



Measurement-Based Analysis of a DoS Defense Module for an Open Source Web Server

Marta Catillo, Antonio Pecchia, Umberto Villano

► To cite this version:

Marta Catillo, Antonio Pecchia, Umberto Villano. Measurement-Based Analysis of a DoS Defense Module for an Open Source Web Server. 32th IFIP International Conference on Testing Software and Systems (ICTSS), Dec 2020, Naples, Italy. pp.121-134, 10.1007/978-3-030-64881-7_8 . hal-03239811

HAL Id: hal-03239811

<https://inria.hal.science/hal-03239811>

Submitted on 27 May 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

Measurement-Based Analysis of a DoS Defense Module for an Open Source Web Server

Marta Catillo, Antonio Pecchia, and Umberto Villano

Dipartimento di Ingegneria
Università degli Studi del Sannio, Benevento, Italy
{marta.catillo,antonio.pecchia,villano}@unisannio.it

Abstract. Denial of Service (DoS) attacks represent an ever evolving landscape, which ranges from bruteforce flooding approaches to more sophisticated low-bandwidth slow techniques. DoS has become a major threat to the availability of modern web servers because of the large number of attack tools across the Internet. In spite of the increasing number of security modules that can be usefully deployed in production servers, there is not a one-fits-all defense solution against DoS.

This paper proposes a measurement-based analysis of a well-established defense module for the Apache web server. The module is tested against both flooding and slow DoS attacks in order to quantify its capability at assuring correct service to legitimate clients. Results indicate that the module can mitigate flooding DoS attacks while causing some performance loss of the server; however, it is ineffective against slow attacks. The findings of our analysis are useful to support the deployment of proper defense mechanisms.

Keywords: Denial of Service · web server · defense · availability.

1 Introduction

Nowadays, web servers are broadly used for the implementation of Internet services. As a consequence, performance assurance of web servers plays a key role in satisfying the needs of a large and growing community of users. The number of **Denial of Service (DoS) attacks** against web servers has greatly increased in recent years. DoS attacks undermine the availability of a *victim* server by making access difficult, it not even impossible, to legitimate users. DoS is an important focus of interest for current research [5], because of the large number of attack tools across the Internet, frequency and disruptive effects.

Numerous surveys and research papers, such as [19] and [30], deal with DoS both from the perspective of the attack and possible defense. Several schemes [13], [1], [10], have been proposed to detect “traditional” **flooding DoS attacks**, i.e., attacks designed to overwhelm a targeted server by means of a massive number of HTTP requests. However, in the last years, DoS attacks evolved into a “second generation”, i.e., the so-called **slow DoS attacks** [25]. These use low-bandwidth approaches that exploit application-layer vulnerabilities and the

intrinsic design of the HTTP protocol. Researchers have invested considerable effort to detect slow DoS attacks, and a wide body of literature, e.g., [27], [6], [16], is currently on this topic.

In order to mitigate DoS attacks, practitioners can rely on a variety of security modules that are recommended for hardening “core” web server installations. Nevertheless, in spite of the increasing number of security modules that can be usefully deployed in production servers, there is not a “one-fits-all” defense solution against DoS attacks.

In this paper we propose a measurement-based analysis of a well-established **defense module** for the Apache web server called `mod_evasive`¹. It is an Apache module intended for DoS, DDoS (*distributed* DoS) and brute-force attacks mitigation. Our analysis is based on direct performance measurements of a victim server during a variety of DoS attacks performed in a controlled testbed. In order to emulate both flooding and slow DoS attacks, we considered three types of DoS: a classic volumetric DoS made by means of a Python script widely used in the literature, a *home-made* flood attack with a very high request rate, and a slow header DoS attack crafted with a specialized tool. Each attack is launched both against the *baseline* server, i.e., no defense module in place, and its hardened version by enabling `mod_evasive`. We collect different service metrics, such as throughput loss, number of successful requests and reply time, in case of *no* and *with defense* in order to quantify the capability of the module at assuring correct service to legitimate users.

To the best of our knowledge, we are not aware of a similar study that assesses a defense module in face of such a mixture of attack conditions and metrics. In fact, while tech blogs and references provide practical guidance for installation and functional testing the modules, there is a lack of comprehensive performance measurements to drive practitioners’ choice. Our analysis provides several interesting outcomes regarding the module in hand. First, the module mitigates flooding DoS attacks; however, it may cause up to 15.74% throughput loss of the server. Second, *in none of the experiments* the defense module was able to ensure the success of all the legitimate HTTP requests, which means that a server will fail some requests anyway in spite of the defense. Finally, the module is ineffective against slow attacks. Overall, the findings of our study provide a better understanding of the capability of the module and its potential limitations in a production environment. Moreover, we recommend to accompany the module assessed in this study by means of supplemental defense mechanisms^{2,3} in order to achieve comprehensive defense.

