changes the forwarding target to the Decoy Machine 2 with a targeted error rate of 70%. In addition, the mitigation class controls the CPU resources of the Decoy Machines to achieve each targeted error rate. It should be noted that the Attacker 1 is a stable attacker and kept to be forwarded to the Decoy Machine 1 based on the already registered NAT mapping.



(a) New Attackers.



(b) Stable Attacker and Transitional Attacker.

Fig. 5 Mitigation mode operation.

6. Evaluation

In this section, we describe the evaluation environment and method. **Figure 6** illustrates the evaluation environment, which includes multiple physical machines—the Web Server, Client Machine 1, and Client Machine 2, as well as virtual machines. **Table 1** presents the physical machine configuration while **Table 2** presents the virtual machine configuration.

The Web Server equips the virtual machines. The host machine operates as the Control Machine. The virtual machines provide the same Web Services and operate as either a Normal Machine or as a Decoy Machine. For our evaluation, we have the machines set up as Normal Machine 1, Normal Machine 2, Decoy Machine 1 (50%), and Decoy Machine 2 (70%), where the bracketed percentages refer to the targeted error rate of the respective Decoy Machine. Note that the details of the implementation of the Web Server are described in Section 5.1.

Client Machine 1 and 2 run the client simulator, which is dedicated software written in Python that simulates the operations of multiple clients with different IP addresses simultaneously. We used the client simulator to prepare the normal users and the attackers. Each attacker uses multiple processes to send GET requests similar to MyDoom. The normal users and the attackers

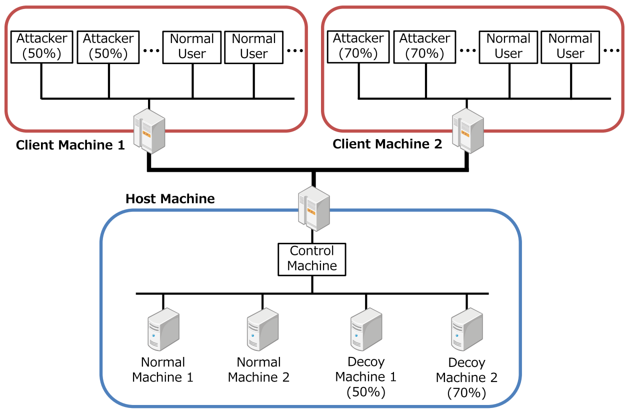


Fig. 6 Evaluation environment.

Table 1 Physical machine specification.

OS	Fedora Linux version 20 (Heisenbug)
CPU	Intel Core i7-4790K (4.00 GHz)
Memory	16 GBytes

Table 2 Virtual machine specification.

OS	Fedora Linux version 20 (Heisenbug)
Memory	1,024 MBytes

are denoted as Normal User, Attacker (50%), and Attacker (70%), where the bracketed percentage refers to the targeted error rate of the respective Attacker.

It should be noted that the client simulator was built solely for the purpose of evaluation of a new defense mechanism and not to attack existing services. Moreover, the safety of the evaluation is guaranteed since there is no unknown code in the client simulator and the evaluation environment was always disconnected from the network.

The client simulator on the Client Machine can generate up to twenty clients on each Client Machine: ten normal users and ten attackers. The maximum number of clients depends on the performance of the physical machine. In order to confirm the effect of number of clients on the error rate, we evaluated two types of cases. In the first case, there are one Normal User and one Attacker (50%) on the Client Machine 1, and one Normal User and one Attacker (70%) on the Client Machine 2. On the other hand, the second case has the following: ten Normal Users and ten Attackers (50%) on the Client Machine 1, and ten Normal Users and ten Attackers (70%) on the Client Machine 2.

There are many other parameters in the evaluation. Therefore, we evaluated our proposed method by varying the parameters and determining an appropriate combination of them, as follows.

