NETWORK STRUCTURE AND CLOUD COMPUTING (CSYE-6225)

PENETRATION TESTING
ATTACK VECTORS

Introduction

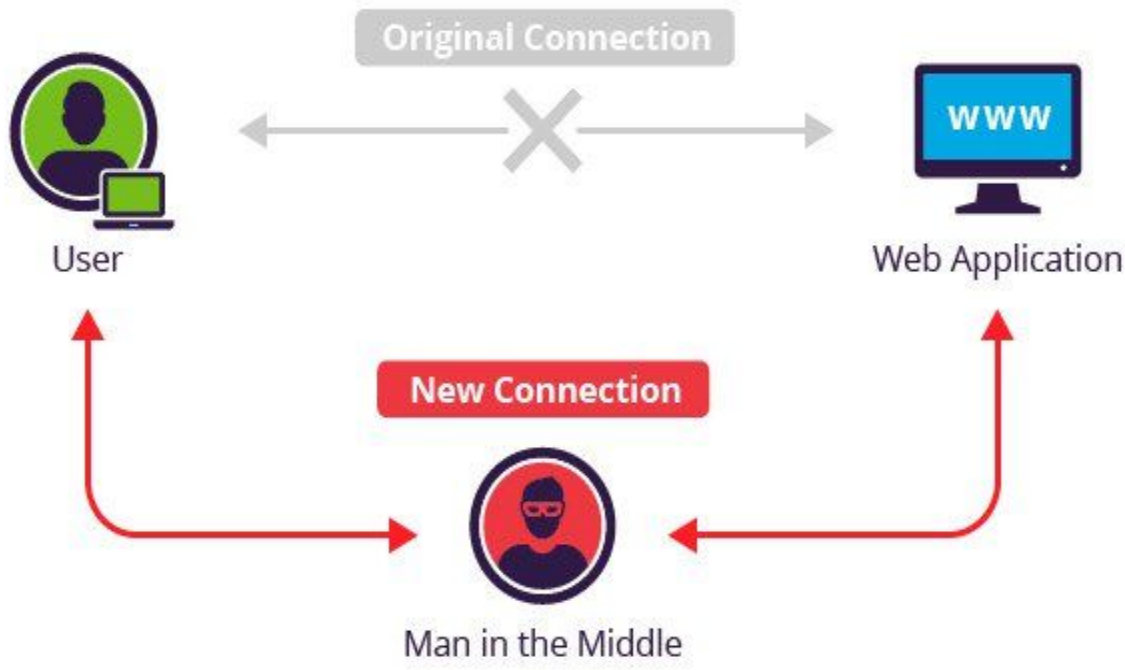
This document details the attack vectors used to perform penetration testing on an application that can be accessed via Restful APIs hosted on an EC2 instance. The Application is can register a user. The primary purpose of the application is for users to keep track of their expenditure by also giving an option to upload receipts to a expense tracked. A registered user can perform CRUD operations on a transaction that he created as well as attach receipts to that transaction. There is also an option to reset the user password and for a registered user to fetch the current time by providing authentication in the request header.

Below are Attack Vectors used to perform penetration testing on the application.

Attack Vectors

1. Man in the middle attack (MITM):

The Man in the middle attack is a term used when a hacker intercepts request from the user or the client which is the browser in most cases and the server where the application that servers the request resides.



In our scenario, this attack was used while creating a transaction using a POST request. The registered user attempts to create a transaction with a specific amount of $2.69. This request is intercepted by the hacker and the amount is changed from $2.69 to $3000 and sent ahead to the server. The server has no idea of the modified request and treats it as the original request and thus saves the transaction data for that particular user. The pictorial description of this attacks is as shown below.

The Original Request :

34.235.116.236:8080/login-0.0.1/transaction/

POST | 34.235.116.236:8080/login-0.0.1/transaction/ | Send | Save

Params | Authorization | Headers (2) | Body | Pre-request Script | Tests | Cookies | Code

none | form-data | x-www-form-urlencoded | raw | binary | JSON (application/json) | Beautify

```
1 {
2   "description": "coffee",
3   "merchant": "starbucks",
4   "amount": 2.69,
5   "date": "09/25/2018",
6   "category": "food"
7 }
```

The Modified Request :

Original request | Auto-modified request | Edited request | Original response | Auto-modified response

Raw | Params | Headers | Hex

```
Postman-Token: c0e271ff-4743-4fda-8f51-8965152980b2
Authorization: Basic bWV2ZW4uZGN1bmhhQGdtYWlsLmNvbTp0ZXN0
User-Agent: Mozilla/5.0 (Linux; U; Android 2.2; en-us; Droid Build/FRG22D) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1
Accept: */*
Host: 34.235.116.236:8080
Content-Length: 115
Connection: close
Origin: foo.example.org

{
 "description": "coffee",
 "merchant": "starbucks",
 "amount": 3000,
 "date": "09/25/2018",
 "category": "food"
}
```