The paper is organized as follows. Section 2 discusses related work in the area of DoS and defense. Section 3 describes the experimental testbed, attack tools and service metrics. Section 4 provides a detailed analysis of the attacks and their impact in case of *no* and *with defense*. Section 5 concludes the work and discusses potential future directions.

¹ https://github.com/jzdziarski/mod_evasive

² <https://sourceforge.net/projects/mod-antiloris/>

³ https://httpd.apache.org/docs/trunk/mod/mod_reqtimeout.html

2 Related Work

The escalation of DoS attacks has attracted a significant interest by the research community, which focused on specialized DoS attacks detection mechanisms. A number of countermeasures have been taken over the years to mitigate DoS attacks [29]. Some conventional approaches rely on monitoring the connection request rate [14]: for example, a client whose connection request rate is higher than a pre-established threshold is marked as an attacker. This technique is mostly ineffective for slow DoS attacks, as shown in [2]; moreover, in some cases even a legitimate requesting user could have a short-term burst of connection requests without leading to an attack [24]. Authors in [7] analyze DoS traffic from a public research dataset and measure its impact under different configurations of a victim server. Results indicate that a proper configuration of the server can strongly mitigate the impact of a DoS attack.

In recent years, with the rapid diffusion of deep learning techniques, many machine learning-based DoS detectors have spread in the literature [18], along with more classic detection techniques. In [11] the authors use three types of analyzers to detect DDoS attacks. In particular, they consider statistical analysis of HTTP flows, users access behavior by graph modeling the paths of the web server as well as the different costs of navigating through the server, and the frequency of HTTP operations carried on the web server. The proposed model can autonomously detect bots and security scanners. However, it is ineffective against slow DoS attacks when number of involved bots is large. The work [2] presents a method that tracks the number of packets a web server receives in a given interval of time. This feature is monitored in two subsequent time intervals in order to detect normal or anomaly behavior. However, feature selection might be a long and complex activity. Monitoring the number of packets received can cause high false positive rate, because a high burst of traffic can be due to different scenarios. Authors in [9] propose a method to detect different types of attacks, such as DoS, by mining text logs in a critical industrial system.

Typically, in a production environment, misconfiguration of the web server could lead to security breaches and make DoS attacks easier for malicious users. Beside regular updating and patching the web server, it is essential to configure it for better security performance. Many popular web servers allow to enable DoS security modules with the aim to mitigate security infringements. Although the use of these modules is highly recommended, there is a lack of studies on the defense topic. Moreover, in some cases, these patches have the limitation of blocking legitimate requests from being served. As stated in [22], countermeasures against DoS/DDoS attacks can be roughly classified in three categories: *survival* techniques, *proactive* techniques and *reactive* techniques. In [23] the authors show that a web server configured with the security modules can really mitigate the attack. However, if it is launched by an increasing number of bots, there is a noticeable delay in the server response; as such, a massive attack scenario can affect the performance of the server. The work [4] provides a detailed analysis of DDoS countermeasures. It highlights the strengths of each method, and also consider some countermeasures that can be taken against each de-

fense mechanism from the perspective of the attacker. A survey of DDoS defense mechanisms can be found in [29]. The authors provide a detailed classification of defense mechanisms driven by the moment in time when the defense should be applied: before the attack (*prevention*), during the attack (*detection*), or after the attack (*identification* and *response*). Authors in [26] present a study that highlights performance and defense mechanisms of Windows Server and Linux Server. The paper evaluates several defense mechanisms, such as *ACLs*, *threshold limit*, *reverse path forwarding* and *network load balancing*.

In the literature, many metrics have been proposed to evaluate both the impact of DoS attacks and defense strategies. However, there are no benchmarks [20] that allow to evaluate effective metrics. In [21] the authors propose the percentage of failed transactions (transactions that do not follow QoS thresholds) as a metric to measure DDoS impact. In particular, they describe a threshold-based model for traffic measurements. If a measurement exceeds the threshold, it indicates poor service quality. However, the duration of the transactions depends on the network load. Therefore, it is difficult to set the absolute duration threshold. *Server timeout* has been used as a metric in [17]. Unfortunately, the damage effects in terms of legitimate traffic drop are not indicated. Finally, in [12], Gupta et al. describe two statistical metrics namely, *Volume* and *Flow*, in order to efficiently detect DDoS attacks.

3 Testbed and Analysis Method

Our analysis is based on direct performance measurements of a victim server during DoS attacks performed in a controlled testbed. We collect a variety of service metrics to gain insight into the impact of the attacks in case of *no* and *with defense*. In the following we describe the experimental testbed, the DoS tools and the service metrics adopted.

3.1 Experimental Testbed

Experiments were conducted on a private cloud infrastructure hosted by a data-center at the University of Sannio. The experimental testbed capitalizes on our previous work [8] and consists of three Ubuntu 18.04.2 LTS nodes, described in the following using the naming shown in Fig. 1.

