Accessing Web application using domain name: https://csye6225-spring2019-jiangyic.me

---

Attack Vectors:

A1 - Injection

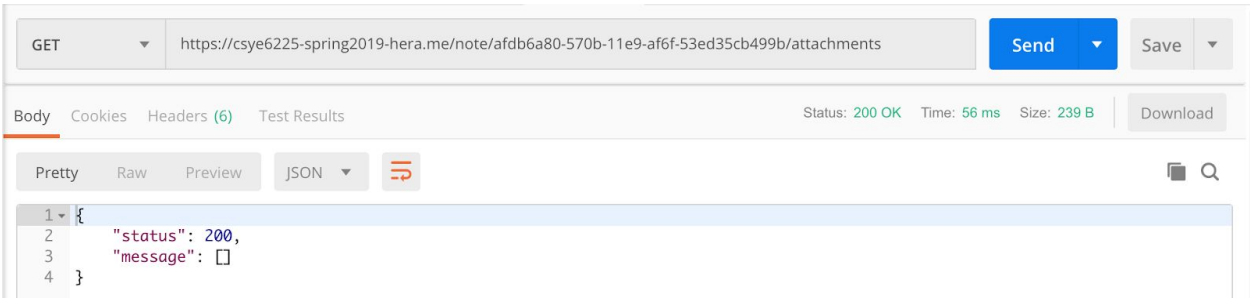
A3 - Cross-Site Scripting (XSS)

A7 – Insufficient Attack Protection

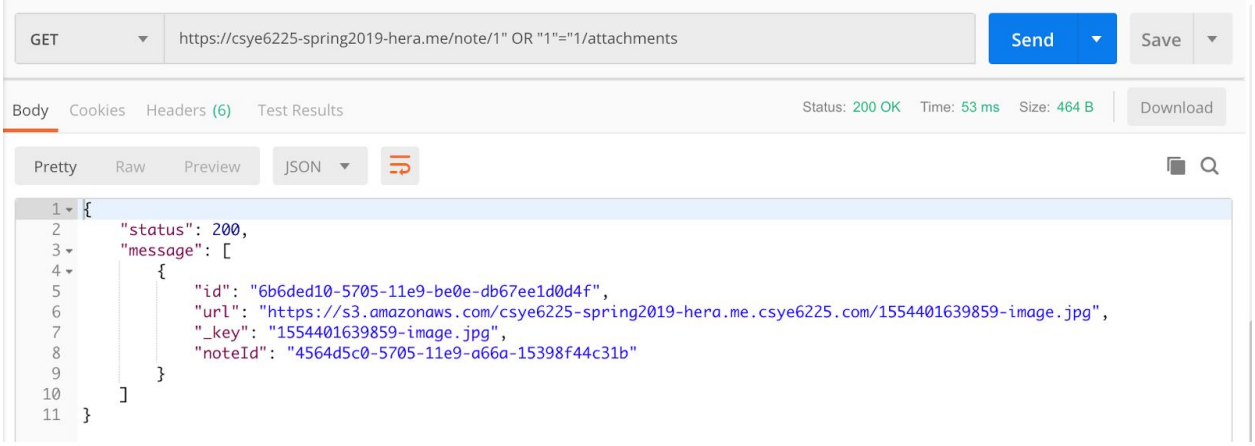
Attacking Results:

1. A1-Injection (*Without Firewall*):

The hacker can get authorized with own credentials and then create a new note. Then tamper with our original sql sentence "SELECT * FROM attachments WHERE noteId=". By replacing noteId with 1" OR "1"="1, the sql sentence becomes SELECT * FROM attachments WHERE noteId=1 OR "1"="1", which will return all attachments in database.