A normal user issues one request per second, which is the minimum issue rate for a client generated by the simulator. An attacker follows the attack scenario described in Section 4.1. There are two parameters for an attack scenario: *Increment* and *decrement*. *Increment* and *decrement* are the increment and decrement of the attack rate, respectively, and are the same value: the product of *Diff_error_rate_attack* and α. *Diff_error_rate_attack* is the difference between the current and targeted error rate. α is a chosen constant value: 1 if the number of the attacker on a physical machine is one, 0.1 if it is ten. The latter value is smaller since it is limited by the performance of the physical machine.

The Main Controller on the Control Machine carries out the attack detection. The parameters for the attack detection are *Period_local*, *Period_global*, and *Th*. *Period_local* is the period in which the issue rate is calculated, and is set to 3 seconds for the simulation. *Period_global* is the period in which the maximum issue rate is determined, and is set to 60 seconds for the simulation. *Th* is a threshold. If the current maximum issue rate is larger than the previous maximum request issue rate by *Th*, then the client is regarded as an attacker increasing the attack rate. The details of the above parameters are described in Section 5.5.1. We prepare two types of *Th*: *Th_50* and *Th_70*, which are the thresholds for detecting the Attackers with a targeted error rate 50% and 70%, respectively. *Th_50* and *Th_70* are set to 100 and 150, respectively, if there is one attacker on a physical machine, and 10 and 15 if there are ten attackers.

Parameters for the resource control mechanism are cpu.cfs_period_us, which is the period in which CPU time is allocated to processes in a group, and cpu.cfs_quota_us, which is the maximum CPU time that all processes in a group can utilize during cpu.cfs_period_us. The cpu.cfs_period_us was set to 100,000. The resource control mechanism dynamically incremented or decremented cpu.cfs_quota_us by the product of 30 and *Diff_error_rate_defense*, where 30 was a set value and *Diff_error_rate_defense* is the difference between the current and targeted error rate.

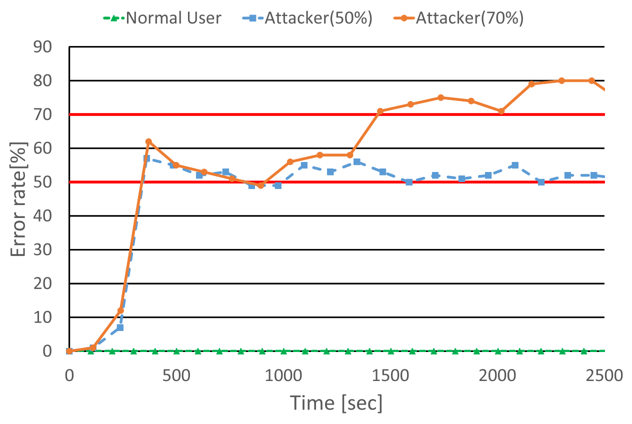
The next section describes the evaluation results of our proposed method for the above parameters.

7. Results and Discussion

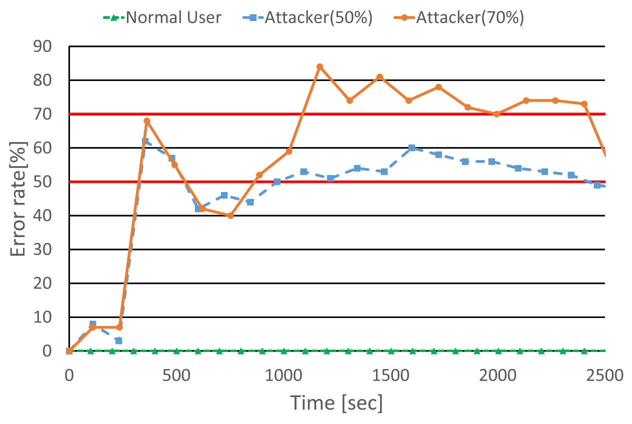
Our experimental results changed with each evaluation, as a result of the instability of the Web Server in overload status. Thus, we performed the same evaluation five times to confirm the change in results, and chose two representative cases from them that had relatively smaller and larger error rate variations.

