

# CSYE 6225

## Spring 2019

### Penetration Testing Report

By:  
Shubhankar Dandekar  
Jayesh Iyer  
Mitali Salvi  
Neha Gaikwad

#### Application Firewall:

A web application firewall (WAF) is an application firewall for HTTP applications. It applies a set of rules to an HTTP conversation. Generally, these rules cover common attacks such as cross-site scripting (XSS) and SQL injection.

While proxies generally protect clients, WAFs protect servers. A WAF is deployed to protect a specific web application or set of web applications. A WAF can be considered a reverse proxy. WAFs may come in the form of an appliance, server plugin, or filter, and may be customized to an application. The effort to perform this customization can be significant and needs to be maintained as the application is modified.

#### Assignment Objective:

1. Deploy AWS WAF to the Application Load Balancer (ALB) that fronts your web servers running on EC2.
2. Use AWS WAF to Mitigate OWASP's Top 10 Web Application Vulnerabilities.
3. All WAF resources and web security rules should be added to your application cloudformation stack.

Identify and test your application against at least 3 attack vectors that do not exploit UI vulnerabilities.

### 1) A7 – Insufficient Attack Protection

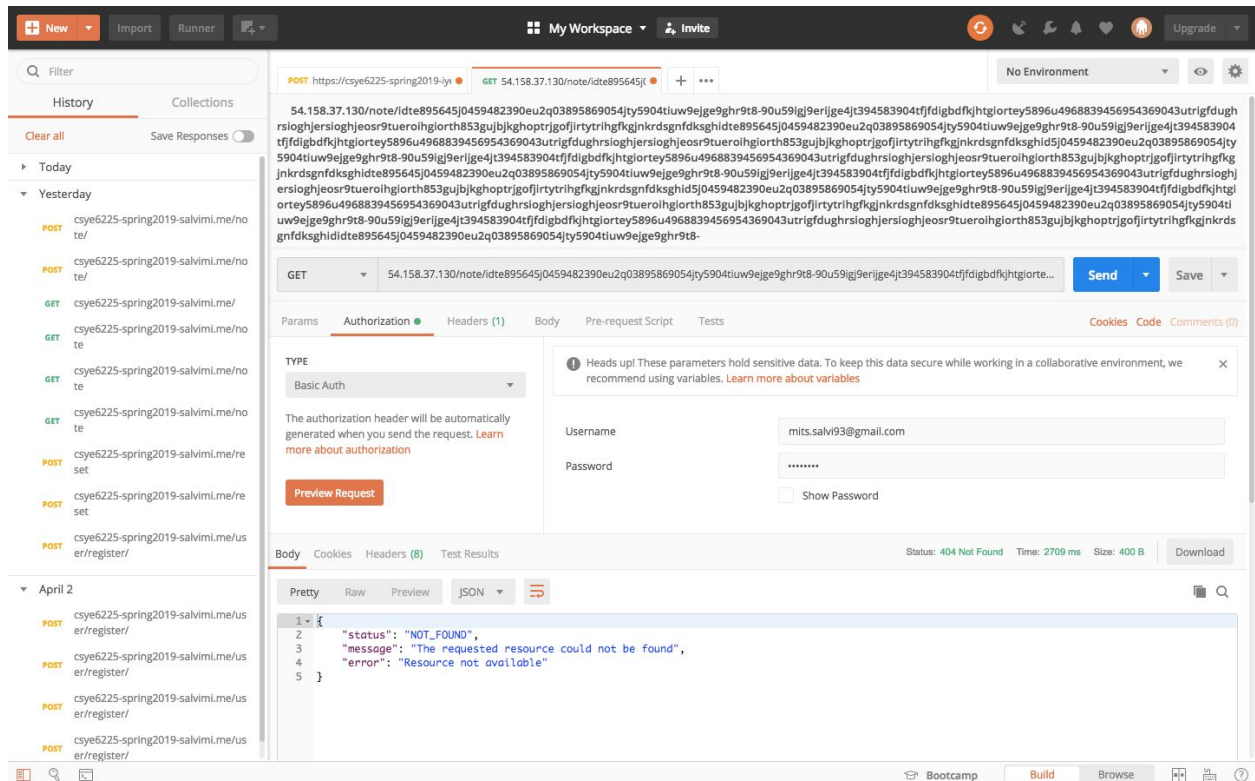
Use AWS WAF to enforce a level of hygiene for inbound HTTP requests. Size constraint conditions help you build rules that ensure that components of HTTP requests fall within specifically defined ranges. You can use them to avoid processing abnormal requests. An example is to limit the size of URIs or query strings to values that make sense to the application.

