Automated DDoS mitigation based on known attacks using a Web Application Firewall

Julik Keijer
University of Twente
P.O. Box 217, 7500 AE Enschede
The Netherlands
j.s.keijer@student.utwente.nl

ABSTRACT

Distributed Denial-of-Service (DDoS) attacks are an ever growing problem with large societal impact. To defend against DDoS attacks we need to set up defences in every possible layer. There is still very little defence in some of the layers. For this research we designed a system for application layer defence against HTTP flood DDoS attacks. We used fingerprints based on known attacks to create rules that can be used in a Web Application Firewall (WAF).

1. INTRODUCTION

Distributed Denial of Service (DDoS) attacks happen often and impact a large number of people [4]. In July of 2018 there were numerous attacks against DigID, a governmental service in the Netherlands. These kinds of attacks are carried out worldwide and have great societal impact. In these kinds of attacks, DDoS is used to overload networks and in that way deny actual users access to the service. The attacks mentioned above use enormous amounts of bandwidth.

The type of DDoS attack this research focusses on is aimed at the application layer [2]. These types of attacks use less bandwidth, however they are equally effective in denying valid traffic to the attack target. Since these attacks use less bandwidth and use requests that seem like normal traffic, they are harder to distinguish from genuine traffic and therefore harder to defend against. In Q2 of 2018 Verisign [17] analysed that 5% of all DDoS attacks were aimed at the application layer.

To defend against application layer attacks, we need to look at application layer defence. One of the defence mechanisms at the application layer is a Web Application Firewall (WAF). A WAF applies certain rules that defend against malicious traffic. Generally these rules are used against cross site scripting or SQL injections, however they can also be used against DDoS attacks. The largest open source WAF is ModSecurity [9]. ModSecurity is already widely implemented and using an open source WAF gives larger freedom in experimentation.

To generate correct rules to be used in the WAF we will look at known attacks and generate rules that apply to these known attacks. To obtain known attacks we will use DDoSDB [5]. DDoSDB holds the characteristics (fingerprints) of a large number of DDoS attacks. Using these fingerprints we can automatically generate appropriate rules against DDoS attacks and use them to defend against application layer DDoS attacks.

To create an application which is able to automatically generate the correct rules, this research is split into the following three Research Questions (RQ).

- RQ1: Which technologies are available to mitigate DDoS attacks at the application layer?
- RQ2: How can we automatically generate WAF rules in order to mitigate application layer DDoS attacks?
- RQ3: What is the performance of the automatically generated WAF rules against application layer DDoS attacks?

In the following section, we will examine the related work and in section 3 through 5 we will further elaborate on the Research Questions. In section 6 the conclusions will be discussed and in section 7 future work will be laid out.

2. RELATED WORK

There is very little related work available when looking at application layer DDoS attacks. Some of the available papers only focus on detection [19]. These papers propose different detection mechanisms to identify DDoS attacks, however they do not propose a defence mechanism. There are also papers that intend to identify and defend against DDoS attacks [20]. This paper did a survey over existing techniques and identifies the properties needed in application layer DDoS protection.

3. EVALUATION OF ATTACK TYPES

To understand the possible defence at the application layer, we must first asses the attack types at the application layer. For this research we look at the two major types of application level attacks:

1. Low and Slow attacks;

2. HTTP flood attacks.

These attacks are much more sophisticated and precise than most transport layer attacks. Therefore they are not detectable with traditional DDoS mitigation tools and a custom approach must be taken.

In the following subsections we will discuss the attack types and the available mitigation strategies.

3.1 Low and Slow attacks

We will discuss two types of Low and Slow attacks:

1. Slowloris attacks;

2. R-U-Dead-Yet attacks.

3.1.1 Slowloris

In a Slowloris attack the attacker tries to open as many connections to the target as possible using HTTP GET requests. The attacker then keeps the connection open by slowly sending request header data to the target, without ever completing the request header. Once the attacker holds all connections it is impossible for valid traffic to reach the server.

