Detection and Analysis of attacks targeting a web server using Splunk

Dyaa Tum, 11924899, Mohammed Awawdeh, 11823092, Abdulkareem Sulieman, 11822972.

***Abstract*—This research focuses on the potential threats to a website, such as DDOS attacks, SQL injection, and buffer overflow. In this project, we created a complete website with a database and tested some common risks. We used the Splunk platform to do data analysis on the data generated by the attacks in order to quickly identify attacks and predict possible threats. The results of the analysis were utilized to determine how to respond to these attacks and propose solutions.**

***Index Terms*—Splunk, DDOS, Buffer Overflow, SQL Injection.**

I. Introduction

WEBSITES are hosted on web servers. Web servers are themselves computers running an operating system; connected to the back-end database, running various applications. Any vulnerability in the applications, Database, Operating system or in the network will lead to an attack on the web server. Some of the threats that could are: Denial-of-Service (DOS) attack, SQL Injection, Buffer Overflow.

We developed a Webserver and used these attacks to target it. And we gathered all of the data generated by this attack, and we automatically sent the gathered data to Splunk using Splunk forwarder, then Splunk analyzed these attacks, and we configured the setting to allow Splunk to send an alert when some of these attacks are detected and determine whether it is a buffer overflow, SQL injection, or DOS attack. Following the identification of the attack, the generation of an alert to the user, and the capture of the attack vector. We developed a countermeasure to the detected attack, ensuring that it does not occur again.

The purpose of Splunk engine collection is to deal with big data generated by these machine language, and made graphical reports with the data. We can observe the characteristics of DDoS attack with data visualization and use Splunk platform to do data analysis for determine the attacks quickly and predict possible hidden crisis.

II. SQL Injection

SQL Injection is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior.

A successful SQL injection attack can result in unauthorized access to sensitive data, such as passwords, credit card details, or personal user information. Many high-profile data breaches in recent years have been the result of SQL injection attacks, leading to reputational damage and regulatory fines. In some cases, an attacker can obtain a persistent backdoor into an organization's systems, leading to a long-term compromise that can go unnoticed for an extended period.

SQL injection prevention techniques

Since user input channels are the primary vector for such assaults, the ideal method is to control and vet user input to look for attack patterns. Developers can also avoid vulnerabilities by implementing the key preventative strategies listed below.

Input Validation The validation process verifies whether or not the type of input given by a user is permitted. Input validation ensures that the type, length, format, and so on are correct. Only values that pass the validation are handled. It helps in the suppression of any commands entered into the input string. In some ways, it is analogous to checking to see who is knocking before opening the door.

Parametrized queries are a method of pre-compiling a SQL statement such that the parameters can be supplied before the statement is performed. This method allows the database to detect the code and distinguish it from the input data. Because the user input is automatically quoted and the given input does not affect the intent, this coding style aids in the prevention of a SQL injection attack.

Signs of SQL Injection (SQLI)

Detecting SQLI attacks can be done by monitoring the user input, an attacker can attempt to perform SQLI attack depending on the form's input fields, he can insert a SQL command in the input field, and if the web server is not well designed, this will cause to insert this command into the database which will retrieve the desired data, so if we noticed and input like **sometext' OR 1=1** this would be a clear sign of a SQLI attack, this input can retrieve all the data inside the database if it gets implemented and not filtered.

This is how will the PHP reads this input: If the sql query is like this for example:
"SELECT * FROM student WHERE id='ID'";
The result will be:

"SELECT * FROM student WHERE id='sometext' OR 1=1";
This statement will always be true since it contains (OR 1=1) which will always lead to retrieve all the database content.

III. DOS Attack

Denial of service (DoS) attacks typically use hierarchical control network formed Botnet attacks and attacks. A DoS attack is designed to interrupt or shut down a network, service, or website, so that the victims were infected with the virus unknowingly and become part of the puppet. According to Akamai's third quarter of 2015 Internet Security Situation report, the DoS attacks grew 180% over the same quarter of 2014.The attack has not reduced but increased, therefore we need to establish good protective measures and analysis methods to reduce disasters and losses. DoS usually attack by huge amount of packets. In recent years, the concept of big ideas and technical data develop rapidly and become popular. In 2015, Big Data has become part of many industries. One of the purposes of the popularity of big data is to solve the problems of information on new patterns.