Also, you can use them to require the presence of specific headers, such as an API key for a RESTful API

Extending the URI length greater than 512 bytes:

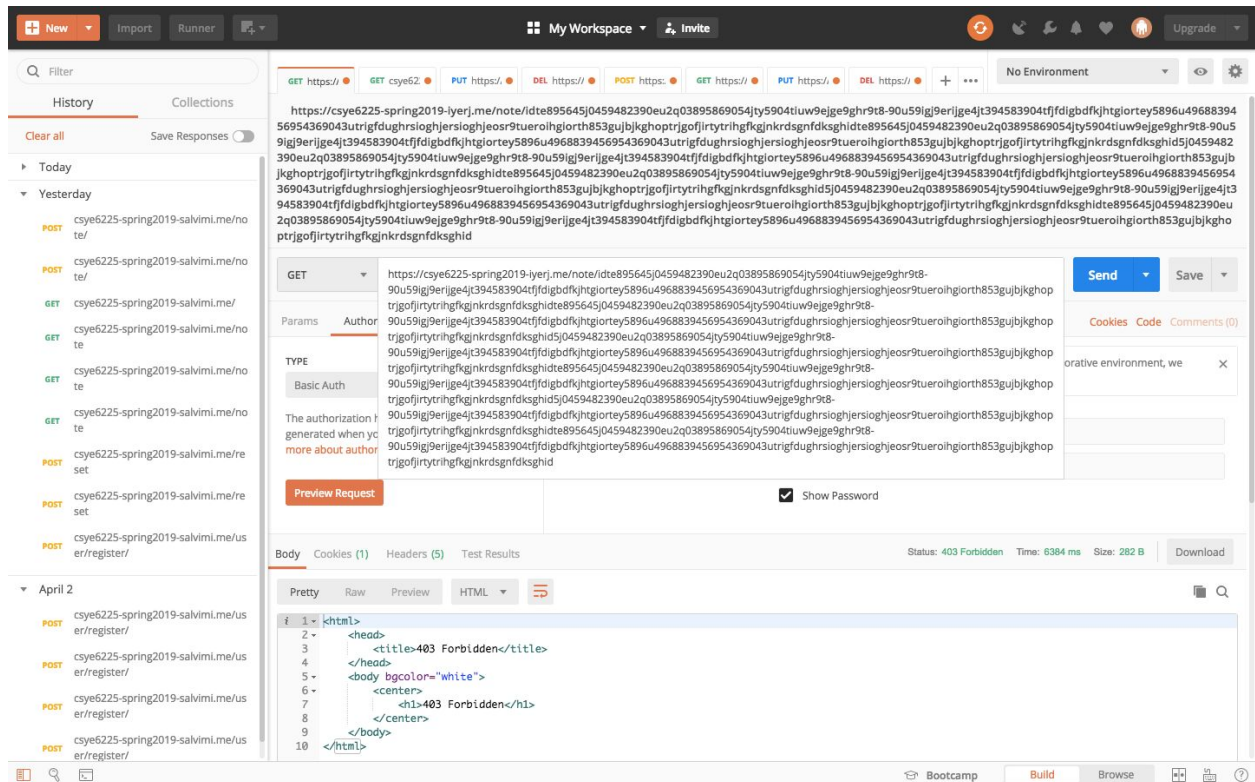
### WITHOUT USING WAF:

Without WAF the request is able to reach the controller and database.



### WITH USING WAF:

Using WAF the request was blocked by the firewall itself.