Figures 7 and **8** show the request error rates of the Web Server. For each figure, (a) and (b) are for the cases with smaller and larger error rate variations, respectively. The request error rate was acquired from the log file of each client. There is a great deal of similarity between the two figures. The key difference between them is the number of clients: four clients in Fig. 7 and forty clients in Fig. 8. In the figures, the vertical axis indicates the error rate in the requests while the horizontal axis displays the time scale. The three types of plotted lines indicate the Normal User, the Attacker with a targeted error rate of 50%, and the Attacker with a targeted error rate of 70%, respectively.

Figure 7 confirms that the error rate of each attacker approaches their targeted error rates, though the transition of the error rate differs between (a) and (b). Moreover, it is apparent that the error rate of the normal users is not affected by the attackers. The attackers are initially forwarded to Normal Machine 1 or 2 and the attack rate is increased. Then, the attackers are detected by the attack detection class and forwarded to Decoy Machine 1 (50%). The resource of Decoy Machine 1 (50%) is controlled by the attack mitigation class. Thus, the error rate of the attackers increases to approach the targeted error rate of 50%. Next, the



(a) Smaller Error Rate Variation.

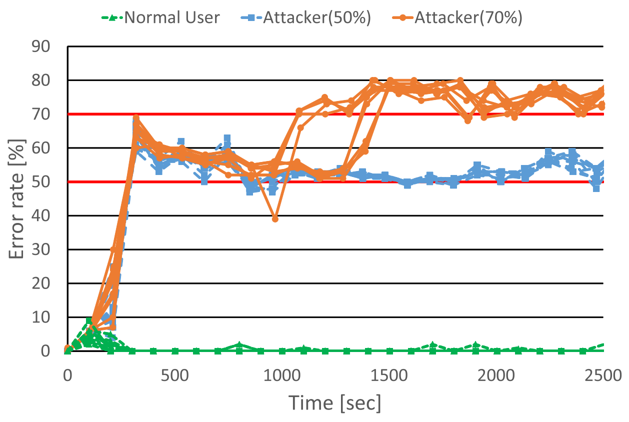


(b) Larger Error Rate Variation.

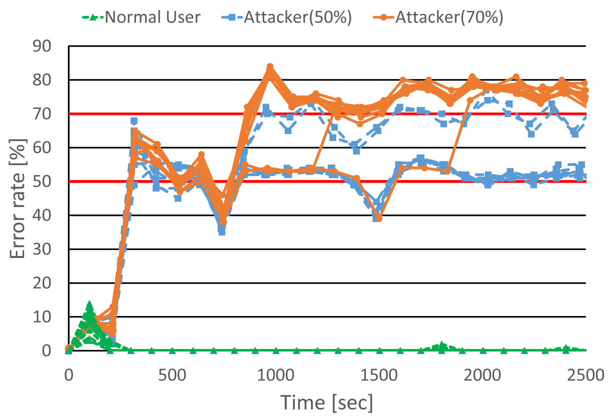
Fig. 7 Request error rate (4 clients).

Attackers (70%) are detected and forwarded to Decoy Machine 2 (70%), the resource of which is controlled by the attack mitigation class. As a result, the error rates of the Attackers (50%) and the Attackers (70%) remain around 50% and 70%, respectively. This confirms that the Web Server can provide each attacker with a different expected server status.

The overall trend shown in Fig. 8 is similar to that shown in Fig. 7. The differences between the figures are as follows. As in Fig. 8 (b), two Attackers (50%) are incorrectly forwarded to the Decoy Machine 2 (70%). This is because the issue rate observed on the Web Server side is larger than the issue rate on the attacker side due to the instability of the Web Server. Although these Attackers (50%) reduce the attack rate but the error rate does not decrease due to the resource control of the Decoy Machine 2 (70%). There is a period during which the error rate is larger than the targeted error rate of 50% by about 20 points. This is because the arrival timing of the requests sometimes changes significantly, hindering the precise measurement of the error rate. The error rate of the normal users is slightly affected during the short time in which the attackers are not detected although it is not usually affected by the attackers. The attackers with the same targeted error rate are usually forwarded to the same Decoy Machine, however, the detailed transition of the error rate depends on each attacker. This is because the error rate of each attacker is calculated using their log file, which includes only part of all responses from the Web Server. However, our mechanism can gen-