There are two general methods of DoS attacks: flooding services or crashing services. Flood attacks occur when the system receives too much traffic for the server to buffer, causing them to slow down and eventually stop.

Other DoS attacks simply exploit vulnerabilities that cause the target system or service to crash. In these attacks, input is sent that takes advantage of bugs in the target that subsequently crash or severely destabilize the system, so that it can't be accessed or used.

Signs of a DoS attack

The United States Computer Emergency Readiness Team, also known as US-CERT, provides guidelines to determine when a DoS attack may be in progress. According to US-CERT, the following may indicate an attack is underway:

1) Slower or otherwise degraded network performance that is particularly noticeable when trying to access a website or open files on the network.
2) Inability to access a website.
3) More spam email than usual.

Preventing a DoS attack

Network security is essential for stopping any DDoS attack attempt. As an attack only has an impact if a hacker has enough time to pile up requests, the ability to identify a DDoS early on is vital to controlling the blast radius.

You can rely on the following types of network security to protect your business from DDoS attempts:

1) Firewalls and intrusion detection systems that act as traffic-scanning barriers between networks.
2) Anti-virus and anti-malware software that detects and removes viruses and malware.
3) Endpoint security that ensures network endpoints (desktops, laptops, mobile devices, etc.) do not become an entry point for malicious activity.
4) Web security tools that remove web-based threats, block abnormal traffic, and search for known attack signatures.

IV. Buffer Overflow

Buffers are memory storage regions that keep data momentarily while it is transported from one location to another. A buffer overflow (or buffer overrun) occurs when the volume of data exceeds the memory buffer's storage capacity. As a result, the software attempting to write data to the buffer overwrites memory addresses next to it. For example, if a buffer for log-in credentials is configured to expect username and password inputs of 8 bytes, the software may write the surplus data across the buffer border if a transaction contains an input of 10 bytes (that is, 2 bytes more than intended).

Buffer overflows can occur in any sort of software. They are generally caused by faulty inputs or a failure to assign enough buffer space. If the transaction overwrites executable code, the software may perform erratically and produce inaccurate results. Errors with memory access or crashes.

Types of Buffer Overflow Attacks:

Stack-based buffer overflows are more common, and leverage stack memory that only exists during the execution time of a function.

Heap-based attacks are harder to carry out and involve flooding the memory space allocated for a program beyond memory used for current runtime operations.

Preventing a Buffer Overflow attack:

Developers can protect against buffer overflow vulnerabilities by implementing security mechanisms in their code or by utilizing languages with built-in protection. Furthermore, newer operating systems have runtime protection. Three common protections are:

• Address space randomization (ASLR) – randomly moves data region address space locations. Buffer overflow attacks often require knowledge of the location of executable code, which is made nearly impossible by randomizing address spaces.

• Data execution prevention—marks specific sections of memory as executable or non-executable, preventing an attack from running code in a non-executable zone.

V. Proposed system and the solutions used to face the mentioned attacks

We built a security system to mitigate these 3 attacks by using the best countermeasures available, and connecting the web server with and IDS (Splunk) to get accurate analyses about the server's overall status and detect any attempt for an attack, to generate alerts if something malicious occurs, we will clarify the techniques we used to built this system and reach the desired goals.

SQLI and the technique used to prevent it

In our website, we have used this query to retrieve data from database and display them to the user depending on the ID he inputs:

```php
// Performing retrieve query execution
$sql = "SELECT * FROM student WHERE id='$ID'";
$result = mysqli_query($conn,$sql);
```

Fig. 1. The query used to retrieve data.

