SENTRY: A Novel Approach for Mitigating Application Layer DDoS Threats

Heng Zhang*, Ahmed Taha*, Ruben Trapero*, Jesus Luna§* and Neeraj Suri*
*TU Darmstadt, Germany §CSA (Europe), United Kingdom
Email:{zhang, ataha, rtrapero, jluna, suri}@deeds.informatik.tu-darmstadt.de, jluna@cloudsecurityalliance.org

***Abstract*—Cloud services are attractive with their advocated technical and economic advantages of transparent resource access, scalability, elasticity and multiple others. However, Cloud services also suffer from multiple infrastructure and application-level threats, with the application-layer distributed denial of service (DDoS) attack being one of the harder ones to mitigate. These attacks typically block the targeted servers by consuming the available resources to result in performance degradation along with the reduced availability of services. While some existing schemes (e.g., intrusion detection/protection) are effective for selective attacks, the evolving application layer DDoS attacks are often able to bypass them.**

We address this problem by proposing and validating a novel and efficient methodology, termed *SENTRY*, that specifically aims to mitigate application-layer DDoS attacks. *SENTRY* utilizes a challenge-response approach that: (a) analyses the attackers physical bandwidth resources, (b) dynamically adapts to the varied work load scenarios, and (c) blocks suspicious service requests from dishonest clients.

I. Introduction

Distributed denial of service (DDoS) attacks constitute a non-trivial security threat for Cloud service providers where the attacks overload the victim server systems to result in degraded services. Most DDoS attacks target the easy-to-attack transport layer (layer 3 of OSI TCP/IP stack) and network layer (layer 4 of OSI TCP/IP stack) of a communications system. The attacks directed at these layers are designed to flood a network interface with attack traffic in order to overwhelm its resources and deny its ability to respond to legitimate traffic.

While network attacks are still a significant challenge due to their scale, the DDoS attacks targeting the application layer may prove to be a more vexing long-term challenge [1]. This challenge arises due to the increasing number and complexity of web applications along with the large network bandwidths of the systems hosting these applications [2] where the attack progressively depletes (versus typical flooding in classical DDoS) the resources from a web or application server. For example, an attack incident occurred at Bitbucket Data Center[1], where the data center was intermittently out of service for over 12 hours [4]. Furthermore, the increasing number of web applications and the shortage of techniques to mitigate DDoS attacks makes them highly attractive targets. Typically, the application layer DDoS attack (a) produces less network traffic than traditional DDoS attack in network channels making their detection hard, (b) causes higher system overhead with the same amount of attacking requests traffic than the traditional DDoS attack in the server side, and (c) displays higher possibility to bypass intrusion and detection systems than the traditional DDoS attack.

In order to address such application layer DDoS attacks, we propose a novel security mitigation scheme called *SENTRY*. *SENTRY* takes advantage of the remote user's local uplink bandwidth to (a) interactively examine the legitimacy of the request in order to dynamically mitigate the resource flooding caused by the application layer DDoS attacks, and (b) to dynamically restrict resource exhaustion effects. Fundamentally, an uplink bandwidth based challenge-response process is imposed on predefined types of service requests. Overall, our schema for mitigation of application layer DDoS attacks makes the following contributions:

- *SENTRY* works at the middleware/protocol level to alleviate the configuration workload caused by dealing with lower-level network details, and allows add-on production line deployment for Cloud service providers.
- *SENTRY* is adaptable to support servers handling varied workload scenarios.
- *SENTRY* aims to defeat the potential dishonest attempts by launching a physical bandwidth based challenge-response process to thwart "smart" adversaries intending to cheat. Consequently, it blocks suspicious service requests from dishonest clients.

Our evaluation shows that the *SENTRY* can effectively mitigate application layer DDoS attacks in practice as demonstrated with four different use cases.

This paper is organized as follows. Section II presents the basic characteristics of application layer DDoS attacks. Section III details the attacker and victim models on which we quantify the performance impact of these attacks. In Section IV, we detail the design of the attack mitigation scheme. Section V presents the experimental evaluation to validate the effectiveness of *SENTRY*. In Section VI, we overview related works on mitigating application layer DDoS attacks.

II. Background

The application layer DDoS attack is a sophisticated DDoS attack that stealthily depletes the available resources on victim servers. Compared to the traditional networking layer oriented DDoS attacks, the application layer DDoS attacks present three main characteristics as follows.

[1] A mainstream code-hosting software-as-a-service (SaaS) provider [3].

Firstly, the application layer DDoS attack is a workload-enhancing attack that manifests the denial of service via resource-starving performance degradation where the resources commonly consist of CPU cycles, I/O, physical memory and network bandwidth. Although Cloud server systems possess massive system resources, a specific type of resource could still become the bottleneck of the overall system performance in some cases. For instance, while the Amazon Cloud service has huge network traffic handling capability, an XML and HTTP protocol based application layer DDoS attack targeting Amazon EC2 resources [5] resulted in a complete saturation of the EC2 resources.