3.1.2 R-U-Dead-Yet

In a R-U-Dead-Yet attack the attacker tries to keep the server waiting using HTTP POST requests. The attacker informs the target server that they are sending a large amount of data, but only does so very slowly. Therefore the attacker keeps the connection open. When this is repeated multiple times all connections to the target server are taken up.

3.1.3 Solutions

There are different commercial solutions available to mitigate Low and Slow attacks [14], [15]. Their solutions rely on reverse proxies, which means that they route all traffic via their own servers and only send complete requests to the target server. This means that the burden of the attack only falls upon the proxy servers, which are more capable to handle the attacks than normal servers.

Among the academic solutions there are different approaches. One part of the solutions focuses on assessing the validity of the packets using a trust score [22], while other solutions rely on analysing web traffic using characteristics of different types of attacks [18].

3.2 HTTP flood

In a HTTP flood attack a large number of legitimate HTTP request is sent to the attack target. These requests are designed to cause as much strain on the attack target as possible while costing very little bandwidth. This makes it possible for the attack to be executed using only a few attackers. The attack is launched using either HTTP GET or POST. With a HTTP GET flood the requests are typically to load images and with a HTTP POST flood the requests often cause the server to execute database queries that take a large amount of time to execute.

3.2.1 Low Orbital Ion Cannon

The Low Orbit Ion Cannon (LOIC) [7] is a network stressing tool suitable for executing HTTP flood attacks. It can be used from just one device, but it can also be used in a large network to execute DDoS attacks. The LOIC has been used in large attacks responding to the relinquished support of Wikileaks [16]. Since we have to simulate a DDoS attack, we will use the Low Orbit Ion Cannon as a reference for a HTTP flood attack.

3.2.2 Solutions

The commercial solutions [6], [3] both indicate that their approach to blocking HTTP flood attack is using traffic analysis and IP reputation. In this academic survey [21] a wide range of solutions, among which IP reputation is presented as a good solution towards HTTP flood attacks.

3.3 Findings

To answer the research question: *"Which technologies are available to mitigate DDoS attacks at the application layer?"* we find that it is difficult to defend against application layer DDoS attacks. The fact that malicious traffic is hard to distinguish from valid traffic makes it complicated for traditional defence mechanisms to defend against application layer attacks.

When we examine signature based defences we observed that the most common method against application layer DDoS attacks is using IP reputation and traffic analysis.

Attack type		IP reputation	Traffic analysis	Proxy servers
Low and Slow	[15]			x
	[14]			x
	[22]		x	
	[18]	x		
HTTP flood	[3]	x	x	
	[6]	x	x	
	[21]	x	x	x

Table 1: Literature examination

4. MITIGATION STRATEGIES

To generate WAF rules this research proposes to use the fingerprints from DDoSDB [5]. DDoSDB is a large database with known DDoS attacks. From these attacks fingerprints are constructed. DDosDB holds many types of DDoS attack, however there are currently no HTTP attacks in DDoSDB. Therefore we will create our own application layer attacks and create fingerprints from those attacks to be used in rule creation.

Even though there are currently no HTTP fingerprints available in DDoSDB, this research proposes to develop WAF rules based on the DDoSDB fingerprints. Since such fingerprints might be available in the future.

4.1 Fingerprint analysis

Firstly the available fingerprints from DDoSDB were analysed. Fingerprints are build up from the following parameters of the DDoS attack:

1. Protocol of the attack
2. Start time of the attack
3. Duration of the attack in seconds
4. Destination ports
5. Source IP addresses
6. Source ports

In the parameters the attacker is indicated as source and the attack target is indicated as destination.

Since the destination port at which an application layer attack is targeted is always 80 or 443. And source ports can be chosen at random. We found that the parameter information that is most useful is the source IP.

From the findings in section 3.3 we gathered that one of the effective measures against application layer DDoS attacks is using IP reputation.

4.2 WAF rules

We identified the Source IP as the best indicator for a DDoS attack based on the available fingerprint data. Therefore we looked at possible rule implementation of WAF rules. As described in the introduction our chosen WAF is ModSecurity [9].