Extending the Body length greater than 52096 bytes:

### WITHOUT USING WAF:

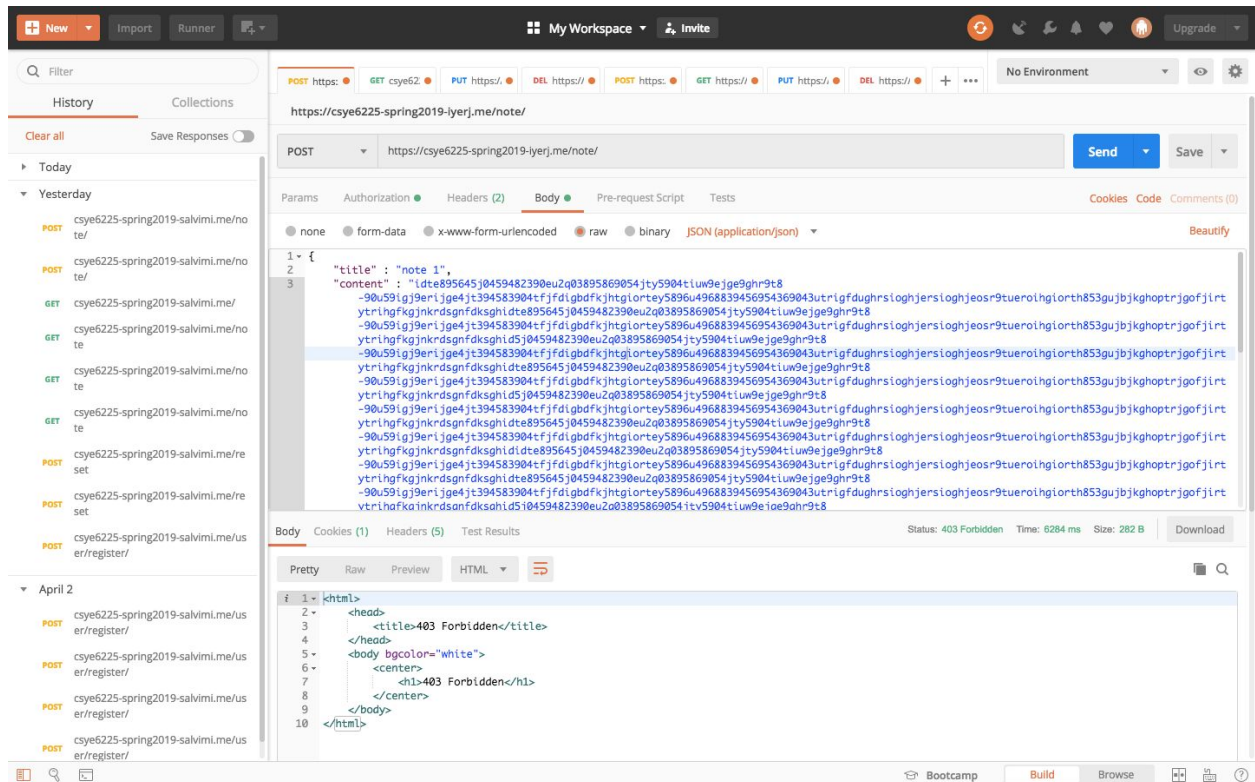
Without WAF the request is able to reach the controller and database.

The screenshot shows a REST client interface with the following components:

- Left Sidebar:** A history of requests. The top section is labeled 'Today' and contains several POST requests to 'https://csye6225-spring2019-salvimi.me/note/'. Below this is a section for 'Yesterday' and then 'April 2', each containing more requests.
- Main Area:** The top part shows the request details: 'POST https://csye6225-spring2019-salvimi.me/note/'. Below this is a tabbed interface with 'Body' selected. The body is a JSON object with a 'title' and a 'content' field. The 'content' field contains a long, base64-encoded string.
- Right Area:** The top part shows the response status: '500 Internal Server Error'. Below this is a tabbed interface with 'Pretty' selected. The pretty-printed response is a JSON object with a 'timestamp', 'status', 'error', 'exception', 'message', and 'path' field.

## WITH USING WAF:

Using WAF the request was blocked by the firewall itself.



Why this attack vector was chosen ?

Malicious actors are able to adapt their tool-sets quickly to exploit new vulnerabilities and launch large-scale automated attacks to detect vulnerable systems. It focuses strongly on the ability to react in a timely manner to new attack vectors and abnormal request patterns, or to application flaws that are discovered.

## 2) A1 – SQL Injection

SQL Injection (SQLi) is a type of an injection attack that makes it possible to execute malicious SQL statements. These statements control a database server behind a web application.