Secondly, the application layer DDoS attack is an asymmetric DDoS attack [6]. The application layer DDoS attack targets very specific application protocols, which entail characteristically high overhead services. The attacker sends a few but selective high overhead service requests to target servers from multiple exploited client hosts resulting in excessive system overhead for the target servers to process them. As a result, the application layer DDoS attack can keep deteriorating the system performance until the target servers are completely out of service.

Thirdly, the application layer DDoS attack is a stealthy type of attack that initiates "normal" service requests that then bypass the "anomalous" behavior focused intrusion detection systems. For example, the authentication service is a necessary application service in many Cloud service systems. But it is vulnerable to the masquerading signature attack that consumes considerable system resources to run the verification process [7]. By distributing the masquerading service requests across multiple attacking sources, the application layer DDoS attack produces "minor" traffic changes that elude the (high) traffic analysis based intrusion detection systems.

Given such characteristics, the need is to develop a mitigation solution that can block attacking service requests from dishonest clients. To achieve this purpose, our proposed scheme designs a resource based challenge-response scheme for mitigating application layer DDoS attacks. The proposed scheme interactively challenges and validates the service requests from the remote clients in order to block the suspicious attacking requests.

III. MODELS

In this section, we (a) describe the attacker model used for performing the application layer DDoS attacks and (b) present the victim model on which we measure the performance impact of these attacks.

A. Attack Model

The goal of the attacker is to overwhelm one or more server resources so that the legitimate clients suffer from high service latency and low throughput. This goal can be made by decreasing the quality of the service provided to their clients. Hence, the first step needed for designing an effective mitigation approach is to characterize the potential behavior of the attackers. To this end, one of the possible methods is to identify the high overhead operations associated with the victim services as depicted in Figure 2 which shows the system workload state in different cases as follows:

Mode A. This is the normal operational case (without attacks) where a server becomes overloaded while processing a high amount of different user service requests within a short period. These heterogeneous service requests swarm into the server continuously as shown in *Mode A* in Figure 1. Different types of service requests present different appearance ratios in the requests flow and cause different processing overhead in the server. The salient observation being that a high-rate of high-overhead services can result in an overload.

Mode B. This is the application layer DDoS attack case where the attackers take advantage of selective high-overhead service requests. The attackers manipulate the targeted client hosts to send high-overhead types of service requests. Although the aggregated number of service requests is not necessarily large, a victim server is overwhelmed, for a specific resource, for processing all the incoming service requests. This *Mode B* represents the workload caused by application layer DDoS attacks in Figure 1. Note that the high-overhead types of service requests characteristically appear very often in application layer DDoS attack cases than in normal workload cases.

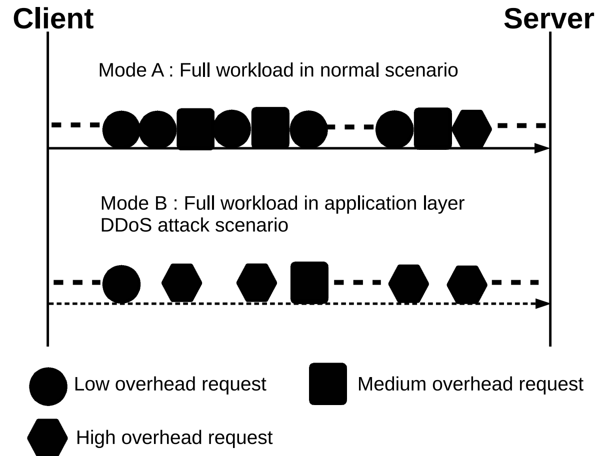


Fig. 1: High system workload situation comparison: normal case vs application layer DDoS attack case

B. Victim Model

Cloud server systems are designed to simultaneously service high volumes of clients requesting varied services. Our victim model focuses on those services which are vulnerable to application layer DDoS attacks. In this paper, we use the example proposed in [6] as our victim model, which presents the different system overhead caused by processing different

types of service requests in an online server system. It can be used to categorize service requests into different classes according to different levels of processing overhead. Based on this example, we consider an online bookstore hosted on multi-tiered architecture as an example of e-commerce application. Figure 2 shows the variation in processing times for different service requests in a bookstore application [6].

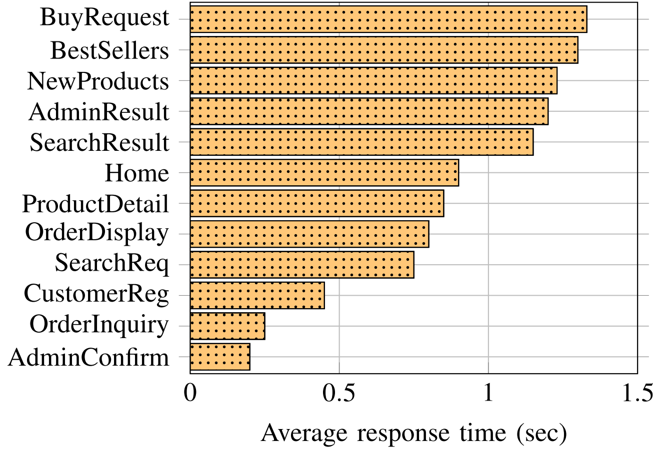


Fig. 2: Processing times for different synamic contetns requests in online bookstore application [6]

As shown in Figure 2, the "BestSellers" service request causes remarkably higher processing overhead than the "AdminConfirm" service request. The reason for such difference is the different amount of system resources needed to perform these requests. For example, the "BestSellers" request involves high resource demanding operations such as inquiring the database, sorting related results and returning the final result to the user.