This query selects every attribute of the table “student“ that the ID matches the input of the user, for example if the user inputs 11924899 then the query will retrieve the attributes for the ID 11924899 and display them. An example for this:

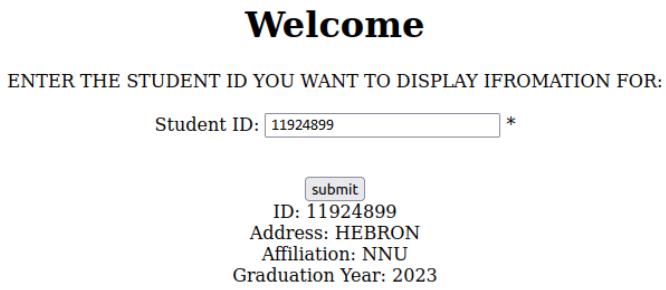

Fig. 2. The Website we built to retrieve user data by ID.

And this is how the SQL query receives the ID from the input field in the form:

```php
<?php
$ID="";
$IDErr="";
if ($_SERVER["REQUEST_METHOD"] == "POST") {
if (empty($_POST["ID"])) {
$IDErr = " ID should be entered ";
}
else {
$ID =$_POST["ID"];
}
}
```

Fig. 3. The PHP code that we wrote to receive the ID from the input field.

The $ID takes a value from $POST[“ID”] But we have a problem here, there are no data validation, which means that the form accepts every input without checking it, for example the user can input a SQL query command and this query will be executed by the server and the database without any problem.

A simple example for this is when the input is test‘ or 1=1, this input is a clear sign that the user is trying to do something illegal because and ID can never be in that form, and the query will process this input as “SELECT * FROM student WHERE id=’test” or 1=1” so when the user tries this input, the result will be like this:

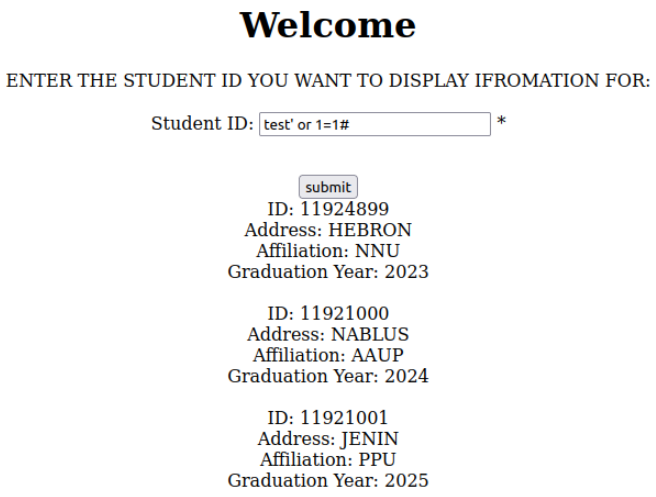

Fig. 4. Example on the vulnerability in our website, the query retrieved all the data in the database.

As we can see, the server retrieved and displayed all the information inside the database, because the input wasn‘t checked and validated and the **or 1=1** statement is always true which means this will be executed always if not validated.

To fix this we need to check and validate the input before it gets accepted and executed, and this was fixed by using the mysqli_real_escape_string, this PHP function escapes special characters in a string for use in an SQL statement, taking into account the current charset of the connection.

This is how the $ID will receive the value after using the mysqli_real_escape_string function:

```php
<?php
include 'db_connection.php';
$conn = OpenCon();
$ID="";
$IDErr="";
if ($_SERVER["REQUEST_METHOD"] == "POST") {
if (empty($_POST["ID"])) {
$IDErr = " ID should be entered ";
}
else {
$ID =mysqli_real_escape_string($conn,$_POST["ID"]);
}
}
```

Fig. 5. The function we used to prevent the attack.

So when the user tries the same input test‘ **or 1=1**, The server will take it as **test/ or 1=1**, and the results will be as shown in the Figure 6.

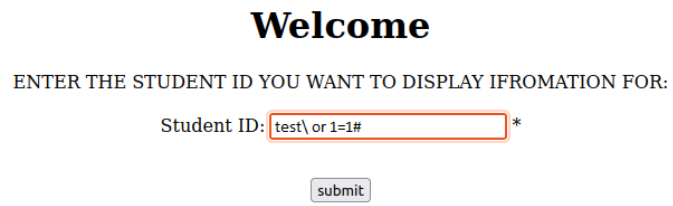

Fig. 6. After solving the vulnerability.