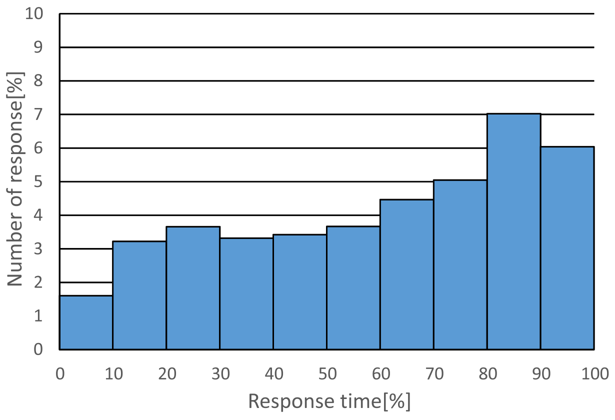


(a) Smaller Error Rate Variation.

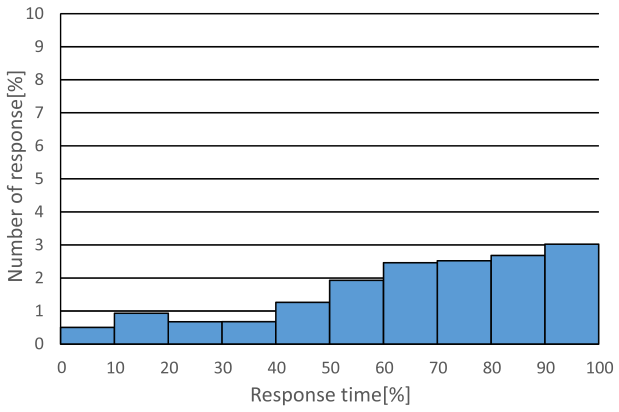


(b) Larger Error Rate Variation.

Fig. 8 Request error rate (40 clients).



(a) Attacker (50%).



(b) Attacker (70%).

Fig. 9 Distribution of response time.

erally maintain functions when the number of clients increases. Moreover, this instability of the defense mechanism may reflect the unstable operation of the server in overload status.

Figures 7 and 8 show only the request error rate, in which response time exceeds a timeout time. The request error rate alone cannot explain the effect on response time. Therefore, we examined the log file of the attackers in the case of Fig. 7 (a) to confirm the distribution of the response time. Each response time is measured by using time module in the Python library, which is inserted into the points where the client sends a request and receives the response of it.

It should be noted that the above measurement method is sufficiently effective in this study even if the measurement accuracy is not very high due to the usage of the time module inserted into the client simulator. This is because, in this study, the purpose of the evaluation is not to examine the precise response time in general circumstances, but to confirm the response of the Web Server with the proposed method is random and tends to decline as the load increases. The results of the measurement sufficiently performed the precision sufficient for the above purpose as shown below.

The results of the above measurement are illustrated in **Fig. 9**, where (a) and (b) respectively show the distributions of the response time of Attacker (50%) and Attacker (70%) during the period from about 2,200 seconds to 2,300 seconds. The horizontal axis shows the response time as a percentage of the timeout time. The vertical axis indicates the number of responses, normalized to the total number of responses and number of no responses, where "no response" means there was no response corresponding to a request sent by a client. Each bar indicates the case in which the response time is between *x*% and *x*+10% of the timeout time, where *x* is a multiple of 10 between 0 and 90 [*3].

Figure 9 confirms that the response times are widely distributed between 0% and 100%. This is because the Web Server in overload delays the response time and this delay is randomly determined. Figure 9 (a) and (b) confirm that, as the error rate increases, the number of responses between 0% and 100% decreases, although the delay time remains irregularly distributed in each case.

In order to confirm the advantage of our mechanism against the existing simple mechanism, we performed the evaluation of the filtering method. In the evaluation, once an attacker is identified by the same detection method as our mechanism, all the packets of the detected attacker are filtered out.

Figures 10 and **11** show the request error rates of the Web Server with the filtering method. The figures correspond to Fig. 7 and Fig. 8, respectively. However, since the filtering method does not have the Decoy Machines in overload status, the figures do not have smaller and larger error rate variations.