In order to facilitate subsequent discussions in this paper, we make the following assumptions:

I) We assume that the Cloud service provider can conduct surveillance on processing the incoming service requests at the server side. This assumption has been put in practice by some Cloud service providers. For example, Amazon offers a monitoring product called "Amazon CloudWatch" [8] that can check the AWS resources situation in an approximate real-time mode.

II) We assume that attackers have full control of the exploited hosts including manipulating the local system resources of the hosts.

IV. Proposed Mitigation Scheme

In this section, we propose a resource based challenge-response scheme for mitigating application layer DDoS attacks. Our mitigation scheme (a) actively challenges the request senders validity, and (b) filters out suspicious requests by verifying the responses from the senders. In order to launch an application layer DDoS attack, attacking participants have to send a large number of attacking requests to overwhelm a target server with enough attack strength which refers to the aggregated sending rate of attack requests to a target server per second. Therefore, we assume that attacking participants will make full use of the local bandwidth resources by sending high overhead service requests more frequently than normal users whose service requests present a uniform arrival rate [9]. In consequence, such high overhead service requests result in excessive system resources consumption. Thus it is a critical task for a security mitigation solution to minimize the attack strength for reducing the system overhead. Hence it is necessary to identify and discard the high overhead attacking requests from the service request flow by examining the request responses to specially generated challenge messages. We explain the proposed mitigation scheme by first describing the system overview and then detailing the moderator component.

A. System Overview

Our system consists of a Cloud client (or remote client), a Cloud server and a novel mitigation component called "Moderator" as depicted in Figure 3. The Moderator is placed at the server side and is responsible for conducting challenge-response processes against incoming service requests in order to mitigate application layer DDoS attacks.

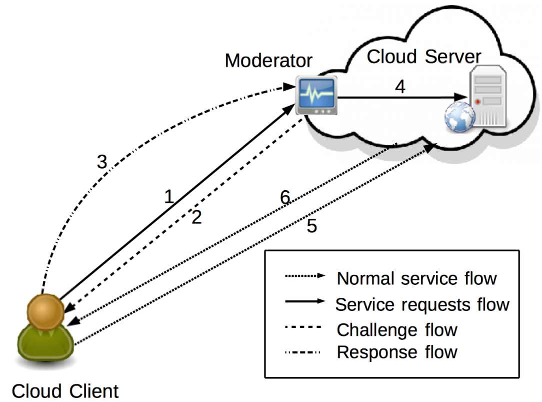


Fig. 3: System overview

The mitigation scheme, as depicted in Figure 3, comprises the following steps:

Step 1. A remote service client sends a high overhead service request flow to the server system

Step 2. The moderator component samples the incoming service requests. Once the high overhead service request is sampled, a challenge message is issued and sent to the client.

Step 3. The client responds to the challenge message with local bandwidth resources (This is completely explained in Section IV-B2).

Step 4. The client's response received by the moderator is verified to check whether it is valid or not. The moderator will drop that sampled service request if it is invalid. Otherwise, it will forward this service request to the server.

Step 5 & 6. The requests that are not sampled by the moderator are served normally by the Cloud server.

Typically, challenges are data structures obtained by considering parameters from the client, such as CPU cycles, memory

or bandwidth resources. In our work, we have chosen the client side physical uplink bandwidth as the base for designing the challenges used in *SENTRY*. The reason for choosing this are:

I. Most network applications offer their services to remote clients using the downlink bandwidth resources [10] (except for few network applications as peer to peer transmission). This means that using the uplink bandwidth for the proposed mitigation scheme resource causes a limited performance influence on these services.

II. The client bandwidth is strictly managed by his/her local Internet service provider (ISP) and cannot be modified by DDoS attackers. Therefore, client bandwidth resource becomes a good base to design the challenges due to its strong speculation-proof property.

B. Moderator Description

The moderator component manages the challenge and response processes. It works as an intermediate component for challenging the selected high overhead service requests and verifying the corresponding responses. Each challenge message encloses the expected size of the response to be sent back by the senders. At the same time, it is independent from the server which facilitates the deployment.

The moderator workflow is depicted in Figure 4. It consists of several internal modules, namely Probing Module, Challenge Module, Receiving Module, Relay Module and Failure Handling Module.