Attackers can use SQL Injection vulnerabilities to bypass application security measures. They can go around authentication and authorization of a web page or web application and retrieve the content of the entire SQL database. They can also use SQL Injection to add, modify, and delete records in the database.

An SQL Injection vulnerability may affect any website or web application that uses an SQL database such as MySQL, Oracle, SQL Server, or others. Criminals may use it to gain unauthorized access to your sensitive data: customer information, personal data, trade secrets, intellectual property, and more. SQL Injection attacks are one of the oldest, most prevalent, and most dangerous web application vulnerabilities.

## WITHOUT USING WAF:

High (Medium)	SQL Injection
Description	SQL injection may be possible.
URL	https://csye6225-spring2019-lyerj.me/
Method	GET
Parameter	Postman-Token
Attack	b8ade40f-3920-4322-b13a-27d75fb5bb04 OR 1=1 --
URL	https://csye6225-spring2019-lyerj.me/?query=query+AND+1%3D1++++
Method	GET
Parameter	query
Attack	query AND 1=1 --
URL	https://csye6225-spring2019-lyerj.me/note/5acd71e9-a8d3-45cb-a269-e23b7a3999f7/attachments?query=query+AND+1%3D1++++
Method	GET
Parameter	query
Attack	query AND 1=1 --
URL	https://csye6225-spring2019-lyerj.me/
Method	GET
Parameter	User-Agent
Attack	PostmanRuntime/7.6.1 AND 1=1 --
Instances	4
Solution	<p>Do not trust client side input, even if there is client side validation in place.</p> <p>In general, type check all data on the server side.</p> <p>If the application uses JDBC, use PreparedStatement or CallableStatement, with parameters passed by '?'</p> <p>If the application uses ASP, use ADO Command Objects with strong type checking and parameterized queries.</p> <p>If database Stored Procedures can be used, use them.</p> <p>Do not concatenate strings into queries in the stored procedure, or use 'exec', 'exec immediate', or equivalent functionality!</p> <p>Do not create dynamic SQL queries using simple string concatenation.</p> <p>Escape all data received from the client.</p> <p>Apply a 'whitelist' of allowed characters, or a 'blacklist' of disallowed characters in user input.</p> <p>Apply the principle of least privilege by using the least privileged database user possible.</p>

## OWASP ZAP REPORT

### USING WAF:

SQL Injection can be performed but we couldn't replicate those scenarios

Why this attack vector was chosen ?

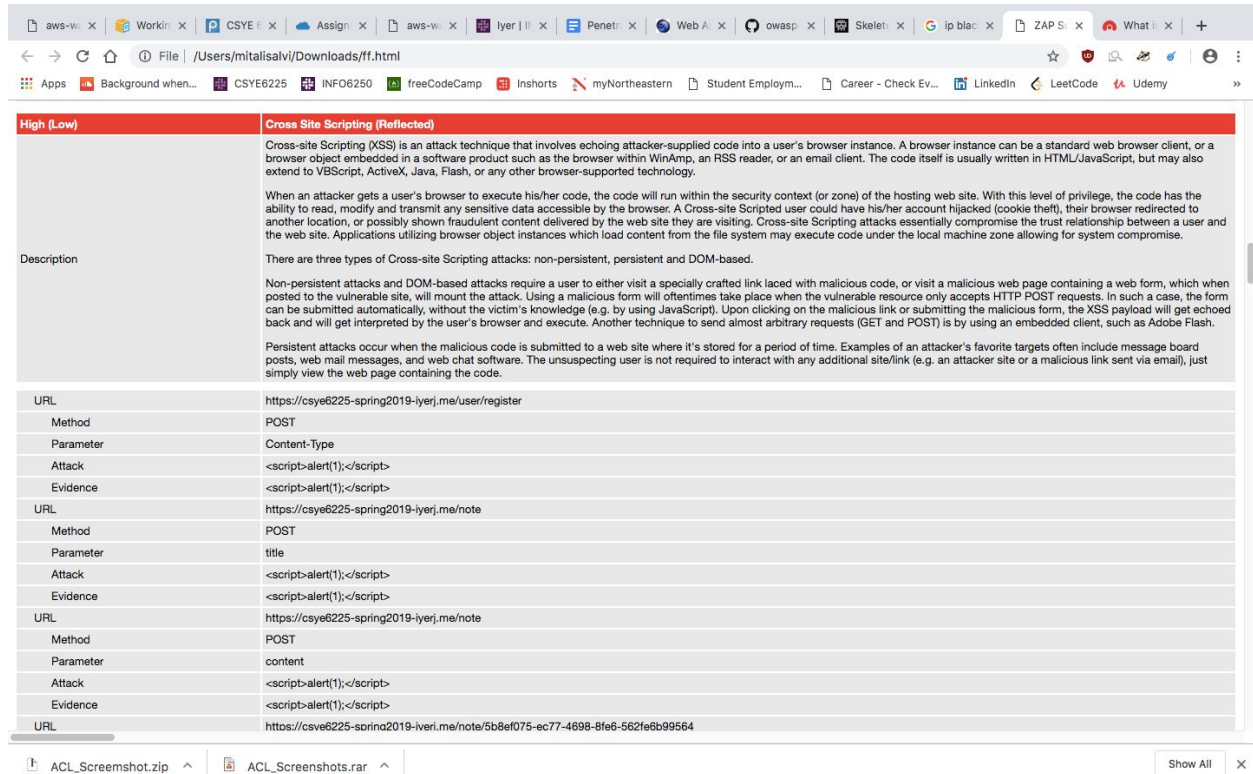
Attackers use almost any source of data as an injection vector, environment variables, parameters, external and internal web services, and all types of users and these vulnerabilities are very often in SQL. Injection can result in data loss or denial of access.