The “**victim**” node hosts an installation of Apache 2.4. This web server is a significant case study, given its widespread use. It can fit a wide-range of websites, ranging from personal blogs to websites that serve millions of users; moreover, it is open source and cross-platform. The Apache web server supports a variety of *modules* –including security-related capabilities– that can be enabled by adjusting the configuration of the *baseline* server installation. In our study we address `mod_evasive`, i.e., a consolidated **defense module** intended to protect a server from DoS, DDoS and bruteforce attacks. We carefully tuned and tested the correct functioning of `mod_evasive` according to the instructions from a well-

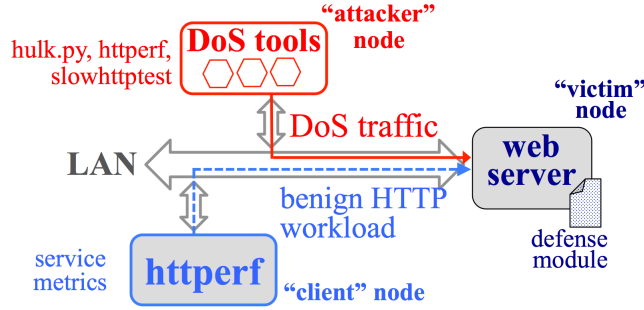


Fig. 1: Experimental testbed.

detailed tech blog⁴. The module inspects incoming requests at the server side and blacklists malicious client IP addresses based on several thresholds and timeouts, such as *site count*, *page count*, *blocking period*. At any time, `mod_evasive` can be seamlessly enabled or disabled by acting on the configuration and re-starting the web server. We conducted an experimental campaign with different attacks both against the *baseline* server and its version hardened by means of `mod_evasive`.

As for the remaining components in Fig. 1, the **"attacker" node** generates the DoS traffic intended to disrupt server operations. To this aim, we use several state-of-the-art attack tools, which are described in Section 3.2. Finally, the **"client" node** hosts `httperf`⁵, which is a widely-used workload generator. This tool makes it possible to set a desired level of workload by regulating different parameters, such as *total connections*, *connections per second* and *requests per connection* and triggers the normative HTTP requests, which aim to emulating the *benign* workload by a legitimate client. During the experiments, the web server is exercised with both DoS traffic and benign workload. We use native metrics produced by `httperf` and a set of derived metrics –detailed in Section 3.3– to monitor the web server and to measure the impact of the attack.

3.2 DoS Attacks and Tools

Our experiments encompass an interesting mixture of DoS attacks that exploit both flooding activities and the intrinsic design of the HTTP protocol. Attacks and their implementation in the testbed are presented in the following.

Hulk DoS. The attack aims to flood the victim server with a massive amount of HTTP requests. In particular, it continuously triggers single or multiple requests and generates a *unique pattern* at each time by randomizing key fields of the requests. Some attack strategies are reported in [15]. We use the well-consolidated `hulk.py` Python script available at GitHub⁶ to launch Hulk DoS against our victim server.

⁴ <https://www.atlantic.net/vps-hosting/how-to-install-and-configure-modevasive-with-apache-on-ubuntu-18-04/>

⁵ <https://github.com/httpperf/httpperf>

⁶ <https://github.com/grafov/hulk>

Bruteforce flooding. We supplement the analysis with a *home-made* flooding attack performed by means of an additional installation of `httperf` at the “attacker” node. To this aim, `httperf` –used in this case for flooding purposes– is configured to open 100 concurrent connections per second and to make 1,000 HTTP requests per connections, which sum up to total 100,000 requests per second: this request rate is by far higher than the actual processing capacity of the server in our testbed. Differently from Hulk DoS, the request pattern is constant; however, the magnitude of the flooding is much higher than Hulk.

Slow header. Differently from the above, *slow header* is not a “mere” flooding attack. Rather, it relies on an HTTP protocol design condition, which requires requests to be *completely* received by the server before they are actually processed. As such, the attack is accomplished by sending incomplete HTTP requests (i.e., without ever ending the header) that end up saturating the connections of the victim server. We use `slowhttpptest`⁷ to execute a *slow header* attack. The tool is configured to use a budget of 5,000 concurrent sockets, which lead to an attack duration of around 4 minutes in our testbed.

3.3 Collection of the Service Metrics

During the experiments we continuously probe the operational status of the server at regular intervals by alternating (i) a run of `httperf` at the “client” node and (ii) collection of the corresponding metrics. At each run, the **benign workload** implemented by means of `httperf` consists of 10 concurrent connections per second and 100 HTTP requests per connections, which sum up to a rate of 1,000 requests per second (*reqs/s*). While collecting the metrics, we set a request timeout of 10 *s* to avoid that `httperf` hangs by waiting for requests that might never succeed in case of attack. We focus on the following metrics from `httperf`:

- **reply rate or throughput (T)**: HTTP requests accomplished by the server within the time unit, measured in *reqs/s*;
- **mean response time (MRT)**: mean time taken to serve a request measured in milliseconds (*ms*);
- **OK requests**: number of requests served with the 200 HTTP status code by the server.