ModSecurity is a module that can be installed in Apache or Nginx. For the purpose of this research ModSecurity has been installed on an Apache Server [1]. ModSecurity can be used to inspect every aspect of web-traffic and execute rules upon the traffic. As a standard rule set for ModSecurity the OWASP core rule set [10] has been designed. Apart from the core rule set it is possible to design custom rules for ModSecurity, which has been done for the purpose of this research. A ModSecurity rule is constructed in the following way:

```
SecRule VARIABLES OPERATOR [ACTIONS]
```

Where the `VARIABLES` can be used to define the parameters you want to evaluate, for example an IP address. The `OPERATOR` can be used to match against the `VARIABLES`, for example `"@ipmatch 127.0.0.1"`. The `ACTIONS` describe what is to be done once the rule is triggered, for example deny the packet.

Since ModSecurity handles the variables we can easily match against known malicious IP addresses. The most suitable way is to create an IP blacklist and let ModSecurity check against these IP addresses. The rule that is constructed for that purpose is the following: `SecRule REMOTE_ADDR "@ipMatchFromFile ip_blacklist.txt"`

Where the `REMOTE_ADDR` is the IP address of the attacker and `ip_blacklist.txt` is a list of known malicious IP addresses.

4.3 Mitigation

When mitigating application layer attacks we will divide the attacks in the two types mentioned in section 3. Low and Slow attacks and HTTP flood attacks.

4.3.1 Low and Slow

To understand the mitigation of Low and Slow attacks there has to be understanding of the process ModSecurity goes through when analysing web-traffic [8]. The phase in which the check against IP addresses is done is phase 1. This phase is executed after the request headers have been parsed and before the request body is parsed. This is described in figure 1.

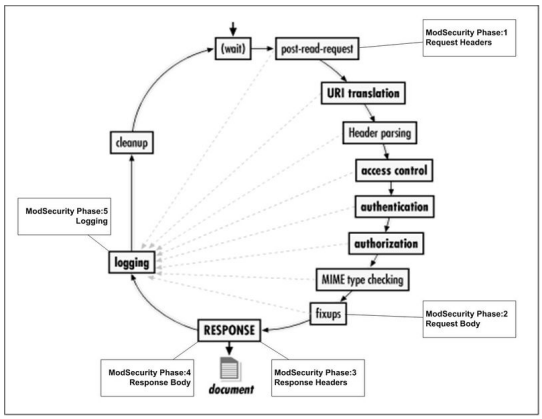

Figure 1: ModSecurity phases [8]

In a Low and Slow attack all request are stuck in the `SERVER_BUSY READ/WRITE` state, which is even before phase 1. A Slowloris attack abuses the `READ` state by making the server wait for a `READ` request and a R-U-Dead-Yet attack abuses the `WRITE` state by making the server wait for a `WRITE` request. Therefore a Low and Slow attack cannot be detected using ModSecurity and the IP blacklist cannot be used. There are however countermeasures designed for ModSecurity

and Apache. For ModSecurity there is the configuration `SecConnReadStateLimit` [12] and the configuration `SecConnWriteStateLimit` [13]. These configurations dictate how many open connections one IP address may have in either the `READ` and `WRITE` states. Using these configurations the effects of a Slowloris attack and a R-U-Dead-Yet attack can be mitigated. In Apache the module mod_reqtimeout [11] can be used to mitigate Low and Slow attacks. With this module it is possible to give a maximum time to complete a request header or a request body. It is also possible to set a minimum speed for data transfer, this mitigates Low and Slow attacks.

4.3.2 *HTTP flood*

HTTP flood attacks rely on a long computation time on the server. Therefore it is critical that the requests are denied without the server spending much time per request. For this the IP blacklist is ideal. With a HTTP flood attack we evaluate the request in phase 1. There we can decide if a request is malicious and drop the request without the server spending any more computation time. The following rule was used:

```
SecRule REMOTE_ADDR "@ipMatchFromFile
ip_blacklist.txt" "phase:1,drop,nolog"
```

This is the same rule as discussed in the subsection WAF rules 4.2, with the actions `drop` and `nolog`. Drop indicates that the request is dropped without any further action and nolog indicates that there will be no log entry, if there is a log entry created per request there is still the possibility that the server will become overloaded because of the log file size.