[*3] There are cases where the measured response time exceeds a timeout time even if the client receives the response. This is because the time module is used for the measurement as already mentioned.

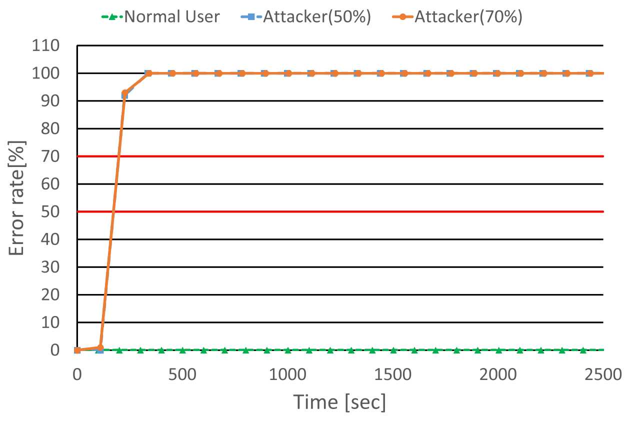


Fig. 10 Request error rate (Filtering, 4 clients).

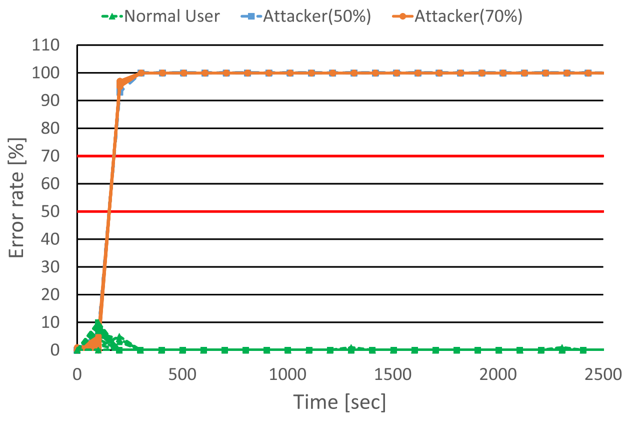


Fig. 11 Request error rate (Filtering, 40 clients).

Figures 10 and 11 confirm that the error rate of the normal users is not affected all the time or slightly affected until all the attackers are forwarded to the Decoy Machines. In addition, the figures confirm that the error rate of each attacker is rapidly changed from zero or a slight amount to 100% and then maintains the value. This indicates that there is no time period during which the response times are widely distributed, although each attacker gradually controls the attack rate while monitoring the server status. Therefore, the attacker can judge that their attack is being mitigated by certain defense method such as filtering.

8. Feasibility and Limitation

8.1 Decoy Machine

We discuss the load of the Decoy Machine. In our method, the host machine includes the Control Machine, Normal Machines, and Decoy Machines. However, the load of the Decoy Machines does not make the Normal Machines slower although the Decoy Machines receive the DoS packets. We show the reasons and the proof below.

Firstly, the load of each Decoy Machine is high while the DoS packets are received. However, as noted in Section 4.2, our proposal can suppress the attack strength, coming to the less resource needed for the Decoy Machine. In addition, we can set the maximum CPU time in the cgroups function to limit the maximum amount of resources that can be utilized by the Decoy Machines. Therefore, we can provide sufficient resources for the Normal Machines.

Secondly, our method achieves the target error rate on each Decoy Machine and suppresses the attack strength of DoS packets received by the Decoy Machine. Although the attack strength is harmful to the Decoy Machine, it is harmless for the host machine including the Control Machine, the Decoy Machine, and the Normal Machines. Therefore, the DoS packets do not negatively affect the Control Machine nor the Normal Machine on the host machine.

Lastly, the evaluation results confirm the following. When the Normal Machines receive the DoS packets, they are negatively affected. However, after the DoS packets are detected and forwarded to the Decoy Machines, the Normal Machines are not negatively affected. The above evaluation results show that it does not make the Normal Machines slower to implement both the Decoy Machines and the Normal Machines on the same host.