## 3)A8 – Cross-Site Request Forgery (CSRF)

Cross-site request forgery attacks predominantly target state-changing functions in your web applications.<sup>40</sup> Consider any URL path and HTTP request that is intended to cause a state change (for example, form submission requests). Are there any mechanisms in place to ensure the user intended to take that action? Without such mechanisms, there isn't an effective way to determine whether the request is legitimate and wasn't forged by a malicious party. Depending solely on client-side attributes, such as session tokens or source IP addresses, isn't an effective strategy because malicious actors can manipulate and replicate these values. CSRF attacks



take advantage of the fact that all details of a particular action are predictable (form fields, query string parameters). Attacks are carried out in a way that takes advantage of other vulnerabilities, such as cross-site scripting or file inclusion—so users aren't aware that the malicious action is triggered using their credentials and active session.



High (Low)	Cross Site Scripting (Reflected)
Description	<p>Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.</p> <p>When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise.</p> <p>There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based.</p> <p>Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash.</p> <p>Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code.</p>
URL	https://csye6225-spring2019-lyerj.me/user/register
Method	POST
Parameter	Content-Type
Attack	<script>alert(1);</script>
Evidence	<script>alert(1);</script>
URL	https://csye6225-spring2019-lyerj.me/note
Method	POST
Parameter	title
Attack	<script>alert(1);</script>
Evidence	<script>alert(1);</script>
URL	https://csye6225-spring2019-lyerj.me/note
Method	POST
Parameter	content
Attack	<script>alert(1);</script>
Evidence	<script>alert(1);</script>
URL	https://csye6225-spring2019-lyerj.me/note/5b8ef075-ec77-4698-8fe6-562fe8b99564

Why this attack vector was chosen ?

Because the web app uses JSON data as input and malicious scripts can be injected using the param in the URI

The web application handles this attack by forming an object of the input parameter and thus disabling the execution of the script. But due to a flaw in the spring security module, this attack cant be totally prevented