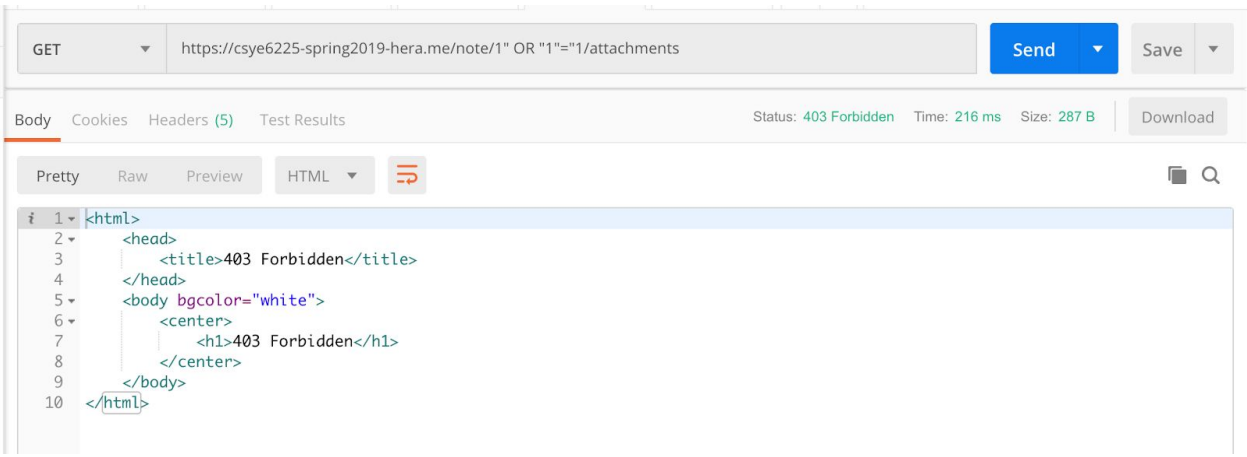


According to the above screenshot, the hacker has no attachments.



The hacker gets other user's attachments from database by injection.

1. A1-Injection (*With Firewall*):
The firewall will forbid the injection. (wafrSQLiRule)



2. A3 - Cross-Site Scripting (XSS without *Firewall*)
The hacker can write some “<script>do some evil thing<script>” script content into the client side content and title input area. After sending the post request to the server, the script data will be inserted into the database. When the administrator want to check all the notes content, he will send a Get request through the route https://csye6225-spring-jiangyic.me/note. Then all the note data including the evil script will be displayed to the administrator. The script can make an fake alert. For example, “<script>Alert(“Your password has been expired”);<script>” will generate a fake warning to administrator, mislead him to register a new account.

POST https://csye6225-spring2019-jiangyic.me/note

Authorization Headers (2) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json)

```
{
    "title":"<script>Alert('evil');<script>",
    "content":"<script>evil<script>"
}
```

Body Cookies Headers (7) Test Results

Pretty Raw Preview JSON

```
{
    "status": 200,
    "noteId": "47130050-5718-11e9-a669-0f4008274730",
    "message": "Note created: <script>Alert('evil');<script>"
}
```

GET https://csye6225-spring2019-jiangyic.me/note

Password ••••••••

Preview Request

Show Pass

Body Cookies Headers (7) Test Results

Pretty Raw Preview JSON


```
[
    {
        "id": "420f3260-5716-11e9-a669-0f4008274730",
        "content": "<script>evil<script>",
        "title": "<script>Alert('evil');<script>",
        "createdOn": "2019:04:04:20:14:32",
        "lastUpdatedOn": "2019:04:04:20:14:32",
        "creator_id": "1"
    },
    {
        "id": "47130050-5718-11e9-a669-0f4008274730",
        "content": "<script>evil<script>",
        "title": "<script>Alert('evil');<script>",
        "createdOn": "2019:04:04:20:28:59",
        "lastUpdatedOn": "2019:04:04:20:28:59",
        "creator_id": "1"
    },
    {
        "id": "84a90cd0-5712-11e9-a669-0f4008274730",
        "content": "Huo123!@",
        "title": "jaing.yig@gmail.com",
        "createdOn": "2019:04:04:19:47:45",
        "lastUpdatedOn": "2019:04:04:19:47:45",
        "creator_id": "1"
    }
]
```


2. A3 - Cross-Site Scripting (XSS with *Firewall*)

The firewall will forbid the XSS attack request. (wafrXSSRule)

POST https://csye6225-spring2019-jiangyic.me/note

Authorization Headers (2) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json)

```
{
    "title":"<script>Alert('evil');<script>",
    "content":"<script>evil<script>"
}
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview HTML

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

POST https://csye6225-spring2019-jiangyic.me/note

Authorization Headers (2) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json)