4.4 Mitigation outside of fingerprints

When looking outside of the fingerprint information it is possible to identify characteristics of a HTTP flood attack, which can be used to mitigate the attack. When evaluating the attack options of the LOIC as pictured in figure 2 we see that the option `append random chars to the URL` is present. This option makes it possible to request random pages or make random requests. This option makes it harder to defend against, since requests seem legitimate. The requests that results from this option can be seen in figure 3, there the random requests is as follows: `GET /GIMGPV`.

ModSecurity offers different tools to use this information in the mitigation. With ModSecurity it is possible to assign variables to an IP address. This makes it possible to ban IP addresses that perform suspicious actions.

4.4.1 *Using IP variables*

The following rules have been identified to mitigate the aspect of requesting random pages using the LOIC:

Rule 1:

```
SecRule RESPONSE_STATUS "Streq 404" "phase:5,
id:1040
log,
msg:'Invalid page request',
setvar:ip.dos_counter=+1,
expirevar:ip.dos_counter=60"
```

Rule 2:

```
SecRule IP:DOS_COUNTER "gt 10" "phase:1,
id:'1041',
drop,
log,
msg:'Client Connection Dropped due to high # of
invalid page request'"
```

Rule 1 evaluates the response Apache formulates to the requests. In an attack where random pages are requested the server will respond with HTTP code 404, meaning that the page could not be found. This rule then adds to the variable `ip.dos_counter` and sets the expiration time to 60 seconds. This means that every time an IP address requests a page that does not exist it will be registered and that information will be kept for 60 seconds.

Rule 2 evaluates if the variable `ip.dos_counter` is greater than 10. Meaning that there were more than 10 request to pages that do not exist in the last 60 seconds. When there is an IP address that matches that requirement all further requests, within those 60 seconds, will be dropped.

4.5 Findings

To answer the research question: *"How can we automatically generate WAF rules in order to mitigate application layer DDoS attacks?"* we evaluated the different attack types and their mitigation strategies.

When evaluating the mitigation of Low and Slow attacks we found that the attack occurs before ModSecurity rules are triggered. Therefore we find that it is not possible to use ModSecurity rules, based on fingerprints to mitigate these attacks. However there are methods available within ModSecurity and Apache to mitigate these attacks which are not present in the core rule set of ModSecurity and are therefore not widely implemented. When evaluating the mitigation of HTTP flood attacks we find that using the parameters available from DDoSDB and the rules available within ModSecurity we can construct an IP blacklist against which the incoming traffic can be checked. When looking at mitigation strategies outside of the fingerprint information a mitigation method using traffic analysis was found.

5. MITIGATION PERFORMANCE

From the findings in section 4.5 we have gathered that it is not possible to use automatically generated

WAF rules against Low and Slow attacks. Therefore we will focus on HTTP flood attacks when evaluating the performance.

5.1 Attack

We will use the Low Orbit Ion Cannon (LOIC) as the attack tool to execute a HTTP flood attack as pictured in figure 2. In this test set-up the Apache server is running on the same device from which the attack is being launched. Therefore the attack destination is set as `IP:127.0.0.1` with `PORT:80`.

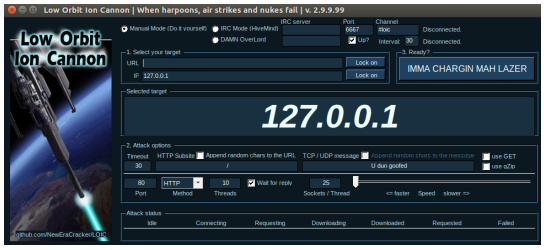

Figure 2: Low Orbit Ion Cannon

In the test set-up the LOIC generates random HTTP GET requests for the target server to fetch, which are depicted in figure 3.