8.2 Control Machine

We explain the relationship between the Control Machine and a single point of failure. Generally, there are multiple SPOFs on a system: OS, application, disk, card, I/O device, power supply device, load balancer, and so on. The ways of eliminating SPOFs differ from each other. In addition, they require additional costs and may incur other new SPOFs. Therefore, all of SPOFs are not always eliminated. The approach to them depends on each system.

For example, a HDD becomes a SPOF. We can use a RAID technology to multiplex a HDD. However, the level and the cost of redundancy are different from RAID level to RAID level. The selection of level depends on each system. In addition, a RAID device is an additional cost and becomes a new SPOF.

Another example is a load balancer, which can multiplex a whole host with SPOFs. There are a single and multiplexed load balancers. The differences between them are cost and availability. Therefore, the service providers provide the single and multiplexed load balancers with lower and higher charges, respectively. If the priority of the cost is relatively higher than the availability for the system, a single load balancer is selected. Otherwise, a multiplexed one is selected.

In this paper, since the Control Machine is a single component through which all packets go in the system, it becomes a SPOF. However, it is not an only SPOF. The host machine has the well-known SPOFs. The reason why we chose this implementation with SPOFs is that it is not practical to introduce a high-cost multiplexing to a small-scale web server which we assume in this paper. In addition, since the attack strength is not heavy for the Control Machine, it is also not practical to multiplex only the Control Machine.

The implementation shown in Section 5 is one of the possible ones to realize our idea, some of which can prevent the Control Machine from being a SPOF. Generally, a system has multiple SPOFs. If we multiplex the more SPOFs, we can achieve the higher availability of the system, although the cost needed for the multiplexing increases. In this paper, we assume the small-scale web server. However, if the additional costs were acceptable, we could give redundancy to the system.

One example of multiplexing is as follows:

1) Add one or more new host machines, each of which is the same as the host machine shown in Section 5.

2) Add a new multiplexed load balancer between the connection to the Internet and the all host machines containing the new ones, distributing the packets to the Control Machines.

The above implementation multiplexes all the SPOFs. Another example is to prepare and multiplex a new host machine only for the Control Machine at a lower cost.

8.3 Attack Scenario

In this paper, we assume the HTTP GET Flood attackers that aim to degrade a service, monitor the targeted server, and conduct adaptive attack rate control. The assumption is determined based on the related works shown in Section 2.1 and Section 2.3. The proposed method makes it appear to the attackers as if the attack is successful in order to impede progress towards more severe or more sophisticated attacks. Moreover, the proposed method isolates the attacks and suppresses the attack strength to protect the legitimate clients from the impact of the attacks. Therefore, the above assumption of the attackers generates the limitation that the proposed method does not defend against unexpected attackers. For example, the proposed method does not assume the disruptive DDoS attacks that launch strong and constant rate attacks. However, it is not practical for a small-scale web server to prevent such attacks on the server side.

Another example is as follows. Suppose that the attackers follow the assumption shown in Section 4.1 except that they prepare two kinds of IP addresses: one for sending the attack packets by the DoS tool and the other for monitoring the server status by a method different from the DoS tool. If the above attackers start to launch the attacks and the monitoring, they notice that the monitored error rate does not increase. Therefore, they can be aware of the defense activity of the proposed method.

However, there is also another possibility that they are unaware of it by the following reason. A load balancer is well-known as a load balancing method and is supplied by the service providers. If a Web Server introduces only a load balancer without our proposed method, each of the clients including the attackers and the normal users is forwarded to one of multiple servers behind the load balancer. The monitoring packets are not always forwarded to the servers which receive the attack packets. Therefore, when the monitored error rate does not change, the attackers may mistake that it is due to just a load balancing even if the proposed method mitigates the attacks.