Beside the metrics above –“natively” provided by `httperf`– we compute the following derived metrics:

- **throughput loss (TL)**: percentage of HTTP requests that *are not* accomplished by the web server within the time unit, i.e., $TL = \frac{(1,000 - T)}{1,000} \cdot 100$, where 1,000 is the request rate. TL varies in the interval $[0, 100]\%$: any point where $TL \neq 0\%$ reflect a slowdown of the server in serving the requests.
- **mean OK requests per connection**: *OK requests* divided by 10, i.e., the number of concurrent connections issued by `httperf`.

⁷ <https://github.com/shekya/slowhttpptest>

In attack-free conditions, in response to the workload of 1,000 *reqs/s*, the metrics above are equal to the following values: $T=1,000$ *reqs/s*, $MRT=0.4$ *ms*, OK requests = 1,000, $TL=0\%$, mean OK requests per connection = 100. Given that in our testbed the only source of legitimate activity is the “client” node, any deviation from these “ideal” values points out the presence of a DoS activity.

4 Experimental Results

This section discusses the results obtained by running the attacks presented in Section 3.2 against the web server in hand. For each attack, we run two independent experiments. The former consists in performing the attack against the *baseline* server installation, i.e., no defense module in place; the latter is done by running the attack after having started the server with its defense module enabled. The two experimental scenarios are denoted by *NO defense* and *WITH defense* through the rest of the paper.

The duration of each experiment is 300 *s*, which is long enough to collect a sample of service metric observations for statistical purposes (e.g., 30 observations); in all the cases, the attack starts at $t=20$ *s* since the beginning of the experiment. At the end of each experiment we (i) store measurement data and logs for subsequent analysis, (ii) clear the logs of the web server, such as **access** and **error** log (iii) stop the workload generator, attack scripts and the web server. In the following, the results are presented by increasing service loss levels caused to the server.

4.1 Hulk DoS

Fig. 2a (\square -marked dashed series) shows how TL varies during the progression of the Hulk attack in case of *NO defense* for our server: the attack causes sporadic spikes of the TL , whose maximum observed value is 77.06% at $t=140$ *s* since the beginning of the run (x-axis). In order to gain insight into the effect of the attack, we complement TL with the *mean OK requests per connection* metric described above. It is worth noting that an increase of TL reflects the slowdown

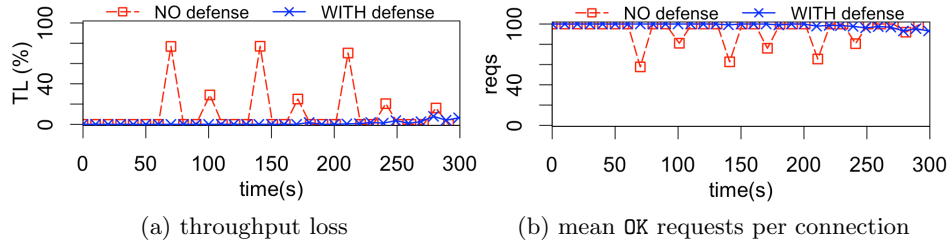


Fig. 2: Impact of the attack (**hulk** script).

```

-- access.log --
[01/Aug/2020:22:44:31 +0200] "GET /index.html?ZBWRUMRX=ANINEAZY HTTP/1.1"
200 11192 "http://engadget.search.aol.com/search?q=PPQ0B0V" "Mozilla/4.0
(compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2;.NET CLR
2.0.50727; InfoPath.2)"

```

Fig. 3: Response to a Hulk DoS HTTP request (*NO defense*).

```

-- access.log --
[01/Aug/2020:22:27:50 +0200] "GET /index.html?TOIHJNNI=ZXGMI HTTP/1.1"
403 459 "http://www.google.com/?q=QILIYIIDL" "Mozilla/5.0 (X11; U; Linux
x86_64; en-US; rv:1.9.1.3) Gecko/20090913 Firefox/3.5.3"
-- error.log --
[Sat Aug 01 22:27:50.377578 2020] [evasive20:error] [pid /*omitted*/]
[client 192.168.111.65:54508] client denied by server configuration:
/var/www/html/index.html, referer: http://www.google.com/?q=QILIYIIDL

```

Fig. 4: Response to a Hulk DoS HTTP request (*WITH defense*).

of the server at serving HTTP requests; however, it does not account for the correctness of the requests, which is summarized by the latter metric in Fig. 2b (\square -marked dashed series) for the *NO defense* scenario. Interestingly, for several data points the metric drops from 100, i.e., the *normative* value for attack-free conditions, to a lower value, such as 57.68 –the worst case– when $t=70$ s. Overall, when the server is not properly protected, Hulk DoS causes both the slowdown of the server and a relevant loss of correct requests at sporadic times.