The user on fetching his transactions would now see the modified transaction in the response

The Edited Response :

GET 34.235.116.236:8080/login-0.0.1/transaction Send Save

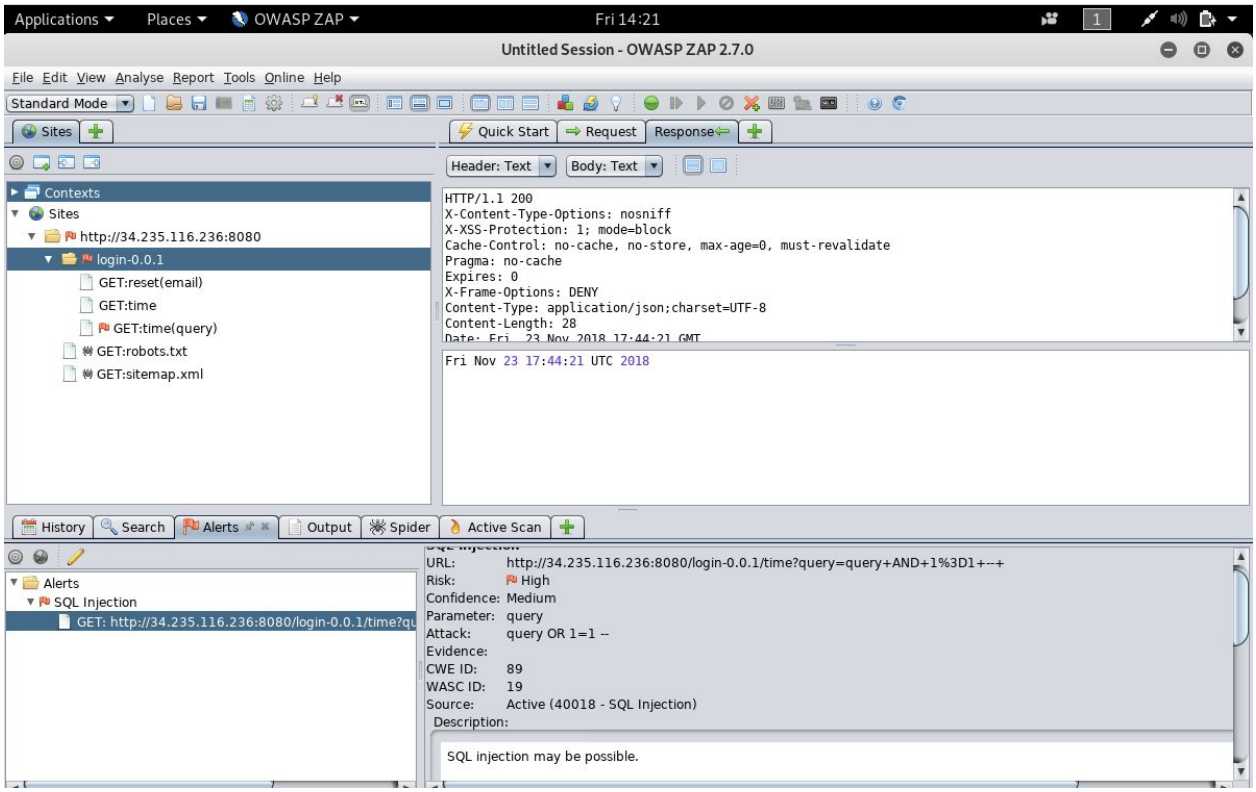
Body Cookies Headers (10) Test Results Status: 200 OK Time: 12954 ms Size: 1.02 KB Download

Pretty Raw Preview JSON


```
43                  "uri": "Images/meven.dcunha@gmail.com_LargerImage.jpeg"
44                }
45            ]
46        },
47        {
48            "statusCode": null,
49            "message": null,
50            "transactionId": "0b5c2b53-ed83-43a3-a0c9-b0f5d0cde4fa",
51            "description": "coffee",
52            "merchant": "starbucks",
53            "amount": 3000,
54            "date": "09/25/2018",
55            "category": "food",
56            "user": {
57                "statusCode": null,
58                "message": null,
59                "id": 1,
60                "email": "meven.dcunha@gmail.com",
61                "password": "$2a$10$iFM99YYh88UEqv88y8JIAe/P6.NcEyVtzUMkOHTkqTPvrKuXSu.4i",
62                "name": "qwerty",
63                "lastName": "qwerty",
64                "active": 1,
65                "roles": [
66                    "ROLE_USER",
67                    "ROLE_ADMIN"
68                ]
69            },
70            "attachments": []
71        }
72    ]
73 }
```