9. Conclusion

In this paper, we propose a method that 1) allows the server to maintain its service to users relatively unaffected by the attacks, 2) responds optimally to each attacker, and 3) impedes the attacker's ability to detect a defensive response. In our proposed method, the Web Server contains a Control Machine, Normal Machines, and Decoy Machines. The Control Machine performs operations for defense against attacks. The Normal Machines supply normal users with the service, while the Decoy Machines fool the attackers. This arrangement ensures that normal users are barely affected by attacks. The Decoy Machines generate different error rates through CPU resource control, presenting different targeted error rates to the attackers. The CPU resource control essentially places each Decoy Machine into an overload status so the server response is random and tends to decline as the load increases, making it appear to attackers as if the attack is successfully degrading the service without being detected. We implemented and evaluated our proposed method. Since the result changed on every evaluation, we performed the same evaluation 5 times. The evaluation results confirmed that our method controlled the CPU resource of each Decoy Machine to make the error rate of each attacker close to their targeted error rate when there were multiple attackers with different targeted error rates. In addition, it was confirmed that the response time of an attacker was widely distributed.

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

References

[1] Zargar, S.T., Joshi, J. and Tipper, D.: A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks, *IEEE Communications Surveys & Tutorials*, Vol.15, No.4, pp.2046–2069 (Mar. 2013).

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

[3] Mirkovic, J., Dietrich, S., Dittrich, D. and Reiher, P.: Internet Denial of Service attack and defense mechanisms, Prentice Hall (2005).

[4] Xin, S., Chen, X., Tang, H. and Zhu, N.: Research on DoS Atomic Attack Oriented to Attack Resistance Test, *Proc. IEEE Conference on Networking, Sensing and Control*, pp.1747–1752 (Apr. 2008).

[5] @police: DoS/DDoS prevention, available from ⟨http://www.npa.go.jp/cyberpolice/server/rd_env/pdf/DDoS_Inspection.pdf⟩ (accessed 2015-12-03).

[6] Srivatsa, M., Iyengar, A., Yin, J. and Liu, L.: Mitigating Application-level Denial of Service Attacks on Web Servers: A Client-transparent Approach, *ACM Trans. Web*, Vol.2, No.3 (July 2008).

[7] Xie, Y. and Yu, S.: Monitoring the Application-Layer DDoS Attacks for Popular Websites, *IEEE/ACM Trans. Networking*, Vol.17, No.1, pp.15–25 (Feb. 2009).

[8] Gadot, Z., Alon, M., Rozen, L., Atad, M., Shulman, Y. and Shrivastava, V.: *Radware 2013 Global Application & Network Security Report*, Radware Ltd. (2013).

[9] Information-technology Promotion Agency Japan: The report of investigation into the DoS attack prevention, available from ⟨http://www.ipa.go.jp/security/fy22/reports/isec-dos/index.html⟩ (accessed 2015-12-03).

[10] Mirkovic, J. and Reiher, P.: A Taxonomy of DDoS Attack and DDoS Defense Mechanisms, *ACM SIGCOMM Computer Communication Review*, Vol.34, No.2, pp.39–53 (Apr. 2004).

[11] Saleh, M.A. and Manaf, A.A.: Optimal Specifications for a Protective Framework Against HTTP-based DoS and DDoS Attacks, *Proc. Symposium on Biometrics and Security Technologies*, pp.263–267 (Aug. 2014).

[12] Gulati, S. and Dhaliwal, A.: Survey on ROQ attacks, *International Journal of Advanced Research in Computer and Communication Engineering*, Vol.2, No.6 (June 2013).

[13] Palmieri, F., Ficco, M. and Castiglione, A.: Adaptive Stealth Energy-related DoS Attacks Against Cloud Data Centers, *Proc. 8th Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pp.265–272 (July 2014).

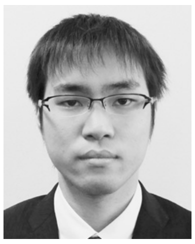
[14] Palmieri, F., Ricciardi, S. and Fiore, U.: Evaluating Network-Based DoS Attacks Under the Energy Consumption Perspective: New Security Issues in the Coming Green ICT Area, *Proc. Conference on Broadband and Wireless Computing, Communication and Applications*, pp.374–379 (Oct. 2011).

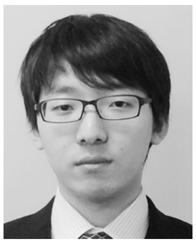
[15] Wu, Z., Xie, M. and Wang, H.: On Energy Security of Server Systems, *IEEE Trans. Dependable and Secure Computing*, Vol.9, No.6, pp.865–876 (Aug. 2012).

