

Splunk Detection of Cyber Security Attacks on Web Server along with Solutions to Prevent them.

Mohammed Adnan¹, Asem Majed²

Abstract—Data monitoring and logging are critical missions made to networks and systems. This is also a key process implemented to servers in general since they help in the detection of security breaches. In this framework, Splunk tool is used for real-time monitoring of a web server logs looking for several attacks that are commonly targeting web servers. First, the web server environment is set up to simulate a database that stores information about students with a website that is accessed by users. Second, log files from the environment is sent to the Splunk platform to conduct analysis to detect the following: Buffer overflow, SQL injection, and Denial-of-Service (DoS) attacks. Using the key characteristics of each attack, Splunk have managed to detect these attacks and gave the proper alerts as required. Finally, different modifications on the code and the framework was made to tackle these attacks and prevent them from compromising the web server.

"Keywords": Splunk, SQL injection, Buffer overflow, DOS Attack.

I. INTRODUCTION

Websites are an attractive point to attackers in which they conduct many different types of attacks through exploiting existing vulnerabilities in the web server environment, this include the website itself, running web server and its related services and the operating system. There must be strong protection to web servers since they are usually accessible from remote users and may hold confidential information about the home network or the users that visit the server.

Web server vulnerabilities are continually changing as cybercriminals continue to seek new ways to exploit your security vulnerabilities and web application security issues. For your organization to be able to properly protect itself against web server vulnerabilities, it's essential to stay up-to-date with common web server vulnerabilities, web browser vulnerabilities, and stay versed on all web application security issues and solutions.

The Web Server is exposed to threats which an attacker can use many common techniques to compromise a web server such as Buffer overflow, SQL injection, and Denial-of-Service (DoS) attacks. All these attacks can lead to catastrophic consequences, hence, must be addressed using the proper mechanisms.

In order to prevent/mitigate an attack, first it must be detected, and here where Splunk comes in place. Splunk is used for monitoring and searching through big data. It indexes and correlates information in an index that makes it searchable, and makes it possible to generate alerts, reports and visualizations. Splunk is fed with log files generated from the web server environment (Apache, PHP, etc). These log files are used for further analysis to obtain unique

characteristics that distinguish different attacks as will be shown.

The following section will review previous work about the aforementioned attacks and how to tackle them. The next section will be an overview on the system and suggested solutions, then results of Splunk charts & alerts will be exhibited in figures. Finally a conclusion stating the results and future work is mentioned.

II. PREVIOUS WORK ABOUT THE AFOREMENTIONED ATTACKS AND HOW TO TACKLE THEM

A. Buffer Overflow

Buffer overflow is also known as Buffer overrun, is a state of the computer where an application tries to store more data in the buffer memory than the size of the memory. This leads to data being stored into adjacent storage, which may sometimes overwrite the existing data, causing potential data loss and sometimes a system crash as well. It is a common programming mistake that most developers commit unknowingly. Hackers are most often exploiting this to gain access to unsolicited data. Now that a vulnerability has been identified with the computers, hackers are bound to exploit it and try to attack various systems through buffer overflow attacks. Now the question arises, how does a hacker execute such an attack, and what are the consequences?

In a buffer overflow attack, the extra data includes instructions that are intended to trigger damaging activities such as corrupting files, changing data, sending private information across the internet, etc. An attacker would simply take advantage of any program which is waiting for certain user input and inject surplus data into the buffer.

As developers, it is our responsibility to check for buffer overflows in our code. If buffer overflows are handled in the code itself, the security of the system is not hampered through buffer-overflow attacks. By using limitation for input fields and applying different types of input validation and so on.

B. DOS Attack

Dos attack (Denial-of-Service) is a cyber attack in which an attacker sends lots of requests (traffic) simultaneously at a time to the target server or network, due to which it become overload and the services of that server or network goes down temporarily. Targeted server and its services are not available till the impact of the attack disappears.

- Some Big & Famous Attacks :-

- The February 2018 GitHub DDoS attack :- Prior to this 2.3 Tbps attack, the largest verifiable DDoS

attack on record targeted GitHub, a popular on-line code management service used by millions of developers. This attack reached 1.3 Tbps, sending packets at a rate of 126.9 million per second. As shown in figure 1