Fig. 2a and Fig. 2b (\times -marked solid series) show TL and mean OK requests per connection when the defense module is on, i.e., *WITH defense*. It can be noted that the defense module is able to mitigate significant spikes and drops observed in the previous scenario. Nevertheless, in spite of the defense, the attack can still affect the server at some extent: the maximum TL is 7.88% and the minimum mean OK requests is 93.12. This is an interesting finding, which indicates that a server may fail to handle some requests also in case of defense.

We closely looked into the logs of the web server, i.e., *access.log* and *error.log*, to motivate the difference in the service loss caused by the Hulk DoS attack in both *NO* and *WITH defense* conditions. As mentioned above, the attack *floods* the server by means of malicious HTTP requests that consist of random URL query string, user agent and referee. Fig. 3 shows an example of such a request, logged by the web server in the *access.log* in case of *NO defense*. Surprisingly, the server accomplishes the malicious request with the 200 (OK) HTTP status code (enclosed in a box in Fig. 3): as a consequence, the computation overhead of serving a malicious request is the same as any other normative request. On the other hand, Fig. 4 shows how a malicious HTTP request by Hulk is tracked by the logs of the server in the *WITH defense* scenario. The most striking outcome is that the malicious request is now **forbidden** the access to the resource with the 403 HTTP status code (enclosed in a box in Fig. 4) and it raises a

Table 1: Service metrics (**hulk** script).

	availability	max TL	max MRT	percentage OK requests by MRT		
				[0, 10] ms	(10, 100] ms	(100, +∞]
<i>NO defense</i>	0.938	77.06%	0.9 ms	100.0%	0.0%	0.0%
<i>WITH defense</i>	0.988	7.88%	0.4 ms	100.0%	0.0%	0.0%

corresponding “**client denied**” notification in the *error.log*. Differently from *NO defense*, the malicious request is aborted before it triggers the normative computation; however, it still needs some handling cycles by the server in order to be flagged as forbidden. As a consequence, although *per-request* overhead is negligible, the overall contribution of a significant flooding activity can affect the server also in case of defense.

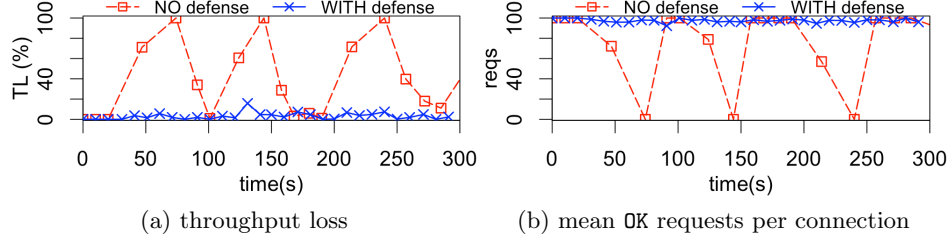
Table 1 summarizes key service metrics obtained for Hulk DoS, such as the *availability*. Through the rest of the paper, we compute the **availability** as the ratio between the number of successful HTTP requests observed during an attack-injection experiment divided by the number of successful HTTP requests that the server would accomplish in attack-free conditions over the same interval. The number of successful HTTP requests is related to an interval of 300 s (i.e., the duration of the experiment). *Availability* can be seen here as the probability that the server will successfully handle a HTTP request under attack.

As shown in Table 1, *availability* is 0.988 in case of defense and thus higher than the value measured in *NO defense*, i.e., 0.938; however, it is not exactly 1 (i.e., attack-free conditions) because –although with defense– the server fails to successfully handle some HTTP requests, as presented above. Another interesting outcome is given by the *percentage of OK requests by MRT*, shown in the rightmost columns of Table 1, where we break down the total number of successful requests reported by the workload generator based on their MRT. It can be noted that both in *NO* and *WITH defense* scenarios, 100% of OK requests are served with a MRT in [0, 10] ms. Noteworthy, 10 ms is seen as a typical maximum tolerable delay for a response of a web server in order to be usefully deployed in multilayer workflows [3]. As such, in our testbed Hulk DoS did not affect the *usability* of successful requests. Nevertheless, it is worth noting that the maximum MRT observed in case of *NO defense*, i.e., 0.9 ms, is significantly higher than *WITH defense*, i.e., 0.4 ms.

4.2 Bruteforce flooding

The following paired runs encompass our *home-made* bruteforce flooding; again, we assess the behavior of the server in case of *NO* and *WITH defense*.