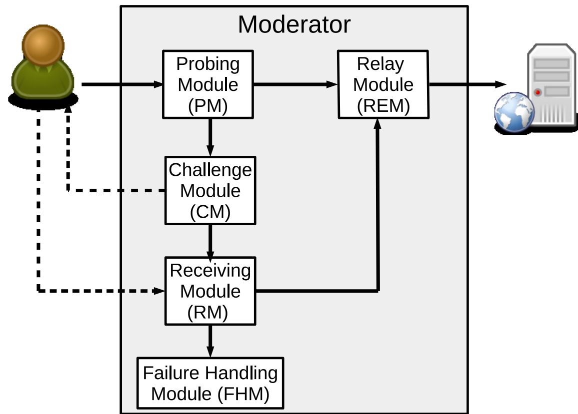


Fig. 4: Internal design and process diagram of moderator

1) Probing Module: The probing module (PM) is responsible for sampling the incoming service requests from clients. It works as a flexible sampler with different possible configurations that can be used to adapt the sampling rate. These configurations are adjusted (by the server administrator) by modifying the following two parameters:

- **Sampling target** (S_{Target}): S_{Target} specifies the targeted type of service requests sampled by the PM. In this paper, we set the high overhead type service requests to be the S_{Target}, as these are the type of requests used in our attacker model (cf., Section III-A). For example, we assume that the service requests of type "BestSellers" specified in the victim model (cf., Section III-B) are the sampling targets from all the other request types specified in the victim model. This means, only "BestSellers" requests are sampled from all types of service requests received by the server.
  For example, in Figure 5 the session of Client N contains all service requests with different overhead types. The light dark blocks refer to those low overhead service requests submitted by Client N. The medium dark blocks refer to medium overhead service requests from Client N and the deep dark blocks refer to the high overhead ones. The specific configuration of S_{Target} depends on how much system resources will be allocated for the moderator component. The more system resources are available for the moderator, the more types of service request can be added into S_{Target}.
- **Sampling Probability** (S_{Prob}): S_{Prob} specifies the percentage of sampled requests (S_{Prob}). For example, if S_{Prob} is 20% then one out of five target service requests is sampled for subsequent challenge-response process. Thus, according to the previous example 20% of "BestSellers" type requests are sampled and sent to the next module.

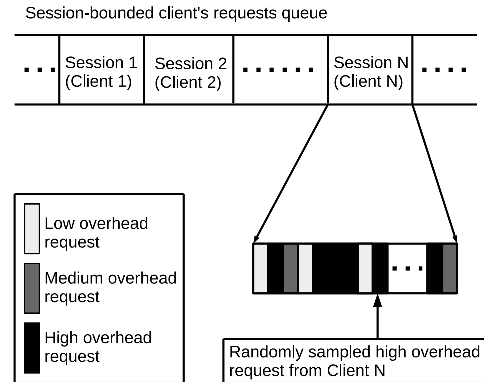


Fig. 5: User session based random service request sampling diagram

Once these parameters are configured, the PM is ready to execute the sampling task for the moderator. At the end of this process, the successful sampled target service request (denoted as *Req*) is sent to the Challenge Module as shown in Figure 4.

2) Challenge Module: The challenge module (CM) issues challenge messages for every sampled request received from the PM. The challenge message is a standard HTTP/1.1 response message with the challenge information embedded in the message body. As there are different type of service requests, we introduce a weighted challenge algorithm (based on the algorithms specified in [11]) to classify the sampled service request *Req* into three different main groups according to their type. Namely, Group G_{low} contains requests with low overhead. Group G_{medium} contains requests with medium overhead. Group G_{high} contains requests with high overhead.

In this challenge message, the CM asks the client for a specific amount of binary data (specified in the challenge message as shown in Figure 6). The client sends a response message containing binary data with the specified size as shown in Figure 6. The weighted challenge algorithm generates the challenge size (CZ) according to the type of Req such that:

$$CZ = \begin{cases} \delta, & \text{if } Req \in G_{low} \\ (\alpha+1)\,\delta, & \text{if } Req \in G_{medium} \\ \delta^{\beta} + (\alpha+1)\,\delta, & \text{if } Req \in G_{high} \end{cases} \quad (1)$$

where α, β and δ are positive integers that can be configured to customize the size of the challenge depending on the Req group (low, medium or high). More details about the algorithm complexity analysis are given in [11]. In order to thwart potential guessing attempts from advanced attackers, all these three variables' values are randomly generated within a set of specified ranges. Once CZ is calculated and generated, it is added to the challenge message as depicted in Figure 6.

