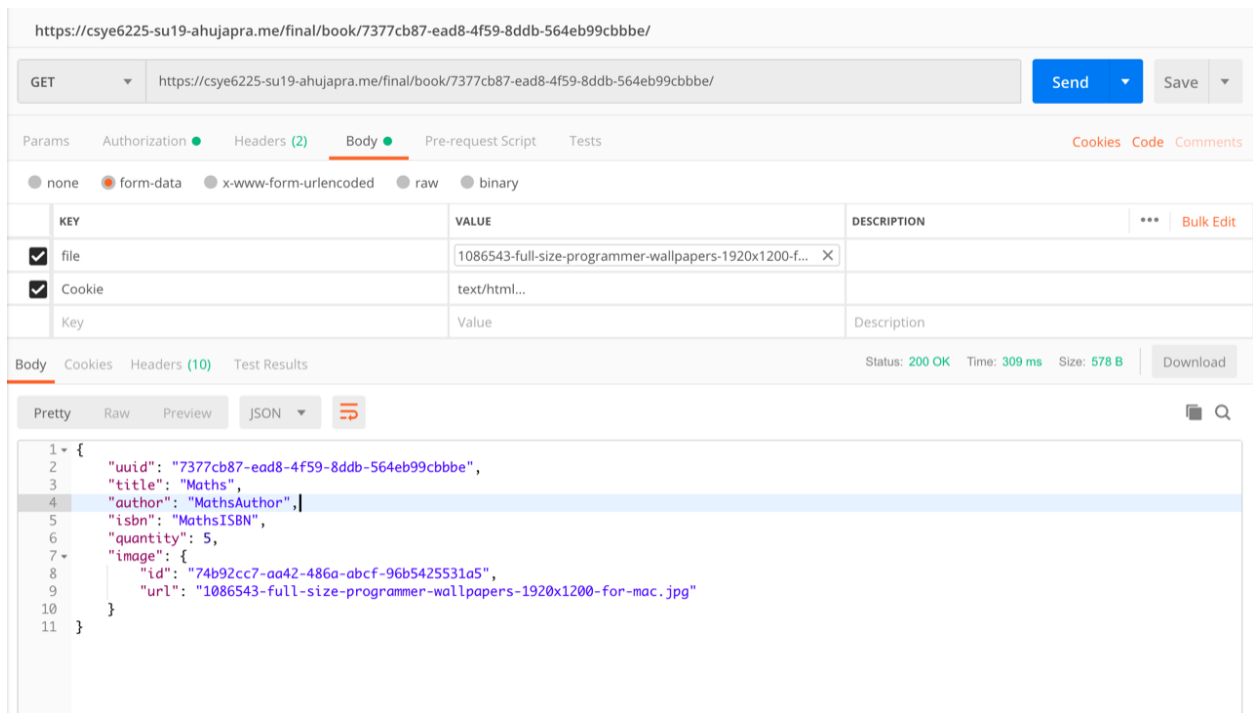


# Attacks on Web Application with and without WAF

## 1. SQL Injection

**Summary:** AWS WAF helps us identify SQL Injection Match set parts of web requests which contains SQL Matches

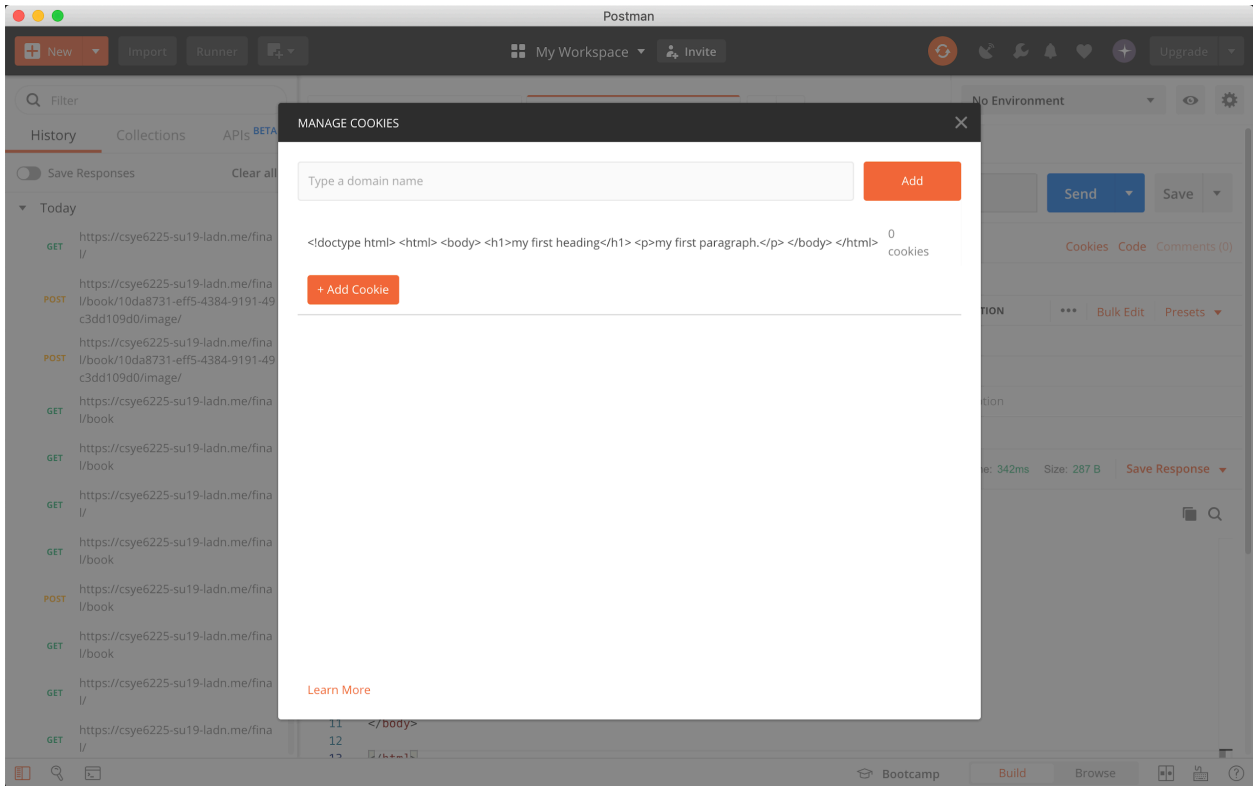
Using No WAF, we were able to send an HTML code using Cookie in “Header” and the value as text/html. The result was still returned by the endpoint since there was no security.



The screenshot shows a web browser interface for a REST client. The URL bar displays `https://csye6225-su19-ahujapra.me/final/book/7377cb87-ead8-4f59-8ddb-564eb99cbbbe/`. The request method is `GET`. The request body is set to `form-data`. The headers section shows a `Cookie` header with the value `text/html...`. The response status is `200 OK` with a time of `309 ms` and a size of `578 B`. The response body is displayed in JSON format:

```
1 {
2   "uuid": "7377cb87-ead8-4f59-8ddb-564eb99cbbbe",
3   "title": "Maths",
4   "author": "MathsAuthor",
5   "isbn": "MathsISBN",
6   "quantity": 5,
7   "image": {
8     "id": "74b92cc7-aa42-486a-abc9-96b5425531a5",
9     "url": "1086543-full-size-programmer-wallpapers-1920x1200-for-mac.jpg"
10  }
11 }
```

However, with WAF enabled, we were able to prevent the request from being sent to the end point as it returned a 403 Forbidden Error. This was the expected response and our first penetration test was successful. This penetration attack was handled by the `SQLInjectionRule` from AWS WAF. The following was the process with WAF