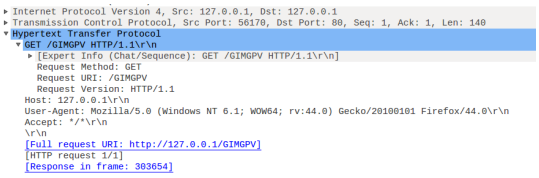

Figure 3: LOIC network packet

Since these are valid HTTP GET request it difficult to identify them as malicious traffic.

To test our ModSecurity rules we constructed the following fingerprint based on this attack.

Parameter	Value
protocol	"HTTP"
start_time	"2019-01-16 14:25:51"
dst_ports	[80]
duration_sec	48.475940227508545
src_ips	[{ip: "127.0.0.1"}]
src_ports	[56170]

Table 2: HTTP flood fingerprint

5.2 IP blacklist defence

We implemented the rule which was identified in section 4.3.2 to mitigate HTTP flood. The rule is dependant on the IP blacklist.

To have a representative IP blacklist we added all IP addresses from DDosDB from the `QUIC` protocol. A protocol similar to HTTP. From the 45 attack fingerprints found for this protocol we extracted over 1.2 million IP addresses for our blacklist. Based on the fingerprint in table 3 we added the `IP:127.0.0.1` to the IP blacklist.

5.2.1 Performance

To test the performance of the ModSecurity rule with the IP blacklist we sent a single request to the Apache Server. Which resulted in the following:

Phase	Computation time in microseconds
1:header processing	509
5:logging	149
Total	659

Table 3: IP blacklist performance

Table 3 indicates that the computation for our rule only takes half a millisecond per request. Which means that the server can handle at least one thousand requests per second for an IP blacklist this size.

5.2.2 Blocking

Using the IP Blacklist we were able to block 100 percent of the malicious traffic, since the IP address of the attacker was known.

5.3 Traffic analysis defence

We added the rules specified in section 4.4.1 to the rule set. Since the attack was coming from one IP address almost the entire attack was mitigated. The first 10 requests were needed to trigger the second rule. After which all packets were dropped. In an attack where a large number of requests per second are sent the first 10 requests are trivial. Which means that using this mitigation strategy a block percentage of over 99 percent can be achieved.

5.4 Findings

To answer the research question: *"What is the performance of the automatically generated WAF rules against application layer DDoS attacks?"* we find that we were able to mitigate 100 percent of a HTTP flood attack using the fingerprint of that attack. We were able to automatically add the parameters of this fingerprint to an IP blacklist which was used to mitigate an HTTP flood attack. We demonstrated the mitigation using attack launched by the LOIC.

We find that using the rules for traffic analysis a mitigation of over 99 percent can be achieved.

6. CONLUSIONS

In this research we analysed different application layer attacks and their defences. In section 3 we reviewed

application level DDoS attacks and their possible mitigation strategies. In that section we concluded that mitigations using IP reputation and traffic analysis are most implemented. In section 4 we proposed a mitigation strategy for HTTP flood attacks. We created a fingerprint for the HTTP flood attack and used that fingerprint to create a WAF rule to mitigate the attack. In section 5 we implemented the rule in ModSecurity and showed that using rule and the IP blacklist were able to completely mitigate the HTTP flood attack. We also showed the possibility to use traffic analysis rules to mitigate HTTP flood attacks.

In section 4.5 we concluded that it is not possible to mitigate Low and Slow attacks using ModSecurity rules based on fingerprints, however we showed that there are mitigation methods available for Apache and ModSecurity.

7. FUTURE WORK

While we were able to completely mitigate the HTTP flood attack using the fingerprint of the attack, we identify that these defence mechanisms can be circumvented using proxies or IP spoofing. We proposed a mitigation strategy based on traffic analysis, it is important for future work to expand this mitigation strategy by expanding the attack fingerprint to also hold information on the HTTP request. Using an expanded fingerprint with, for example the user agents or the GET request information. Using this information it could be possible to identify malicious traffic without relying on the IP address.