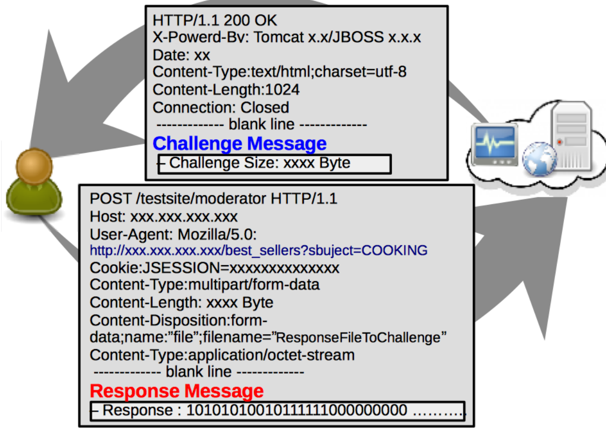


Fig. 6: Challenge and response messages

3) Receiving Module: The receiving module (RM) receives and validates the response of the challenge from the remote client. Its main responsibility is, firstly, to check the actual size of the challenge response. Then the RM verifies whether the received response size matches with the challenge response size specified by CM. The RM also uses the client's Session ID (denoted by SID_{Req}) to identify the sender of Req and forward or discard the client's service request Req. More specifically, the RM receives the HTTP request message with a POST method, which is the response to the challenge message issued by the CM. This response message contains the required data submitted by the remote client. The RM retrieves the client's session information SID_{Req} from the message header and checks the size of the binary data in the message body. Then, it compares the retrieved response information from the clients and the issued challenge information from the CM. If the response matches the issued challenge message, Req is assumed to be sent from a honest client for correctly making an uplink bandwidth response to specified challenge size. Therefore, RM will route this request Req together with its session information SID_{Req} to the Relay Module (REM) for further process. On the other side, if the response mismatches the issued challenge message, Req is marked as a suspicious attacking request. As a result, the RM sends this request Req and its session information SID_{Req} to the Failure Handling Module (FHM) as depicted in Figure 4.

It is worth highlighting that not all the failures in this challenge-response mitigation process are from attackers. Some legal clients might also occasionally suffer from some transient connection congestion or hardware failures. In this case, it is expected that the clients will resubmit their service requests and make correct responses when challenged again by the moderator. However, attackers can either not make correct responses to the challenge messages or only a limited number of attacking service requests can be processed by server systems if they are "smart enough" to mimic all behaviors of a normal client. For the former case, all attacking requests are filtered out thoroughly. For the latter case, the attacking strength is significantly minimized as only a limited amount of attacking requests get processed at the server and most of the attacking requests are blocked by unsuccessful uplink bandwidth resource responses.

4) Relay Module: The relay module (REM) acts as moderator's output interface and its main responsibility is to forward the sampled service request Req to the server system. Obviously, any non-sampled service requests are directly relayed to server systems by REM as shown in Figure 4.

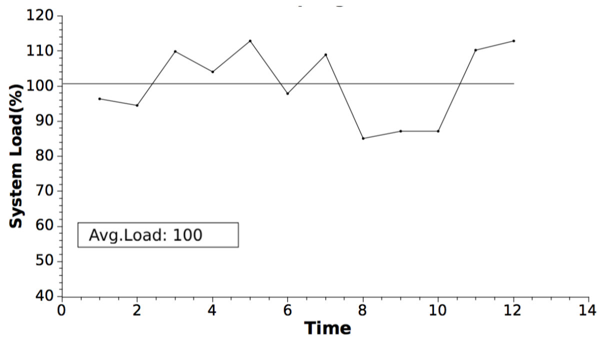
5) Failure Handling Module: The failure handling module (FHM) is an optional module in our design. Once a sampled service request Req failed to make a correct response to corresponding challenge message, it will be handled by the FHM. The FHM is responsible to execute some post-challenge processes, which comprise intrusive IP banning, request redirecting, user information logging and so on.

V. Evaluation & Discussion

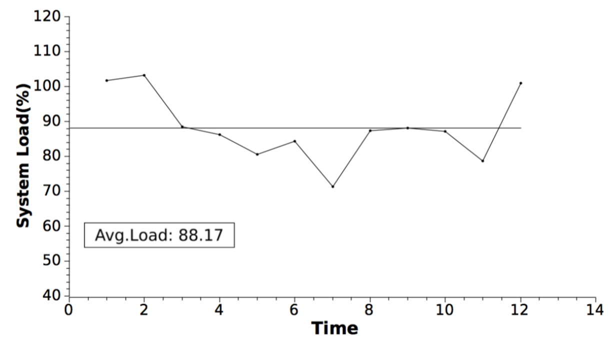
In this section, we evaluate the performance of the moderator component for several configurations. We discuss the results corresponding to each setting, and subsequently outline the comparisons with contemporary works to illustrate the advantages of our design.

A. Experiment

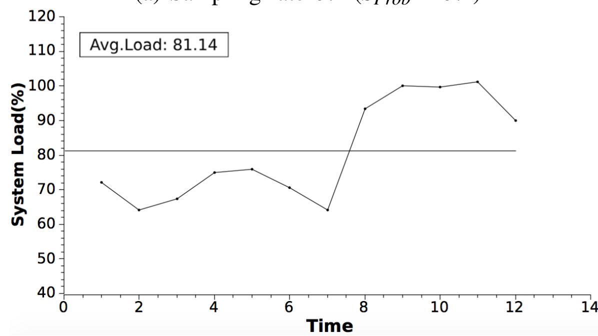
SENTRY consists of three elements as specified in Section IV-A (cf., Figure 3): a web server, a moderator and a Cloud client. The web server is implemented using a Jboss application server and Mysql to offer the database services. We used the victim model shown in Section III-B as our web server. The moderator consists of a set of developed JSP files deployed on the Jboss application server. The Cloud clients are modeled as emulated browsers which are used to emulate human clients' operations by sending different type of service requests to the web server. Emulated browsers send different service requests to the online bookstore application scenario