On Splunk we detect the attack using the query, and we set an alert if this query has been inserted in the input field again. As shown in the Figures 7,8 (At the end of the page).

DOS and the technique used to prevent it

We prevented DOS attacks by depending on UFW firewall, adding some rules and configuring it the way it makes DOS attacks useless. First we enabled UFW firewall and we allowed the traffic on port 80 (HTTP), then we configured the before.rules UFW file by adding these :

```
### Add those lines after *filter near the beginning of the file
:ufw-http - [0:0]
:ufw-http-logdrop - [0:0]


### Start HTTP ###

# Enter rule
-A ufw-before-input -p tcp --dport 80   -j ufw-http
-A ufw-before-input -p tcp --dport 443  -j ufw-http

# Limit connections per Class C
-A ufw-http -p tcp --syn -m connlimit --connlimit-above 50 --connlimit-mask 24 -j ufw-http-log

# Limit connections per IP
-A ufw-http -m state --state NEW -m recent --name conn_per_ip --set
-A ufw-http -m state --state NEW -m recent --name conn_per_ip --update --seconds 10 --hitcount

# Limit packets per IP
-A ufw-http -m recent --name pack_per_ip --set
-A ufw-http -m recent --name pack_per_ip --update --seconds 1  --hitcount 20  -j ufw-http-log

# Finally accept
-A ufw-http -j ACCEPT

# Log-A ufw-http-logdrop -m limit --limit 3/min --limit-burst 10 -j LOG --log-prefix "[UFW HT
-A ufw-http-logdrop -j DROP

### End HTTP ###
```

Fig. 9. The UFW rules added to prevent the DOS attack.

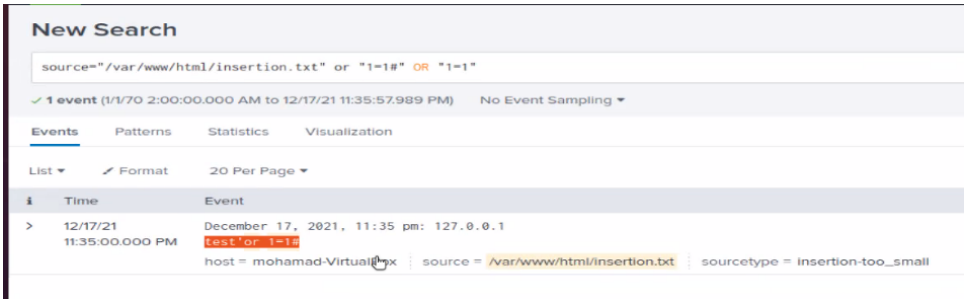

Fig. 7. Logs related to the SQL Injection attack .

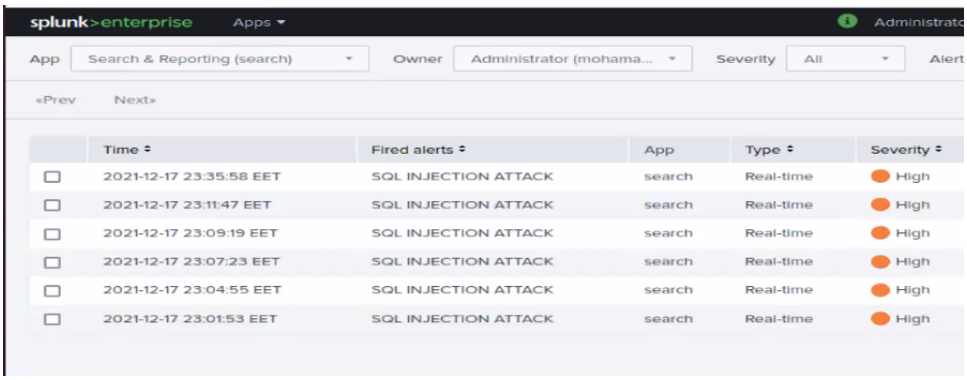

Fig. 8. The alerts generated by Splunk when the SQL Injection happened .