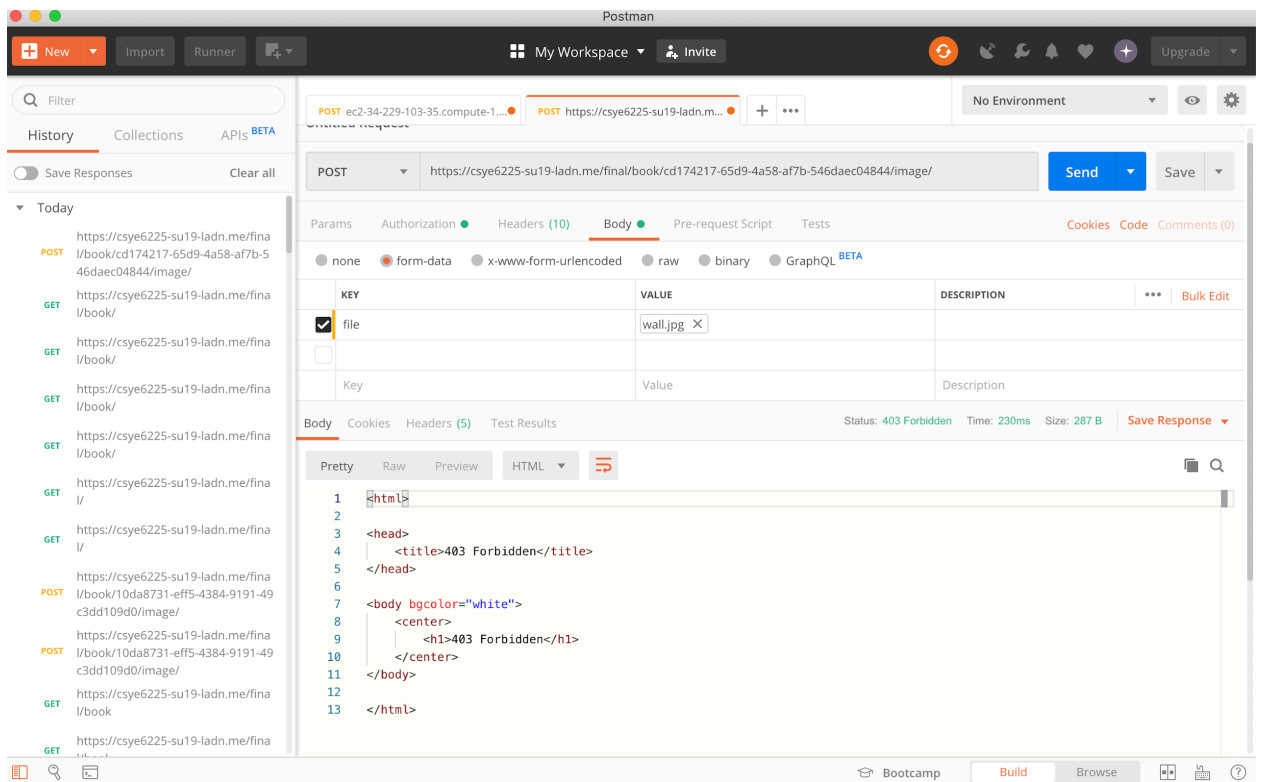