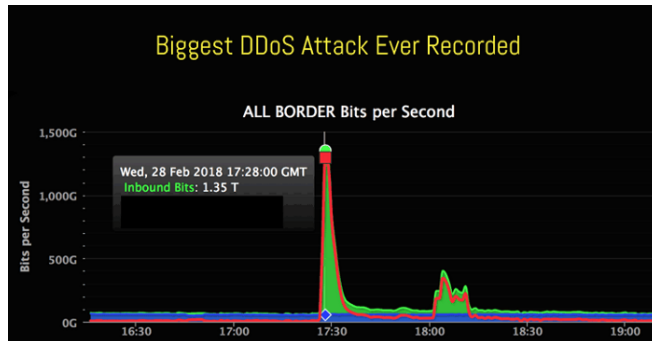


Fig. 1. Biggest DDOS attack

Unfortunately, there is no 100% successful ways to protect a victim from falling under the prey of malicious DoS/DDoS attackers. However, users can apply some prevention tactics to reduce the likelihood of an attacker to use a computer to attack other computers. These prevention tactics are: 1- Install Antivirus software with the latest updates. 2- Install a firewall and try to configure it with the most recent updates to restrict traffic. 3- Apply filtering of emails to manage unwanted traffic.

C. SQL Injection

SQL injection is an attack technique that exploits a security vulnerability occurring in the database layer of an application. Hackers use injections to obtain unauthorized access to the underlying data, structure, and DBMS. By an SQL injection an attacker can embed a malicious code in a poorly-designed application and then passed to the back-end database. The malicious data then produces database query results or actions that should never have been executed. By using an SQL Injection vulnerability, given the right circumstances, an attacker can use it to bypass a web application's authentication and authorization mechanisms and retrieve the contents of an entire database. SQL Injection can also be used to add, modify and delete records in a database, affecting data integrity. To such an extent, SQL Injection can provide an attacker with unauthorized access to sensitive data. To prevent the SQL Injection we can validate user-supplied input for expected data types, including input fields like drop-down menus or radio buttons, not just fields that allow users to type in input.

III. RESULTS THAT SHOW SPLUNK CHARTS WHEN AN ATTACK EXISTS

A. Splunk Monitoring

Log files were uploaded to Splunk to be continuously monitored in which events were extracted from the log files and then indexed automatically so it can detect attacks

attempts it real time. Figure 2 shows the most important log file which is access.log that is created by Apache server to record all activities occurring on the website. This file (along with php_error) will be used for analysis in order to detect the previously mentioned attacks.

Fig. 2. Adding data to Splunk

B. SQL Attack & Detection

For this attack, we need to enter a malicious query to retrieve all the information related to students from the database. Normally, for our website, you can retrieve only one row of data related an ID, which is the primary (unique) key. In this attack, we enter a malicious query in ID input which is "1 or 1" which means "True" for all database entries, so it will retrieve all rows from the database as shown in figure 3.

ID	Name	affiliation	Address
1111111	mohammed	CIS	nablus
11820208	Asem	NIS	KSA
11820745	mohammed	CIS	nablus
11821976	mohammed	NIS	nablus

Student ID	1 or 1
Student Name	
Student affiliation	
Student address	
<input type="button" value="insert"/> <input type="button" value="delete"/> <input type="button" value="get"/>	

Fig. 3. SQL injection attack

Then, we try to detect this attack using Splunk by searching about this malicious query (1 or 1), so as figure 4 shown the search statement to detect this attack. After that we save it as an alert so if this query occurred again, Splunk will generate an alert for that event as shown in figure 5

C. DOS Attack & Detection

For this attack, we used a tool that generates huge requests (traffic) simultaneously at a time to the target server. After the attack is performed, our web page will not open as shown in figures 6 & 7. For Splunk detection, we look for a large number of request that is larger than 300 request per second as shown in figure 8, also figure 9 shows charts when the traffic is normal before dos attack occur. Then we generate the alert so that if the attack occurs, the alert will be generated as shown in figure 10.

Fig. 4. SQL injection search in splunk

Fig. 5. SQL injection alert

Fig. 6. DOS attack

Fig. 7. DOS attack command

Fig. 8. DOS search in Splunk

Fig. 9. DOS charts

Fig. 10. DOS alert

D. Buffer Overflow attack & Detection

This attack happens when a user enters data that exceeds the maximum capacity of some field, leading to an inconsistent response from the server that may reveal unauthorized data to the user. We have two cases for our framework: insert mode, and select mode.

In the first mode, the attacker insert data with larger length than the one that is predefined in the database, this will result in error that is returned from the PHP code as shown in figures 11 & 12.

Fig. 11. Buffer overflow attempt 1

Fig. 12. Buffer overflow attempt 2

The errors are logged in the `php_error` file (Figure 13).

Fig. 13. php_error log file

[illegible]

The message is logged in the Apache access log file (Figure 15).