By adding these configurations to the before.rules file shown in figure 9, this results in:

Rate limiting: A rate limit of x connections per y seconds means that if x connections has been initiated in the last y seconds by this profile, it will be dropped. Dropping is actually a nice protection against flooding because the sender won't know that you dropped it. He might think the packet was lost, that the port is closed or even better, the server is overloaded. Imagine how nice, your attacker thinks he succeeded, but in fact you are up and running, him being blocked.

Connections per IP: A connection is an open channel. A typical browser will open around 5 connections per page load and they should last under 5 seconds each. Firefox, for example, has a default max of 15 connections per server and 256 total.

Connections per Class C: Same a above, but this time we apply the rule to the whole Class C of the IP because it is quite common for someone to have a bunch of available IPs. This means for example all IPs looking like 11.12.13.* We decided to go for 50 simultaneous connections.

Packets per IP:

This is the challenging part. Due to a limitation that is not easy to circumvent, it is only possible to keep track of the last 20 packets. At the same time, it might add a considerable overhead to track 100 packets for each IPs. While big website may eventually need more than this, like I said, you should take a look in a proper CDN.

We decided to go for 20 packets / second / IP.

After adding these to the before.rules file, we need to reload the UFW firewall by using sudo ufw reload so the new configurations are applied now.

For testing the DOS attacks on the server, we used LOIC tool to flood the server with requests, and this is how the results was before configuring the UFW firewall, in Figure 10.

192.168.1.182 is our server's IP address

We can see that there are a huge number of successful requests on the server which will bring the server down if it can't handle it, and we could go for a bigger number if we didn't stop the flooding.

And now when trying the same attack but after the new configurations, the results shown in Figure 11.

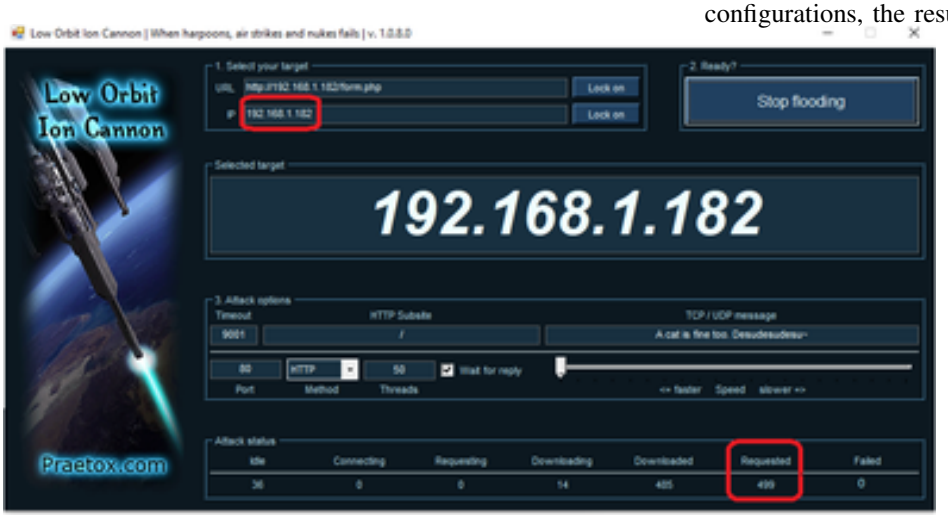

Fig. 10. Attacking our website using a tool called LOIC, and the Requested packets were 499.