## 4) A2 – Broken Authentication and Session Management

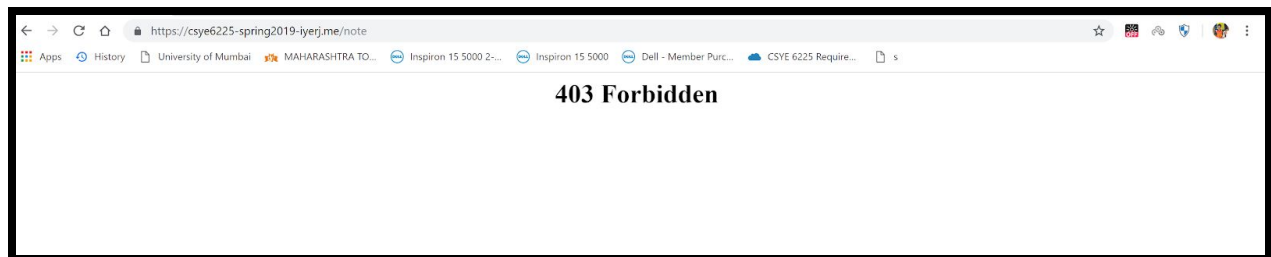
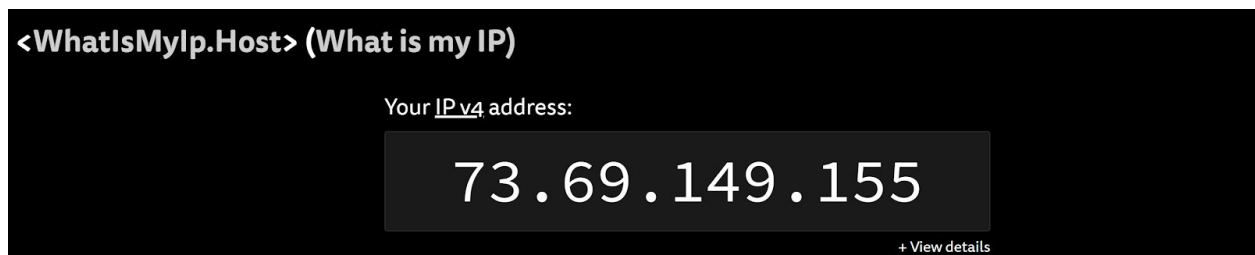
Flaws in the implementation of authentication and session management mechanisms for web applications can lead to exposure of unwanted data, stolen credentials or sessions, and

impersonation of legitimate users.<sup>9</sup> These flaws are difficult to mitigate using a WAF. Broadly, attackers rely on vulnerabilities in the way client-server communication is implemented. Or they target how session or authorization tokens are generated, stored, transferred, reused, timed-out, or invalidated by your application to obtain these credentials. After they obtain credentials, attackers impersonate legitimate users and make requests to your web applications using those tokens.