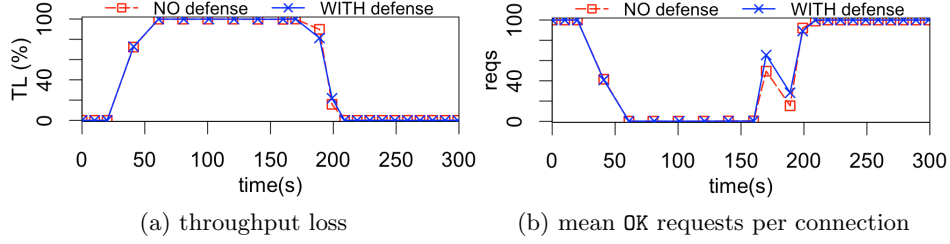
Fig. 5a and Fig. 5b (□-marked dashed series) show how TL and mean OK requests per connection vary during the *NO defense* run. At three data points, i.e., $t=74, 144$, and 240 s the metrics reach 100% and 0, respectively. This

Fig. 5: Impact of the attack (`httperf` bruteforce flooding).Table 2: Service metrics (`httperf` bruteforce flooding).

	availability	max TL	max MRT	percentage OK requests by MRT		
				[0, 10] ms	(10, 100] ms	(100, +∞]
<i>NO defense</i>	0.498	100.00%	46.5 ms	86.6%	13.4%	0.0%
<i>WITH defense</i>	0.975	15.74%	0.6 ms	100.0%	0.0%	0.0%

means that, during those runs of `httperf` at the “client” node, the web server succeeds to accomplish *none* of the requests. Overall, the attack strongly affects the normative operation of the server through all the duration of the run. For example, it can be noted that $TL=0\%$ in almost no point of the x-axis in Fig. 5a. As for the *WITH defense* run, Fig. 5a and Fig. 5b (\times -marked solid series) make it possible to appreciate the invaluable benefit of the defense module for both the throughput and OK requests. TL is constantly around its mean value of 3.36% over the duration of the run with a peak of 15.74% in the worst case; similarly, *mean OK requests per connection* varies around a mean of 97.20. Both the values are really close to 0% and 100, i.e., the ideal values in attack-free conditions. Again, in spite of the strong mitigation effect, there is a performance loss in the *WITH defense* scenario. Such as for Hulk DoS, this is due to the server cycles needed to forbid the massive amount of malicious requests with the 403 HTTP status code, which is more significant in this attack.

Table 2 summarizes key service metrics obtained in face of bruteforce flooding. *Availability* is 0.498 in *NO defense*: the server failed to successfully reply around half of the HTTP requests issue by the workload generator when compared to the attack-free scenario. Moreover, 13.4% out of the total successful requests are replied with a MRT in (10, 100] ms (i.e., *percentage of OK requests by MRT* in Table 2), which means that a substantial number of correct replies is strongly delayed because of the attack. *Availability* in case of defense is 0.975, such as shown in bottom row of Table 2. This value is lower than 0.988, which was measured for *WITH defense* in Hulk DoS. Although far from the “ideal” value of 1, it is reasonably satisfactory given the magnitude of the flooding. Noteworthy, the defense module allows all successful requests to be served with a MRT in [0, 10] ms.

Fig. 6: Impact of the attack (*slowhttptest* slow header).

4.3 Slow Header

The last attack addressed in this study is *slow header*. Fig. 6a and Fig. 6b (\square -marked dashed series) show how TL and mean OK requests per connection vary during the *NO defense* run. It can be noted that the attack is able to quickly saturate the web server: TL is 99.8% starting from $t=60$ s –thus 40 s since the beginning of the attack– up to $t=190$ s, which means that the server fails to handle almost all benign HTTP requests of the workload generator during this timeframe. Similar considerations apply to mean OK requests per connection, whose value is 0 over the same timeframe. The server resumes its normative operations at $t=210$ s after a short transitory. Fig. 6a and Fig. 6b (\times -marked solid series) show the metric for the *WITH defense* run: the appearance of both the time series is very close to the *NO defense* experiment.

Since the presence of the defense module reflects into *no* significant change of the service metrics at application level, we performed further analysis to support the claim that the module assessed in this study provides no mitigation for *slow header* attacks. To this objective, we leverage the outcome of *slowhttptest*, which accounts for the *socket usage*. Fig. 7 compares the number of *opened* and *connected* sockets during the progression of the attack for *NO* and *WITH defense* (dashed and solid series, respectively). The 5,000 sockets mentioned in Section 3.2 are opened at the beginning of the attack at $t=20$ s and progressively closed

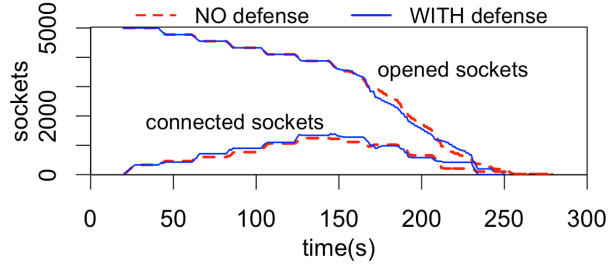
Fig. 7: Socket usage by *slowhttptest*

Table 3: Service metrics (`slowhttpptest` slow header).