2 . SQL Injection Attack

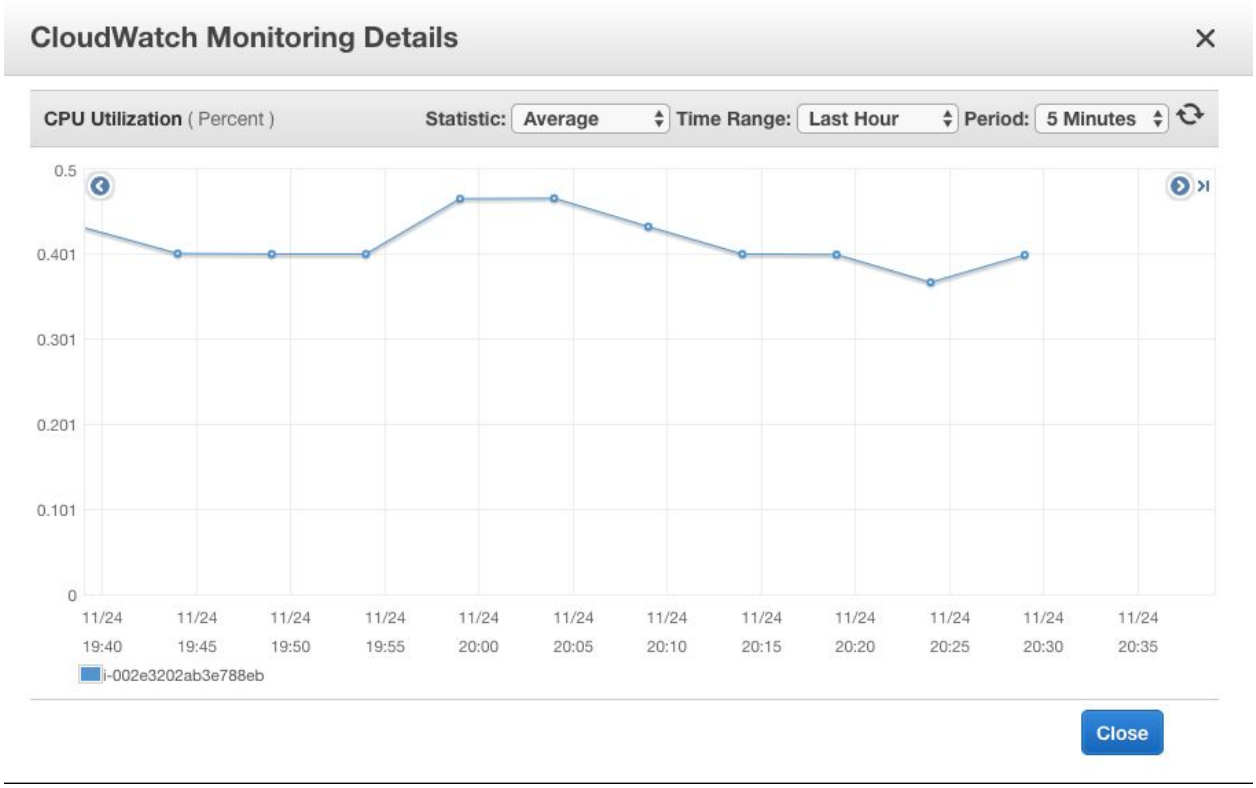
This attack uses malicious SQL code to manipulate the backend database information and could also lead to loss of data altogether. In our specific scenario we have used the tool OWASP ZAP to check for SQL injection vulnerabilities in the application. The report generated is as shown below



From the above result it can be seen that an SQL injection can be possible on the get time API for a registered user. Although the chances of attempting a SQL injection in a GET request seems minimal, the report suggest that with the api call modification as shown in the image, there is a possibility of an SQL injection and has termed it as a High risk alert.

3. Brute Force Attack to degrade performance

Brute Force attack is typically used to gain access to username and password by repeatedly attempting various combinations. The way Brute Force has been used in our scenario is a bit deviant in our scenario. The reset password API is repeatedly hit (3000 request) thus impacting the performance of the instance. The graph of the performance when the attack was attempted is as shown below :



From the above image it can be noted that the CPU Utilization of the instance increased by approximately 8% when the Brute force attack was attempted. The main goal of the hack here is to degrade the server utilization eventually.