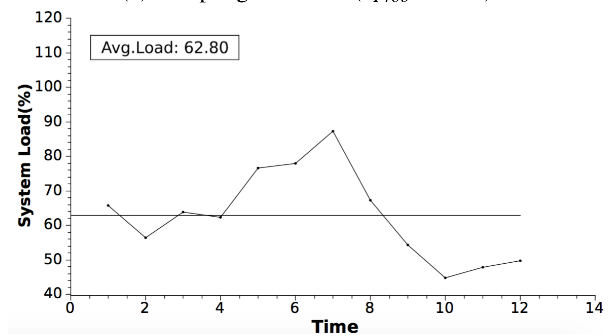
(a) Sampling rate 0% ($S_{Prob} = 0\%$)



(b) Sampling rate 33% ($S_{Prob} = 33\%$)



(c) Sampling rate 66% ($S_{Prob} = 66\%$)



(d) Sampling rate 80% ($S_{Prob} = 80\%$)

Fig. 7: System overhead graph with different sampling rates

such as searching books, inquiring Best Sellers, registering new accounts, confirming orders and so on. Furthermore, we deploy a group of emulated browsers to act as attack participants in order to frequently submit high overhead attack requests.

In our experiment, we deployed more than 600 concurrent emulated browsers, where 100 of them were specially configured as attacking participants to deliberately submit attacking service requests specified by S_{Target}. These attacking service requests took up to 25% amount of the overall service requests. We configured the sampling parameter S_{Target} to be modeled as "BestSellers" type request for referring to high overhead service request (cf., Section III-B), and "Search" type request for medium overhead service request as specified in Equation 1. In addition, we also configured S_{Prob} with different sampling rates (0%, 33%, 66%, and 80%) to investigate the behavior of the moderator in different situations. Thus, we measured the system workload at each of the defined four sampling rates. To perform this, we collected 600 seconds of system workload data after the web server entered the stable status. The workload graph of our tests is presented in Figure 7. It consists of four sub-graphs for plotting each test. In these sub-graphs, the X-axis is the testing time (in seconds) and each time unit represents 50 seconds. The Y-axis shows the server workload (in percentage).

1) Attack Scenario with $S_{Prob} = 0$: The first experiment presents the server working at full load with the moderator deactivated. In this case, the overall emulated browsers are connected to the web server to request various kinds of services simultaneously. As shown in Figure 7a , the web server's average work load is full. The server received up to 79097 service requests from the emulated browsers. Therefore, the online bookstore server is overloaded and the service is likely suffering a denial of service attack.

2) Attack Scenario with $S_{Prob} = 33\%$: In the second experiment, the moderator is activated and the sampling rate is configured at 33%, which means $1/3$ of submitted search and bestsellers requests are sent to the moderator. As shown in Figure 7b, web server's average work load is reduced to 88.17%. The server system received 78787 service requests from emulated browsers, while 5599 service requests failed to get served due to incorrect responses to challenge messages.

3) Attack Scenario with $S_{Prob} = 66\%$: In the third experiment, the moderator is activated and the sampling rate is configured with 66%, which means that $2/3$ of the submitted search and bestsellers requests are sent to the moderator. As shown in Figure 7c, web server's average work load decreased to 81.14%. The server system received 79157 service requests from emulated browsers. While, 10518 service requests failed to get served due to their incorrect responses to challenge messages.

4) Attack Scenario with $S_{Prob} = 80\%$: In the fourth experiment, the moderator is activated and the sampling rate is configured with 80%. As shown in Figure 7d, web server's average workload further decreased to 62.80%. The server system received 79281 service requests from emulated browsers. While, 12310 service requests failed to get served due to their incorrect responses to challenge messages.

TABLE I: Experimental Results Assessment Table

Data	**Moderator Sampling Rate**			
	0	33%	66%	80%
Mean System Load	100%	88.17%	81.14%	62.80%
Blocking Rate	0	7.11%	13.29%	15.53%
False Negative Rate	N.A	13.86%	19.47%	22.36%

B. Discussion

From these experiments, three specific type of data have been collected and presented in Table I, namely mean system load, blocking rate and false negative rate.

The mean system load reflects the performance of the moderator component and the effectiveness of *SENTRY*. As the first row in Table I shows a monotonic system load decrease from 100% (when moderator is deactivated, $S_{Prob} = 0$) to 88.17% (with $S_{Prob} = 33\%$) and further dropped to 81.14% when the sampling rate raised to 66%. The system load decreases to 62.80%, when S_{Prob} is 80%. From the above data, the application layer DDoS attack threat is successfully mitigated.

The reason for the system load drop is that attack participants are controlled by attacking scripts. Those scripts are not able to decode moderator's challenge messages and make correct bandwidth responses accordingly. In case any attacker is "smart enough" to mimic human behavior, the limited client side bandwidth resource imposes rigid restriction on attacking scripts, which can not send too many attacking requests to achieve the service denial purpose.