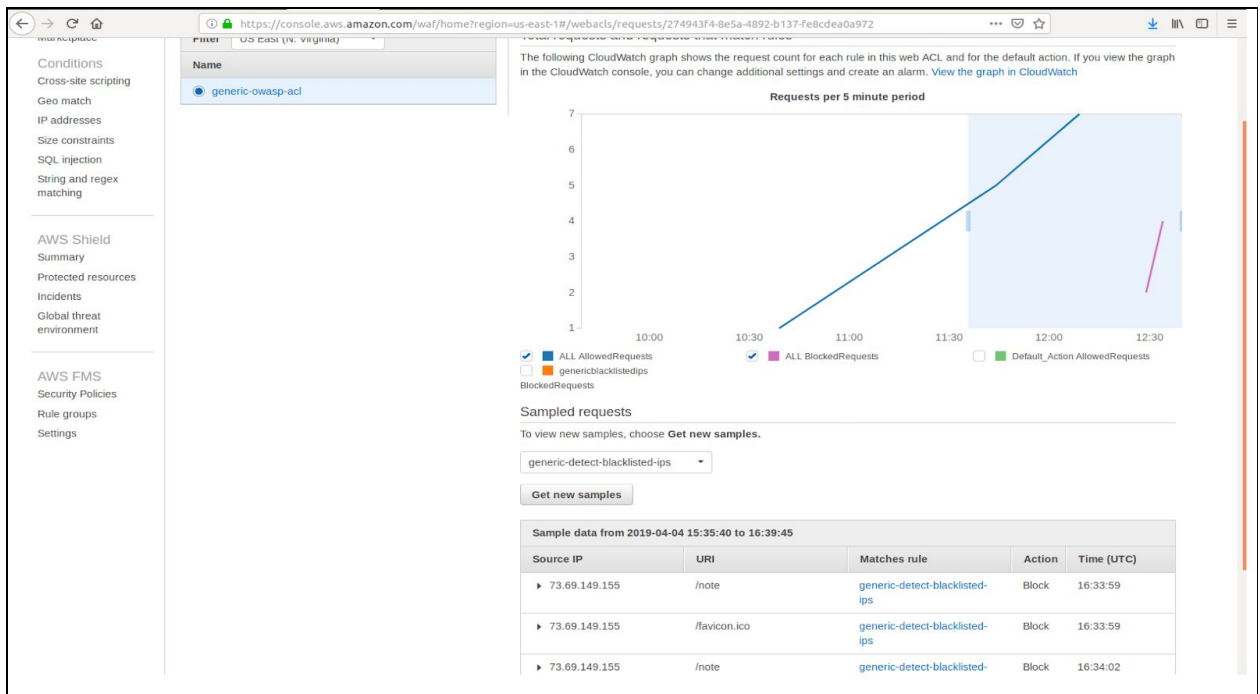
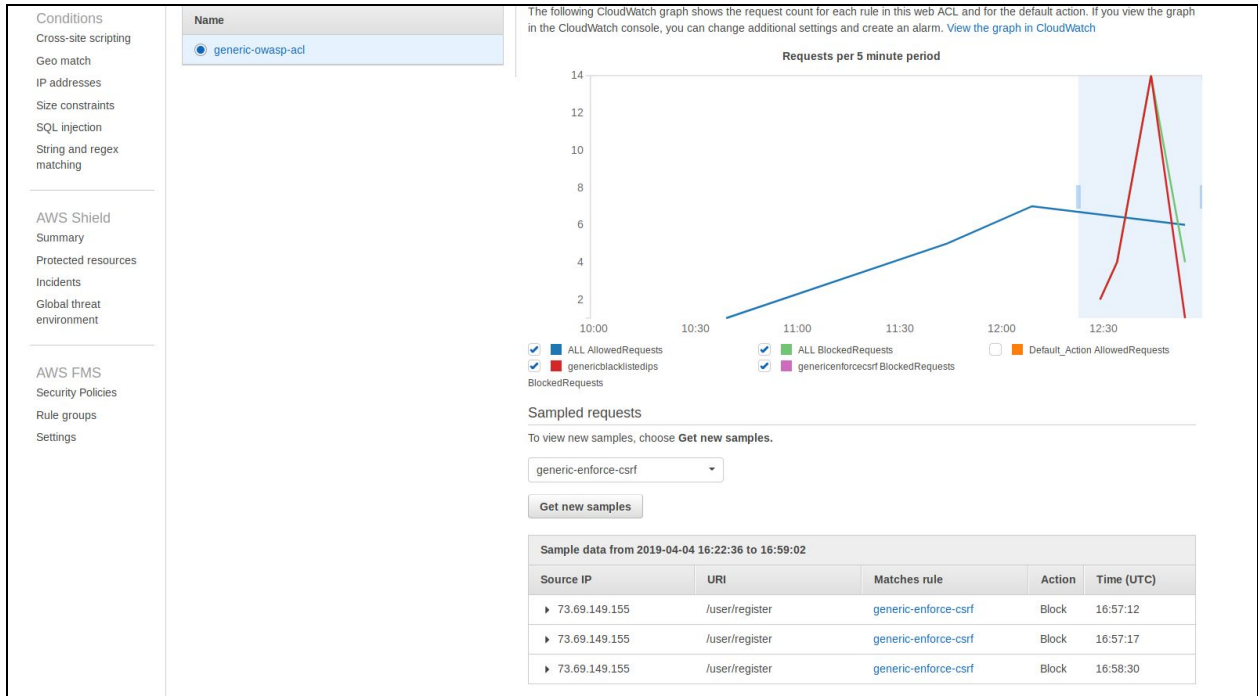
**IP Blacklisting:**Matches IP addresses that should not be allowed to access content. A hacker or cracker can gain access to a computer and deliver payload or malicious code. Hacker can access personal data etc and we can stop that by Blacklisting them for a while until the user provides his/her credentials.

Why this attack vector was chosen ?

Attackers choose this attack vector as they access to millions of valid username and password combinations for credential stuffing, default administrative account lists, automated brute force, and dictionary attack tools.







References:

[https://www.owasp.org/index.php/Web\\_Application\\_Firewall](https://www.owasp.org/index.php/Web_Application_Firewall)

<https://d0.awsstatic.com/whitepapers/Security/aws-waf-owasp.pdf>