```
{
    "title":"<script>Alert('evil');<script>",
    "content":"<script>evil<script>"
}
```

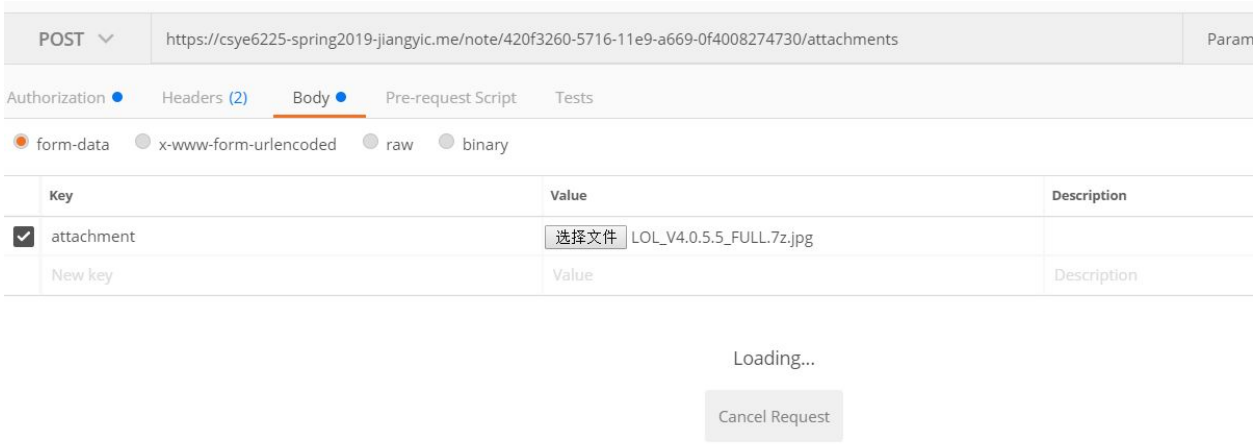
Body Cookies Headers (5) Test Results

Pretty Raw Preview HTML

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

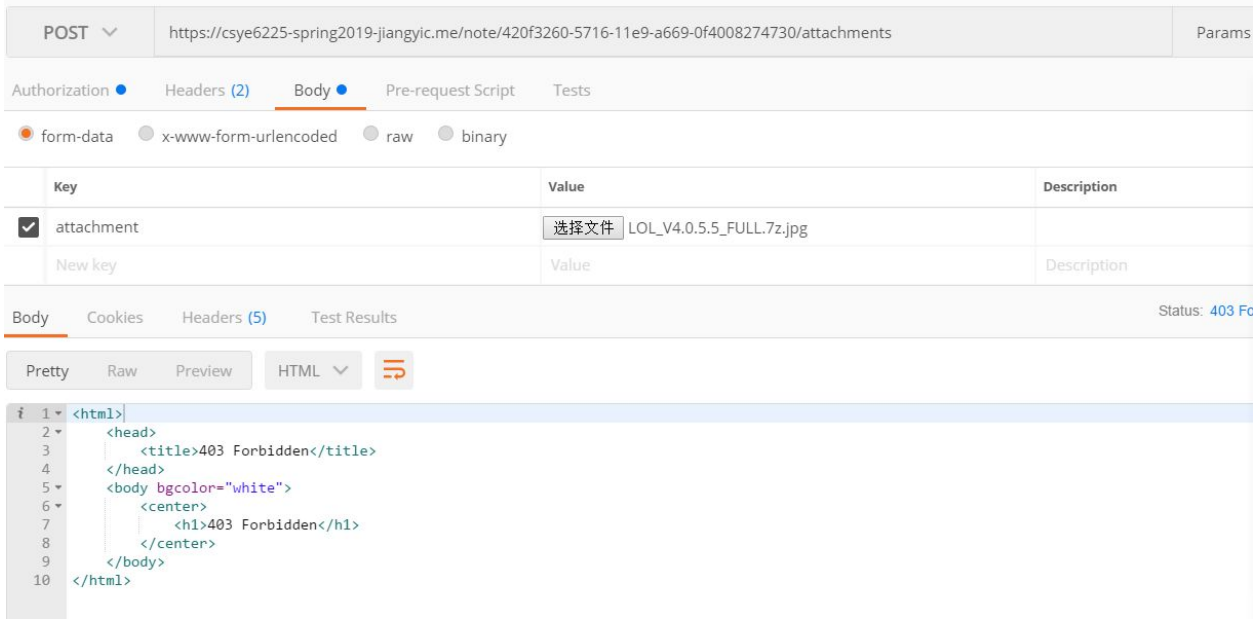
3. A7 – Insufficient Attack Protection(without *Firewall*)

The hacker can upload a malicious large attachment to the instances' S3 bucket deliberately. It can induce considerable delays to instances or even crash them.



3. A7 – Insufficient Attack Protection(with *Firewall*)

The Firewall will check the attachment size in the request body, to make sure it is less than the maximum allowed size. Or the request will be forbidden. (wafrSizeRestrictionRule)



-

Reason For choosing A1, A3 and A7

SQL injection and XSS attack is the most common format attack in the server attacking. Also, our instances can not run under very heavy load, this is important for them. So, we add the A7.