The blocking rate refers to the percentage of blocked attacking requests in the entire service request flow when the moderator is activated. From the second row in Table I, the blocking rate raises from 7.11% (with 33% sampling rate) to 13.29% (with 66% sampling rate) and then to 15.53% (with 80% sampling rate). Considering that 25% of the service flows are attacking requests, these blocking rates can still be maintained at high level. For example, it can still block up to 77.64% attacking requests in the overall attacking service flow even when the sampling rate is equal to 80%.

The false negative rate shows the mitigation performance of the moderator component. It refers to the rate of those attacking requests which *SENTRY* failed to block. As in the third row in Table I, the false negative rate increases gradually as long as we increase the sampling rate. It shows a false negative rate of 13.86% with 33% sampling rate and 19.47% with 66% sampling rate and 22.36% with 80% sampling rate. As a result, the higher the sampling rate is, the higher the false negative rate is.

VI. Related Work

This section presents a survey on the state of art regarding to addressing application layer DDoS attacks.

Many researchers have attempted to mitigate the security threat of application layer DDoS attacks using a variety of techniques. For example, Stavrou et al. [12] proposed a heterogeneous countermeasure against DoS attacks based on a graphical Turing test which is an enhanced version of the classical CAPTCHA method [13]. Since the classical CAPTCHA method was breached by the work from Mori and Malik [14]. This graphical Turing test method consumed a considerable amount of server resources to generate graphical CAPTCHAs for remote users.

Yen and Lee [15] introduced a statistical technique to mitigate the threat of random querying based application layer DDoS attacks. A potential problem of their technique is that attacking sources are assumed to generate service requests in Round-robin mode, which mismatches with the real attacking scenarios. While Xie et al. [16] presented an improved semi-Markov model to mitigate application layer DDoS attack threats by profiling the dynamic access behaviors of aggregated proxy traffic from remote clients. However, the semi-Markov algorithm complexity depends on its parameters which are very challenging to choose.

Seufert and O'Brien [17] presented a machine learning based defence mechanism to mitigate the DDoS threat. They collected data from the OSI TCP/IP layer 3, layer 4 and layer 7 from all incoming user requests and employed an artificial neural networks (ANN) to categorize the user types for the isolation purpose. Unfortunately, it is an expensive computational task to run the ANN algorithm especially when the traffic is at a very large scale. In contrast, Yu et al. [18] developed a light weight application layer DDoS attack mitigation solution by leveraging the trust property to differentiate attackers from the normal user group. The trust is derived from a user's visiting history and encrypted for the storage. While it can be exploited by attackers to trigger a significant amount of workload for computing the trust.

Khor and Nakao [19] designed a self-verifying Proof-of-Work (sPoW) mitigation solution against application layer DDoS attack. The sPoW mechanism grants normal users to access different services by solving different difficulty-level puzzles. However, it is incapable of defeating high intelligent adversaries who can pass the puzzle tests.

Furthermore, Wang et al. [20] designed a graphical inference model based mitigation solution in Software Defined Network(SDN) which is an emerging network architecture decoupling data plane and control plane in traditional network architectures. However, it is an experimental solution and not feasible to deploy in contemporary network architectures.

Few works addressed the DDoS problem by applying resource-based schemes. The resources are rigidly constrained by physical capacity limitations and difficult to be speculated by malicious attackers. For instance, Abadi et al. [21] proposed a memory resource based scheme to defeat adversarial clients by equipping a memory-bounded hard functions. While Walfish et al. [22] proposed a bandwidth resource based scheme to make normal clients get served with more bandwidth resource than abnormal ones by requesting all connected clients to join a bandwidth resource auction. Moreover, Khanna et al. [23] proposed another bandwidth resource based scheme on the shared communication channels. However, it is a relatively heavy-weight solution for deployments in practice.

This review highlights that existing approaches encounter

TABLE II: Comparison Table of Application Layer DDoS Attack Mitigation Schemes

Mitigation Approach	Characteristics	Comments
Turing test based mitigation scheme [12]	Graphic Turing Tests	High service latency High execution overhead
Statistic based mitigation scheme [15]	Statistics based Statistical model dependent	High false negative rate
Trust based mitigation scheme [18]	Trust analysis based Analyzing browsing behaviors	False negative rate depending on attacking profile Huge amount of log files required Vulnerable to human behavior mimic attacks
Machine learning based mitigation scheme [17]	Machine learning based Sample collection Feature extraction algorithm needs training	High execution overhead
Software Defined Network (SDN) based mitigation scheme [20]	SDN based Control network flow with separate planes	Communication overhead depends on SDN structure
Hidden semi-Markov model based mitigation scheme [16]	Statistic processes based High mitigation rate	Difficulty in model parameter selection
Resource based mitigation scheme [22]	Bandwidth resource based Legal users get higher service possibility	Client status information required to maintain at the server side

validity of assumptions or high cost of implementation as briefly summarized in Table II. Since all these contemporary security solutions do not satisfactorily or efficiently mitigate the application layer DDoS attack threat, it forms the premise for *SENTRY*.