Fig. 15. Apache access log file

The screenshot shows the Splunk Search interface. At the top, there's a navigation bar with 'splunk enterprise' logo, 'Apps', and user roles like 'Administrator', 'Messages', 'Settings', 'Activity', 'Help', and 'Feed'. Below this is a search bar with 'Search' and a 'Search & Reporting' button. The main section is titled 'New Search' and contains a search query: `(sourcetype=access_combined status=404) OR (sourcetype=shs_error "Out of range value" OR "data too long")`. Below the query, it shows '38 events (12/19/21 1:00:00:00 PM to 12/19/21 1:06:42:000 PM)' and 'No Event Sampling'. There are tabs for 'Events (38)', 'Patterns', 'Statistics', and 'Visualization'. The 'Events' tab is active, showing a 'Format Timeline' view with a 'Zoom Out' button and a 'Dismiss' button. A timeline visualization is shown at the bottom, with a green bar representing the data range. The timeline is labeled '1 hour per column'.

The screenshot shows the D3.js visualization interface. At the top, there is a search input with the text "(sourceType=access_combined status=414) (sourceType='file_error' 'Out of range value' 'Data too long') | eval size=ln(size) | stats value(size) by size". Below the search input, there is a status bar showing "61 events (before 12/13/21 12:24:38,000 PM)" and "No Event Sampling". The main visualization area displays a bar chart with a single bar representing the value 8238. The y-axis is labeled "count" and ranges from 0 to 8238. The x-axis is labeled "size" and ranges from 0 to 8,500. The chart is titled "Bar Chart" and "Format: BI Thelis".

An alert was created based on the previous search command in which it will inform the administrator that an attack attempt is happening at real time 18.

1	Title *	Actions	Owner 1	App 2	Sharing 3	Status 3
+	<div> <div>Buffer_Overflow</div> <div> <div>Enabled: <input type="checkbox"/> Yes, <input checked="" type="checkbox"/> Disable</div> <div>Permissions: <input type="checkbox"/> Private, <input checked="" type="checkbox"/> Owned by assem2000, <input type="checkbox"/> Edit</div> <div>Modified: <input type="checkbox"/> Dec 16, 2021 12:33:38 AM</div> <div>Alert Type: <input type="checkbox"/> Real-time, <input checked="" type="checkbox"/> Edit</div> <div>Trigger Condition: <input type="checkbox"/> Per-Result, <input checked="" type="checkbox"/> <input type="button" value="Edit"/></div> <div> <div>Actions: <input type="checkbox"/> +1 Action</div> <div><input type="button" value="Add"/> Add to Triggered Alerts</div> </div> </div> </div>	<div>Open in Search</div> <div>Edit ▾</div>	assem2000	search	Private	Enabled

App	Search & Reporting [search]	Owner	Administrator (asm2000)	Severity	All	Alert	All	Filter	
+New	Nests								Showing 1-12 of 12 results
	Time		Fired alerts	App	Type	Severity	Mode	Actions	
<input type="checkbox"/>	2021-12-10 13:12:27 West Bank Gaza Standard Time	Buffer_Overflow	search	Real-time	High	Per Result	View results	Edit search	Delete
<input type="checkbox"/>	2021-12-10 17:04 West Bank Gaza Standard Time	Buffer_Overflow	search	Real-time	High	Per Result	View results	Edit search	Delete
<input type="checkbox"/>	2021-12-10 17:04 West Bank Gaza Standard Time	Buffer_Overflow	search	Real-time	High	Per Result	View results	Edit search	Delete
<input type="checkbox"/>	2021-12-10 19:43 West Bank Gaza Standard Time	Buffer_Overflow	search	Real-time	High	Per Result	View results	Edit search	Delete
<input type="checkbox"/>	2021-12-10 19:43 West Bank Gaza Standard Time	Buffer_Overflow	search	Real-time	High	Per Result	View results	Edit search	Delete
<input type="checkbox"/>	2021-12-10 19:43 West Bank Gaza Standard Time	Buffer_Overflow	search	Real-time	High	Per Result	View results	Edit search	Delete

✓ 1 event (11/70 2:00:00.000 AM to 12/10/21 12:34:52.590 AM)
No Event Sampling ▼

Events
Patterns
Statistics
Visualization

List ▼
✎ Format
20 Per Page ▼

i	Time	Event
>	12/10/21 12:34:51.000 AM	[09-Dec-2021 22:34:51 UTC] Data too long for column 'address' at row 1 host = HP source = C:\wamp64\logs\php_error.log sourcetype = php_error

IV. PROPOSED SYSTEM AND SOLUTIONS

For DoS attack, the most efficient solution is to prevent it by using a firewall or to limit the number of packets per second using a firewall. For this framework, we used limitation solution to prevent any possible DoS attack targeting the website using Linux UFW firewall. This is done by modifying the configuration file `/etc/UFW/before.rule` and add these line as shown in figure 21 to limit the connections, after that, we perform the attack to check if the alert will generate or not, as shown in figure 22 that there's no any alert.