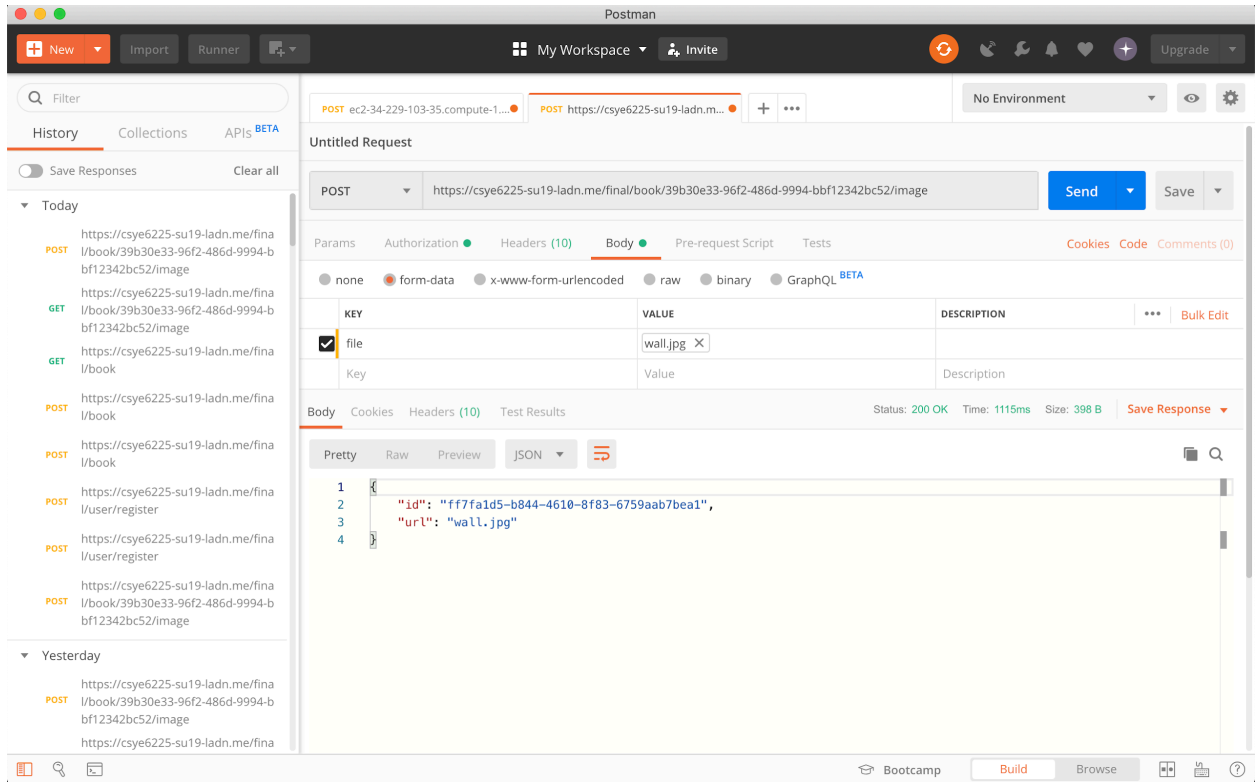