	availability	max TL	max MRT	percentage OK requests by MRT		
				[0, 10] ms	(10, 100] ms	(100, +∞]
<i>NO defense</i>	0.001	100.00%	2107.8 ms	0.0%	0.0%	100.0%
<i>WITH defense</i>	0.001	100.00%	2132.2 ms	0.0%	0.0%	100.0%

later on. Moreover, the number of connected sockets reaches a maximum of 1,400 around $t=150$ s, which is consistent with the TL in Fig. 6b. Interestingly, mitigation actions implemented by the defense module do not affect at all the attacker’s *socket usage*, which is the same in *NO* and *WITH defense*.

As for the summary of the metrics⁸, shown in Table 3, we note that *availability* is 0.001 in both the runs: during the attack, the server accomplished almost none of the requests of the workload generator with respect to an attack-free scenario. It is worth noting that out of the very small number of requests (e.g., 120 in the *WITH defense* run) that occasionally succeed during the attack, 100% are served with a MRT higher than 100 ms and thus unable by a potential client.

Overall, it can be reasonably claimed that the defense module assessed in this study provides no mitigation for *slow header* attacks. As such, the module should be accompanied by supplemental defense mechanisms in order to achieve comprehensive defense in production environments.

5 Conclusion

This paper proposed an initial measurement study assessing a DoS defense module for a widely-used web server. Results highlighted *pros* and *cons* of the module. Most notably, we observe that, in spite of the defense, a web server may occasionally fail to serve legitimate requests; moreover, the module in hand does not protect from *slow header* attacks.

Our study builds around the intuition that existing defense modules might not provide exhaustive coverage for all modern DoS attacks: this is pursued by instantiating the study in the context of a widely-used web server. As for any measurement study, there may be concerns regarding the validity and generalizability of results, based on the aspects listed in [28]. We rely on a mixture of experiments consisting of direct emulation of DoS attacks by means of state-of-the-art tools. More important, we support our findings with a significant range of service metrics and by direct inspection of socket usage. Overall, these mitigate internal validity threats and provide a reasonable level of confidence on the experimental results. Noteworthy, our analysis can be reasonably ported to other web servers and attacks.

⁸ Differently from previous attacks, which span the entire duration of the experiment, here metrics are computed with reference to the interval [60, 160] s, which reflects the full magnitude of the attack, with no transitory included.

In the future we will extend our measurements to further defense modules, attack types and web servers. There are several modules that can be used by practitioners to harden production servers; however, there is a lack of comparative studies and frameworks providing clear guidance and measurements. In this respect, we will augment the method, set of metrics and assessment framework –currently focused on DoS attacks– based on the type of attacks and corresponding defense modules. Our work is extremely useful to tune defense mechanisms, to find a well-balanced mix between defense modules and performance overhead, and to discover attacks that are not covered yet by existing modules.

References

1. Agarwal, M., Pasumarthi, D., Biswas, S., Nandi, S.: Machine learning approach for detection of flooding DoS attacks in 802.11 networks and attacker localization. *Int. J. Mach. Learn. & Cyber.* **7**, 1–17 (2014)
2. Aiello, M., Cambiaso, E., Mongelli, M., Papaleo, G.: An on-line intrusion detection approach to identify low-rate DoS attacks. In: *Proc. International Carnahan Conference on Security Technology*. pp. 1–6. IEEE (2014)
3. Alizadeh, M., Greenberg, A., Maltz, D.A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., Sridharan, M.: Data center TCP (DCTCP). *ACM SIGCOMM Comput. Commun. Rev.* **40**(4), 63–74 (2010)
4. Beitollahi, H., Deconinck, G.: Analyzing well-known countermeasures against distributed denial of service attacks. *Computer Commun.* **35**(11), 1312 – 1332 (2012)
5. Bonguet, A., Bellaiche, M.: A survey of denial-of-service and distributed denial of service attacks and defenses in cloud computing. *Future Internet* **9**(3), 43 (2017)
6. Cambiaso, E., Aiello, M., Mongelli, M., Vaccari, I.: Detection and classification of slow DoS attacks targeting network servers. In: *Proc. International Conference on Availability, Reliability and Security*. Art. No.: 61. pp. 1 – 7. ACM (2020)
7. Catillo, M., Pecchia, A., Rak, M., Villano, U.: A case study on the representativeness of public DoS network traffic data for cybersecurity research. In: *Proc. International Conference on Availability, Reliability and Security*. Art. No.: 6. pp. 1–10. ACM (2020)
8. Catillo, M., Pecchia, A., Villano, U.: Towards a framework for improving experiments on DoS attacks. In: Shepperd, M., Brito e Abreu, F., Rodrigues da Silva, A., Pérez-Castillo, R. (eds.) *Quality of Information and Communications Technology*. pp. 303–316. Springer International Publishing, Cham (2020)
9. Cinque, M., Della Corte, R., Pecchia, A.: Contextual filtering and prioritization of computer application logs for security situational awareness. *Future Generation Computer Systems* **111**, 668 – 680 (2020)
10. Dharini, N., Balakrishnan, R., Renold, A.P.: Distributed detection of flooding and gray hole attacks in wireless sensor network. In: *Proc. International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials*. pp. 178–184. IEEE (2015)
11. Giralte, L.C., Conde, C., de Diego, I.M., Cabello, E.: Detecting denial of service by modelling web-server behaviour. *Computers & Electrical Engineering* **39**(7), 2252 – 2262 (2013)
12. Gupta, B.B., Misra, M., Joshi, R.: An ISP level solution to combat DDoS attacks using combined statistical based approach. *International Journal of Information Assurance and Security* **3**, 102–110 (2012)