Fig. 21. UFW limitation rule

Fig. 22. Splunk alert

B. SQL Injection Solution

Input validation is a key solution that is used to prevent this type of attack, this involve techniques implemented in the website code to validate the type and contents of input fields. The first method is done by changing the 'type' attribute in the HTML input tags to accept numbers only as shown in figure 23. However, this is not sufficient since an attacker

```
<table border="1">
<tr><td>Student ID</td><td><input type="number" name="studentID" value="1 or 1"></td></tr>
<tr><td>Student Name</td><td><input type="text" name="name" value="<?php echo $name;>"></td></tr>
<tr><td>Student Address</td><td><input type="text" name="address" value="<?php echo $address;>"></td></tr>
<tr><td>Affiliation</td><td><input type="text" name="affiliation" value="<?php echo $affiliation;>"></td></tr>
<tr><td>Graduation Year</td><td><input type="number" name="gradYear" value="<?php echo $gradYear;>"></td></tr>
</table>
```

Fig. 23. Type=number

may alter the URL to include its malicious SQL query. Therefore an additional code segment was added to the PHP code to validate datatype and content of the input. Figure 24 shows a code line that is used to filter out the data inside 'studentID' field which is the most important one because it's usually targeted for SQL injection attacks. Moreover, the code only allows specific data types and characters as shown in figures 25 and 26.

```
if($_SERVER["REQUEST_METHOD"]=="GET"){
    $conn = new mysqli("localhost","root","","students");
    $studentID = filter_var($_GET['studentID'],FILTER_SANITIZE_NUMBER_INT);
    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error);
    }
}
```

Fig. 24. Filtering the input

```
else if ($sub == "select") {
    if (!preg_match("/^[0-9]*$/", $studentID)) {
        $ErrMsg = "Only numeric value is allowed. <br>";
        echo $ErrMsg;
    } else {
        $sql = "select * from student where studentID=$studentID";
        $result = $conn->query($sql);
    }
}
```

Fig. 25. Checking the input (when select)

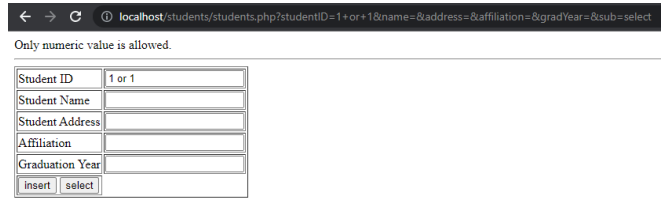
```
if($sub == "insert"){
    if (!preg_match("/^[a-zA-Z][0-9]*$/", $name $address $affiliation $gradYear)) {
        $ErrMsg = "Only alphabets and whitespace are allowed. <br>";
        echo $ErrMsg;
    } else {
        $sql = "insert into student(name, address, affiliation, gradYear)
        values('$name', '$address', '$affiliation', '$gradYear')";
        if ($conn->query($sql) === TRUE) {
            // $last_id = $conn->insert_id;
            echo "New record created successfully";
        }
    }
}
```

Fig. 26. Checking the input (when insert)

When a suspicious input is entered, the website refuse to continue and notify the user with a message as shown in figures 27 and 28.

C. Buffer overflow Solution

For this type of attack, the same approach that is used in SQL injection prevention is followed, that is input validation. The key solution is to check the length of the input to ensure

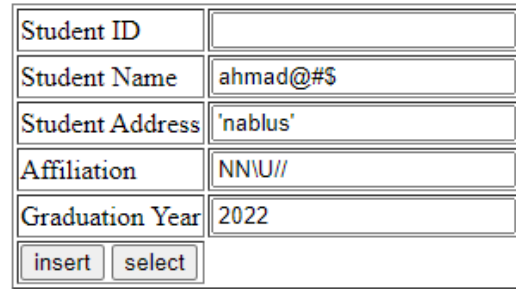


Only numeric value is allowed.

Student ID	1 or 1
Student Name	
Student Address	
Affiliation	
Graduation Year	
<input type="button" value="insert"/> <input type="button" value="select"/>	

Fig. 27. Attack attempt

Only alphabets and whitespace are allowed.