## 2. OverSized request Block

**Summary:** Sometimes the web requests are populated with bulky requests in order to bring the server down. This is sometimes used by the attackers to overload the server with heavy requests where the data can contain heavy objects in the form of images or zip files.

Without WAF, we can send a file of big size greater than 100KB. This might be a small size for now but without proper rules, user might have the flexibility to send a big file size without restriction. As can be seen below in the first image, a file of size 400KB is uploaded successfully without WAF.

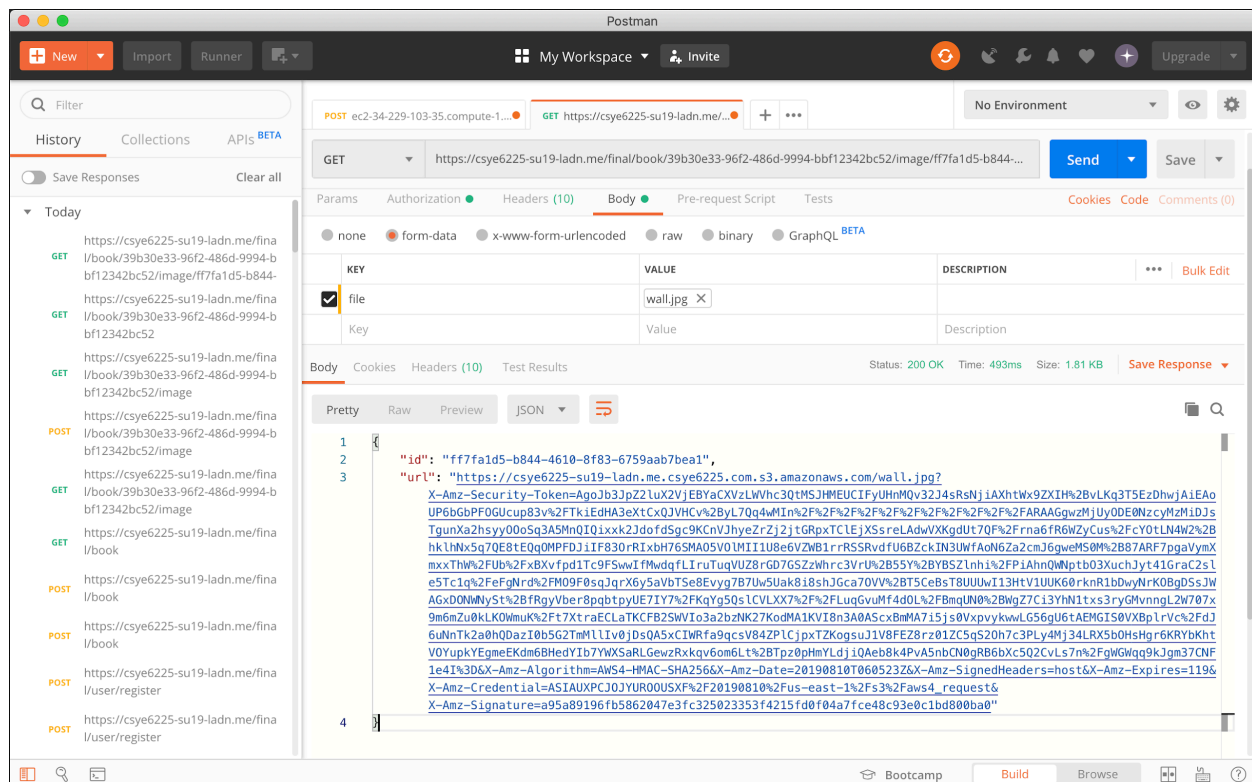
However, in the second image, the action is forbidden due to the restrictions set by WAF. Our second test was successful. This penetration attack was handled by SizeRestrictionRule from AWS WAF. The following was the process with WAF in the second image.



### 3. IP Blocking for Blacklisted IP addresses

**Summary:** Attackers usually use a different Ip from logging into the systems. These IP's are used to hide their real location and hence are blacklisted. Since we had created a VPC with the IP address 11.0.0.0/16 and the associated subnets lie in the above range, we have blocked them for the sake of this assignment.

Without WAF, we have access to the above IP address range and can successfully make the requests. As can be seen, we got the signed URL from the endpoint, since the IP was not blocked.



Now we block 11.0.0.0/16 on WAF as below,

After, blocking the above IP address, we get a 403 forbidden error. We have successfully completed the third penetration test. This penetration attack was handled by IpBlackListRule from AWS WAF. The following was the process with WAF.

## IpBlacklistRule



Edit rule

When a request originates from an IP address in IPSet for blacklisted IP addresses avoiding security vulnerabilities

IP Addresses in IPSet for blacklisted IP addresses avoiding security vulnerabilities