VII. Conclusion

In this paper, we investigated the threat of application layer DDoS attacks against server systems. In order to mitigate this threat, we propose an uplink bandwidth based challenge-response mitigation scheme (called *SENTRY*) with flexible mitigating capability and strong speculation-proof property. In addition to evaluate the performance of the proposed mitigation scheme, we implemented *SENTRY* in a software component called "Moderator" that is deployed at the server side. The experimental results collected from the evaluation process demonstrate the effectiveness of the proposed challenge-response mitigation mechanism. Therefore, the proposed mitigation scheme can assist servers to thwart the application layer DDoS attacks by reducing unnecessary system overhead which caused by processing the attacking service requests. In future work, we will focus on addressing the tunability of the moderator's sampling parameters to improve performance.

VIII. Acknowledgment

Research supported by Hessen LOEWE TUD CASED.

References

[1] Akamai Technologies, "Akamai state of the internet security report," 2015, https://www.akamai.com/us/en/multimedia/documents/report/q4-2015-state-of-the-internet-security-report.pdf.

[2] S. Ranjan, K. Karrer, and E. Knightly, "Wide area redirection of dynamic content by internet data centers," *Proc. of INFOCOM*, pp. 816–826, 2004.

[3] Atlassian, "Bitbucket Data Center," https://bitbucket.org.

[4] Glenn Butcher, "Atlassian subject to Denial Of Service attack," 2011, http://blogs.atlassian.com/2011/06/atlassian_subject_to_denial_of_service_attack.

[5] S. VivinSandar and S. Shenai, "Economic denial of sustainability (edos) in cloud services using http and xml based ddos attacks," *International Journal of Computer Applications*, vol. 41, no. 20, pp. 11–16, 2012.

[6] S. Ranjan, R. Swaminathan, M. Uysal, and E. Knightly, "Ddos-resilient scheduling to counter application layer attacks under imperfect detection," *Proc. of INFOCOM*, pp. 1–13, 2006.

[7] J. Mirkovic and P. Reiher, "A taxonomy of ddos attack and ddos defense mechanisms," *In SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.

[8] Amazon Inc, "Amazon CloudWatch," 2015, https://aws.amazon.com/cloudwatch/details/?nc2=h_ls.

[9] Y. Xie and S. Yu, "A large-scale hidden semi-markov model for anomaly detection on user browsing behaviors," *In Transactions on Networking*, vol. 17, no. 1, pp. 54–65, 2009.

[10] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot, "Packet-level traffic measurements from the sprint ip backbone," *In IEEE Network*, vol. 17, no. 6, pp. 6–16, 2003.

[11] R. Sedgewick and K. Wayne, *In Algorithms*. Pearson Education, 2011.

[12] A. Stavrou, J. Ioannidis, A. Keromytis, V. Misra, and D. Rubenstein, "A pay-per-use dos protection mechanism for the web," *Proc. of Applied Cryptography and Network Security*, pp. 120–134, 2004.

[13] L. Von, M. Blum, N. Hopper, and J. Langford, "Captcha: Using hard ai problems for security," *Proc. of EUROCRYPT-Advances in Cryptology*, pp. 294–311, 2003.

[14] G. Mori and J. Malik, "Recognizing objects in adversarial clutter: Breaking a visual captcha," *Proc. of Computer Society Conference on Computer Vision and Pattern Recognition*, pp. I–134, 2003.

[15] W. Yen and M. Lee, "Defending application ddos with constraint random request attacks," *Proc. of Asia-Pacific Conference on Communications,*, pp. 620–624, 2005.

[16] Y. Xie, S. Tang, X. Huang, C. Tang, and X. Liu, "Detecting latent attack behavior from aggregated web traffic," *In Computer Communications*, vol. 36, no. 8, pp. 895–907, 2013.

[17] S. Seufert and D. O'Brien, "Machine learning for automatic defence against distributed denial of service attacks," *Proc. of International Conference on Communications*, pp. 1217–1222, 2007.

[18] J. Yu, C. Fang, L. Lu, and Z. Li, "A lightweight mechanism to mitigate application layer ddos attacks," *Proc. of Scalable Information Systems*, pp. 175–191, 2009.

[19] S. Khor and A. Nakao, "Daas: Ddos mitigation-as-a-service," in *Proc. of Applications and the Internet*, 2011, pp. 160–171.

[20] B. Wang, Y. Zheng, W. Lou, and Y. Hou, "Ddos attack protection in the era of cloud computing and software-defined networking," *In Computer Networks*, vol. 81, pp. 308–319, 2015.

[21] M. Abadi, M. Burrows, M. Manasse, and T. Wobber, "Moderately hard, memory-bound functions," *In Transactions on Internet Technology*, vol. 5, no. 2, pp. 299–327, 2005.

[22] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker, "Ddos defense by offense," *In SIGCOMM Computer Communication Review*, vol. 36, no. 4, pp. 303–314, 2006.

[23] S. Khanna, S. Venkatesh, O. Fatemieh, F. Khan, and C. Gunter, "Adaptive selective verification: An efficient adaptive countermeasure to thwart dos attacks," *In Transactions on Networking*, vol. 20, no. 3, pp. 715–728, 2012.