Student ID	
Student Name	ahmad@#\$
Student Address	'nabulus'
Affiliation	NN\U//
Graduation Year	2022
<input type="button" value="insert"/> <input type="button" value="select"/>	

Fig. 28. Suspicious input

it's below the maximum range of the predefined capacity in the database. Figure 29 shows the database structure to be used as a reference for the maximum length of some field. First, 'maxlength' attribute was added to each input field that

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	studentID	int(8)			No	None		AUTO_INCREMENT	Change Drop More
2	name	varchar(10)	latin1_swedish_ci		No	None			Change Drop More
3	address	varchar(10)	latin1_swedish_ci		No	None			Change Drop More
4	affiliation	varchar(10)	latin1_swedish_ci		No	None			Change Drop More
5	gradYear	int(5)			No	None			Change Drop More

Fig. 29. Database structure

prevents input that is longer than the designed (Figure 31). Again, there must be additional validation at the PHP code

```
<table border="1">
<tr><td>Student ID</td><td><input type="number" name="studentID" maxlength=8 value="<?php echo $studentID;>"></td></tr>
<tr><td>Student Name</td><td><input type="text" name="name" maxlength=10 value="<?php echo $name;>"></td></tr>
<tr><td>Student Address</td><td><input type="text" name="address" maxlength=10 value="<?php echo $address;>"></td></tr>
<tr><td>Affiliation</td><td><input type="text" name="affiliation" maxlength=10 value="<?php echo $affiliation;>"></td></tr>
<tr><td>Graduation Year</td><td><input type="number" name="gradYear" maxlength=5 value="<?php echo $gradYear;>"></td></tr>
</table>
```

Fig. 30. maxlength attribute

side. An if statement was added that prevents the execution of 'insert' query unless the field values are below the maximum length as indicated in figure 31.

```
if($sub == "insert"){
    if (!preg_match("/^[a-zA-Z][0-9]*$/", $name $address $affiliation $gradYear)) {
        $ErrMsg = "Only alphabets and whitespace are allowed. <br>";
        echo $ErrMsg;
    } elseif (strlen($studentID)>8 || strlen($name)>10 || strlen($address)>10 ||
        strlen($affiliation)>10 || strlen($gradYear)>5) {
        $ErrMsg = "Only alphabets and whitespace are allowed. <br>";
        echo $ErrMsg;
    } else {
        $sql = "insert into student(name, address, affiliation, gradYear)
        values('$name', '$address', '$affiliation', '$gradYear')";
        if ($conn->query($sql) === TRUE) {
            // $last_id = $conn->insert_id;
            echo "New record created successfully";
        }
    }
}
```

Fig. 31. Checking the input length

When attempting to insert too long values, the website returns an alert message as shown in figure 32.

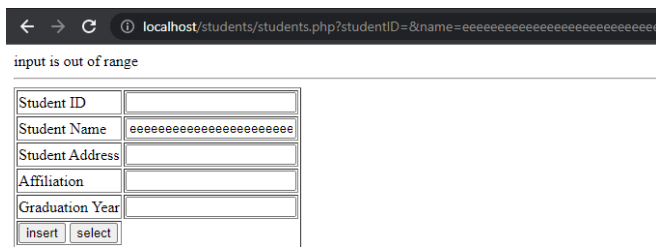


Fig. 32. Buffer overflow attempt

V. CONCLUSIONS

These attacks are the most common attacks that target web servers. Because of their destructive strength and the problems it cause to the systems such as crashing the system or retrieving sensitive information from the system. We used Splunk to detect these attacks as fast as possible dynamically by monitoring log files. Splunk helps us, also by using visualization to understand the nature of the attacks, when they happen, the attack vector, and to check if the traffic is abnormal or not.

After understanding these attacks and detecting them, we could prevent them using the different solutions that included using UFW firewall rules for DoS attack and using input validation for the input to prevent buffer overflow and SQL injection attacks. After implementing these mechanisms, Splunk results showed that the attacks are no longer shown in the triggered alerts while it's real-time monitoring and the prevention solutions worked as expected.

REFERENCES

- [1] <https://medium.com/@charithra/introduction-to-sql-injections-8c806537cf5d>
- [2] <https://www.w3schools.in/ethical-hacking/dos-attacks-and-its-prevention/>
- [3] <https://systemweakness.com/dos-ddos-attack-df5aedb66c19>
- [4] <https://www.educba.com/what-is-buffer-overflow/>
- [5] <https://info-savvy.com/web-server-attacks/>
- [6] <https://www.techfunnel.com/information-technology/web-server-vulnerabilities-attacks-how-to-protect-your-organization/>
- [7] <https://www.javatpoint.com/form-validation-in-php>
- [8] https://www.w3schools.com/html/html_form_attributes.asp