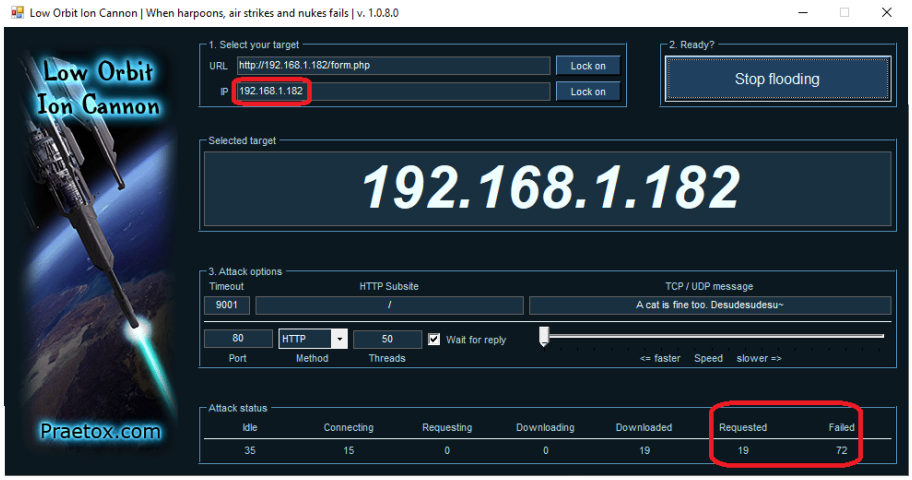

Fig. 11. Attacking our website after the new configuration.

As we can see in Figure 11, the successful requests were only 19 this time, because we configured it to be maximum 20 request for the same IP per second, all the other requests failed to access the server. By this way, we prevented the DOS attacks on our server and we tested it, the results were pretty good and satisfying.

And on Splunk we were sending all the logs for analysis, and Splunk generate a chart that shows how the number of packets increased, at a sertine of time. As shown in the figure 12.

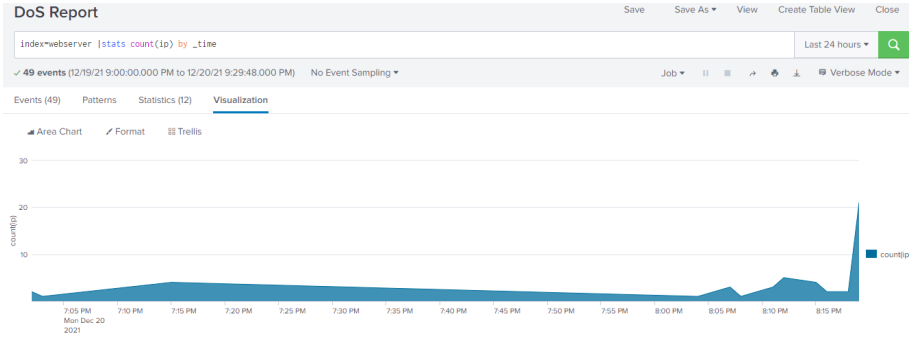

Fig. 12. Chart generated using Splunk for the DOS attack.

VI. Buffer overflow and the technique used to prevent it

We used a simple technique to prevent buffer overflow on the server, and that was specifying the max length for an input field, For example, the ID field can have a maximum of 8 elements as an input, it‘s not allowed to input more than 8, Same thing for all other input fields, so the attacker cant input more than the server can handle and process.

VII. Conclusion

There is many types of attacks targeting the web servers, in this study we choose three attacks that are considered from the top threats on the web servers. And what most security administrators suffers from in there organizations.

Data base is considered a valuable asset for the organizations, so the attacker always try to target it. And we simulate SQL Injection attack on our website that gits all the data from the data base, then we did the countermeasure to prevent this attack.

DOS is the another attack that we simulate on our website, we sent a huge number of packets in a short period of time, and the Splunk gave us alerts for this attack, at the end we provide the countermeasure for the dos attack, Then we made configuration so that the website does not receive more than 20 at the same time.

Chart analysis from Splunk was used to assist in reading the gathered data to help detect the possibility of attack, and data mining was used to discover hidden risks. By using chart analysis, more traffic flow characteristics could be used to differentiate normal operations from abnormal ones, such as changes in traffic wave forms, turning points in transitions, and abnormal transmission of packets.

VIII. References

1) Attack detection of distributed denial of service based on Splunk, Publisher: IEEE https://ieeexplore.ieee.org/document/7840355
2) https://www.ptsecurity.com
3) https://askubuntu.com/
4) https://www.imperva.com/
5) https://www.techtarget.com/
6) https://www.paloaltonetworks.com/