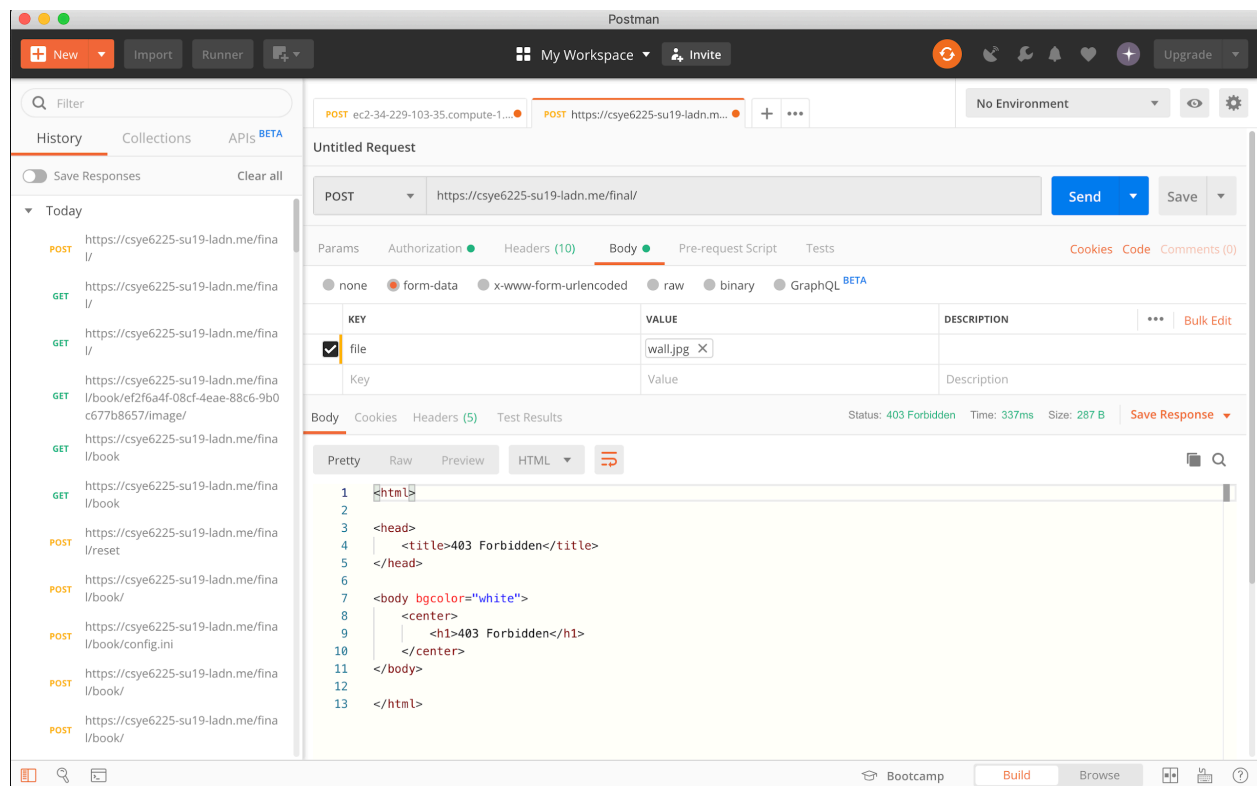
192.168.0.0/16

169.254.0.0/16

172.16.0.0/16

127.0.0.1/32

11.0.0.0/16



## 4. Authentication Token Verification

**Summary:** At times, the authentication token can contain alien cookies or values which can invade a particular application server. This needs to be addressed. In our application testing, we decided to add a cookie in the request header with value csye6225-summer2019 and added a rule to identify this if found in the application header.

Without WAF, the application returns a response with the above cookie while without a header, it does not. The fourth test was successful and this penetration attack was handled by AuthTokenRule from AWS WAF. The first is the test with WAF while next is without WAF.

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

- URL:** `https://csye6225-su19-ahujapra.me/final/`
- Method:** GET
- Headers (3):**
  - Authorization: Basic cHBzYWwh1amFAZ21haWwY29tOjBhcHB5MTIzNCE=
  - Content-Type: application/json
  - Cookie: csye6225-summer2019
- Status:** 200 OK, Time: 312 ms, Size: 418 B
- Body (JSON):**

```
{  "timestamp": "2019-08-10T20:05:23.665+0000",  "message": "20:05:23.665",  "details": ""}
```

The screenshot shows the Postman interface with the following details:

- URL:** `https://csye6225-su19-ladn.me/final/`
- Method:** GET
- Headers (2):**
  - Content-Type: application/json
  - Cookie: csye6225-summer2019
- Status:** 403 Forbidden, Time: 277ms, Size: 287 B
- Body (HTML):**

```
<html>
<head>
  <title>403 Forbidden</title>
</head>
<body bgcolor="white">
  <center>
    <h1>403 Forbidden</h1>
  </center>
</body>
</html>
```