[16] Jin, J., Nodir, N., Im, C. and Nam, S.Y.: Mitigating HTTP GET flooding attacks through modified NetFPGA reference router, *1st Asia*

NetFPGA Developers Workshop (June 2010).
[17] Lu, W. and Yu, S.: A HTTP Flooding Detection Method Based on Browser Behavior, *Proc. Conf. IEEE Computational Intelligence and Security*, Vol.2, pp.1151–1154 (Nov. 2006).
[18] Lee, J., Jeong, H., Park, J., Kim, M. and Noh, B.: The Activity Analysis of Malicious HTTP-based Botnets using Degree of Periodic Repeatability, *Proc. Conf. Security Technology*, pp.83–86 (Dec. 2008).
[19] Stevanovic, D., Vlajic, N. and An, A.: Detection of Malicious and Non-malicious Website Visitors Using Unsupervised Neural Network Learning, *Elsevier Applied Soft Computing*, Vol.13, No.1, pp.698–708 (Jan. 2013).
[20] Das, D., Sharma, U. and Bhattacharyya, D.: Detection of HTTP Flooding Attacks in Multiple Scenarios," *Proc. Conf. Communication, Computing & Security*, pp.517–522 (Feb. 2011).
[21] Baik, N., Kang, N., Pak, H. and Sim, W.: Analysis and design of an intrusion tolerance node for application in traffic shaping, *Proc. Conference on Control*, Automation and Systems, pp.857–862 (Oct. 2008).
[22] Chen, Y.W.: Study on the Prevention of SYN Flooding by Using Traffic Policing, *Proc. Symposium on Network Operations and Management*, pp.593–604 (Apr. 2000).
[23] Garg, A. and Reddy, A.: Mitigation of DoS attacks through QoS regulation, *Microprocessors and Microsystems*, Vol.28, No.10, pp.521–530 (Dec. 2004).
[24] Kuzmanovic, A. and Knightly, E.W.: Low-Rate TCP-Targeted Denial of Service Attacks (The Shrew vs. the Mice and Elephants), *Proc. Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pp.75–86 (Aug. 2003).
[25] Takahashi, T., Taguchi, G., Kobayashi, R. and Kato, M.: HTTP-GET Flood provision by dynamic resource control to virtual machine, *IEICE Trans. Inf. Syst.*, Vol.J94-D, No.12, pp.2058–2068 (Dec. 2011) (in Japanese).
[26] Watanabe, M., Kobayashi, R. and Kato, M.: HTTP-GET Flood Prevention Method by Dynamically Controlling Multiple Types of Virtual Machine Resources, *J. Inf. Process.*, Vol.23, No.5, pp.655–663 (Sep. 2015).
[27] Lin, C., Liu, J. and Chen, C.: Access Log Generator for Analyzing Malicious Website Browsing Behaviors, *Proc. 5th Conf. Information Assurance and Security*, pp.126–129 (Aug. 2009).
[28] Sardana, A. and Joshi, R.C.: Autonomous Dynamic Honeypot Routing Mechanism for Mitigating DDoS Attacks in DMZ, *Proc. IEEE International Conference on Networks* pp.1–7 (Dec. 2008).
[29] Creasy, R.J.: The origin of the VM/370 time-sharing system, *IBM J. Research and Development*, Vol.25, No.5, pp.483–490 (Sep. 1981).

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 of Technology. His research interests include computer architecture, parallel processing, and network security.


Genki Otani is currently a Bachelor Course student of Toyohashi University of Technology. His research interests include network security.


Takuro Yoshida received his B.E. degree in Computer Science and Engineering from Toyohashi University of Technology in 2015. He is currently a master's-degree student at the same university. His research interests include network security.

Masahiko Kato received his B.E. and M.E. degrees in Engineering from Toyohashi University of Technology and D.E. degree in Systems and Information Engineering from University of Tsukuba respectively. He is now working for Internet Initiative Japan Inc. He is currently interested in network security.