13. Ismail, M.N., Aborujilah, A., Musa, S., Shahzad, A.: Detecting flooding based DoS attack in cloud computing environment using covariance matrix approach. In: Proc. International Conference on Ubiquitous Information Management and Communication. Art. No.: 36. pp. 1–6. ACM (2013)
14. Kang, B., Kim, D., Kim, M.: Real-time connection monitoring of ubiquitous networks for intrusion prediction: A sequential knn voting approach. *International Journal of Distributed Sensor Networks* **11**(10), 1–10 (2015)
15. Kaur, H., Behal, S., Kumar, K.: Characterization and comparison of distributed denial of service attack tools. In: Proc. International Conference on Green Computing and Internet of Things. pp. 1139–1145. IEEE (2015)
16. Kemp, C., Calvert, C., Khoshgoftaar, T.: Utilizing netflow data to detect slow read attacks. In: Proc. International Conference on Information Reuse and Integration. pp. 108–116. IEEE (2018)
17. Ko, C., Hussain, A., Schwab, S., Thomas, R., Wilson, B.: Towards systematic IDS evaluation. In: Proc. DETER Community Workshop. pp. 20–23 (2006)
18. Liu, H., Lang, B.: Machine learning and deep learning methods for intrusion detection systems: A survey. *Applied Sciences* **9**(20), 4396 (2019)
19. Mahjabin, T., Xiao, Y., Sun, G., Jiang, W.: A survey of distributed denial-of-service attack, prevention, and mitigation techniques. *International Journal of Distributed Sensor Networks* **13**(12), 1–32 (2017)
20. Mirkovic, J., Arikan, E., Wei, S., Thomas, R., Fahmy, S., Reiher, P.: Benchmarks for DDOS defense evaluation. In: Proc. Military Communications Conference. pp. 1–10. IEEE (2006)
21. Mirkovic, J., Hussain, A., Wilson, B., Fahmy, S., Reiher, P., Thomas, R., Yao, W.M., Schwab, S.: Towards user-centric metrics for denial-of-service measurement. In: Proc. Workshop on Experimental Computer Science. pp. 1–14. ACM (2007)
22. Mirkovic, J., Reiher, P.: A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Comput. Commun. Rev.* **34**(2), 39–53 (2004)
23. Moustis, D., Kotzanikolaou, P.: Evaluating security controls against HTTP-based DDoS attacks. In: Proc. IISA. pp. 1–6. IEEE (2013)
24. Nagaratna, M., Prasad, V.K., Kumar, S.T.: Detecting and preventing ip-spoofed ddos attacks by encrypted marking based detection and filtering (EMDAF). In: Proc. International Conference on Advances in Recent Technologies in Communication and Computing. pp. 753–755. IEEE (2009)
25. Sikora, M., Gerlich, T., Malina, L.: On detection and mitigation of slow rate denial of service attacks. In: Proc International Congress on Ultra Modern Telecommunications and Control Systems and Workshops. pp. 1–5. IEEE (2019)
26. Treseangrat, K., Kolahi, S.S., Sarrafpour, B.: Analysis of UDP DDoS cyber flood attack and defense mechanisms on Windows Server 2012 and Linux Ubuntu 13. In: Proc. International Conference on Computer, Information and Telecommunication Systems. pp. 1–5. IEEE (2015)
27. Tripathi, N., Hubballi, N.: Slow rate denial of service attacks against HTTP/2 and detection. *Computers & Security* **72**, 255 – 272 (2018)
28. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: *Experimentation in Software Engineering: An Introduction*. Kluwer Academic (2000)
29. Zargar, S.T., Joshi, J.B.D., Tipper, D.: A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *IEEE Communications Surveys & Tutorials* **15**, 2046–2069 (2013)
30. Zhijun, W., Wenjing, L., Liang, L., Meng, Y.: Low-rate DoS attacks, detection, defense, and challenges: A survey. *IEEE Access* **8**, 43920–43943 (2020)