ModSecurity has far more capabilities than IP matching, such as the IP variables. If the fingerprint information is extended it is possible to make use of more of the capabilities of ModSecurity.

Currently there are no HTTP attacks in DDoSDB. We showed that with the right information ModSecurity can be highly effective. Therefore it is important that there will be HTTP attacks added to DDoSDB. Using the attack traces it might be possible to identify more characteristics of application layer attacks.

We concluded in this research that it is not possible to mitigate Low and Slow attacks using ModSecurity rules based on fingerprints. In future work it might be possible to block Low and Slow attacks by evaluating the requests before Apache does. By using the fingerprint information of a Low and Slow attack it might be possible to block the attack before it reaches the read/write state.

8. REFERENCES

[1] Apache. https://www.apache.org/. Accessed: 2018-01-20.

[2] Application layer ddos attacks. https://www.cloudflare.com/learning/ddos/application-layer-ddos-attack/. Accessed: 2018-11-28.

[3] Cloudflare-http-flood. https://www.cloudflare.com/learning/ddos/http-flood-ddos-attack/. Accessed: 2018-01-20.

[4] Ddos attacks in q3 2018. https://securelist.com/ddos-report-in-q3-2018/88617/. Accessed: 2018-11-29.

[5] Ddosdb. https://ddosdb.org/about. Accessed: 2018-11-28.

[6] incapsula-http-flood. https://www.incapsula.com/ddos/attack-glossary/http-flood.html. Accessed: 2018-01-20.

[7] Lioc. https://github.com/NewEraCracker/LOIC/. Accessed: 2018-01-20.

[8] mod-phases. https://github.com/SpiderLabs/ModSecurity/wiki/Reference-Manual-%28v2.x%29#Processing_Phases. Accessed: 2018-01-20.

[9] Modsecurity. https://www.modsecurity.org/about.html. Accessed: 2018-11-28.

[10] Owasp-crs. https://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project. Accessed: 2018-01-20.

[11] reqtimeout. https://httpd.apache.org/docs/2.4/mod/mod_reqtimeout.html. Accessed: 2018-01-20.

[12] Secconnreadstatelimit. https://github.com/SpiderLabs/ModSecurity/wiki/Reference-Manual-%28v2.x%29#SecConnReadStateLimit. Accessed: 2018-01-20.

[13] Secconnwritestatelimit. https://github.com/SpiderLabs/ModSecurity/wiki/Reference-Manual-%28v2.x%29#SSecConnWriteStateLimit. Accessed: 2018-01-20.

[14] Slowloris. https://www.incapsula.com/ddos/attack-glossary/slowloris.html. Accessed: 2018-01-19.

[15] Slowloris. https://www.cloudflare.com/learning/ddos/ddos-attack-tools/slowloris/. Accessed: 2018-01-19.

[16] Ut-loic. https://ris.utwente.nl/ws/portalfiles/portal/5095225/2010-12-CTIT-TR.pdf. Accessed: 2018-01-20.

[17] Verisign distribute denial of service trends report. https://www.a10networks.com/sites/default/files/a10-tps-eb-verisign-distributed\-denial-of-service-trends-report-vol-5\

\-issue-2.pdf. Accessed: 2018-01-19.

[18] L. C. Giralte, C. Conde, I. M. de Diego, and E. Cabello. Detecting denial of service by modelling web-server behaviour. *Computers & Electrical Engineering*.

[19] H. H. Jazi, H. Gonzalez, N. Stakhanova, and A. A. Ghorbani. Detecting http-based application layer dos attacks on web servers in the presence of sampling. 2017.

[20] S. A. Mohammed and A. M. Azizah. A novel protective framework for defeating http-based denial of service and distributed denial of service attacks. 2015.

[21] K. Singha, P. Singha, and K. Kumarb. Application layer http-get flood ddos attacks: Research landscape and challenges. 2016.

[22] J. Yu, C. Fang, L. Lu, and Z. Li. A lightweight mechanism to mitigate application layer ddos attacks. In *Scalable Information Systems*, 2009.