For the above reading the Brute force attack was made for 300 request using Burp Suite which is depicted in the below images

Intruder attack 3

Attack Save Columns

Results | Target | Positions | Payloads | Options

Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1 Payload count: 4

Payload type: Simple list Request count: 300

Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

Paste
Load ...
Remove
Clear

meven
test
newtest
testall

Add Enter a new item

Add from list ... [Pro version only]

Payload Processing

Intruder attack 3

Attack Save Columns

Results | Target | Positions | Payloads | Options

Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Cluster bomb

```
§GET /login-0.0.1/reset?email=§meven.dcunha@gmai§ HTTP/1.0
Host: 34.235.116.236:8080
User-Agent: Mozilla/5.0 (Linux; U; Android 2.2; en-us; Droid Build/FRG22D) AppleWebKit/533.1 (KHTML, like
Gecko) Version/4.0 Mobile Safari/533.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: close
Upgrade-Insecure-Requests: 1
Origin: foo.example.org
```

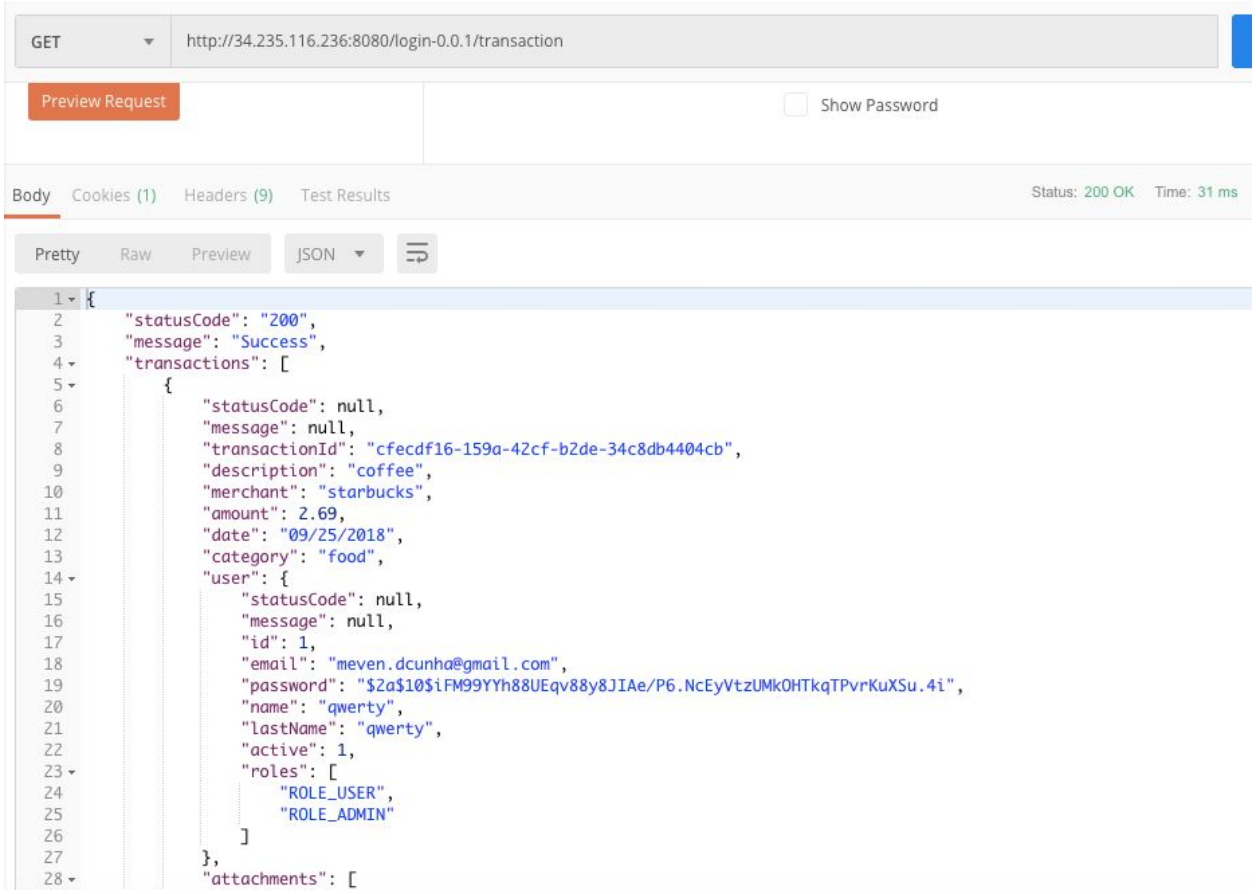
Add §
Clear §
Auto §
Refresh

? < + > Type a search term 0 matches Clear

2 payload positions Length: 415

4. Sensitive Data Exposure

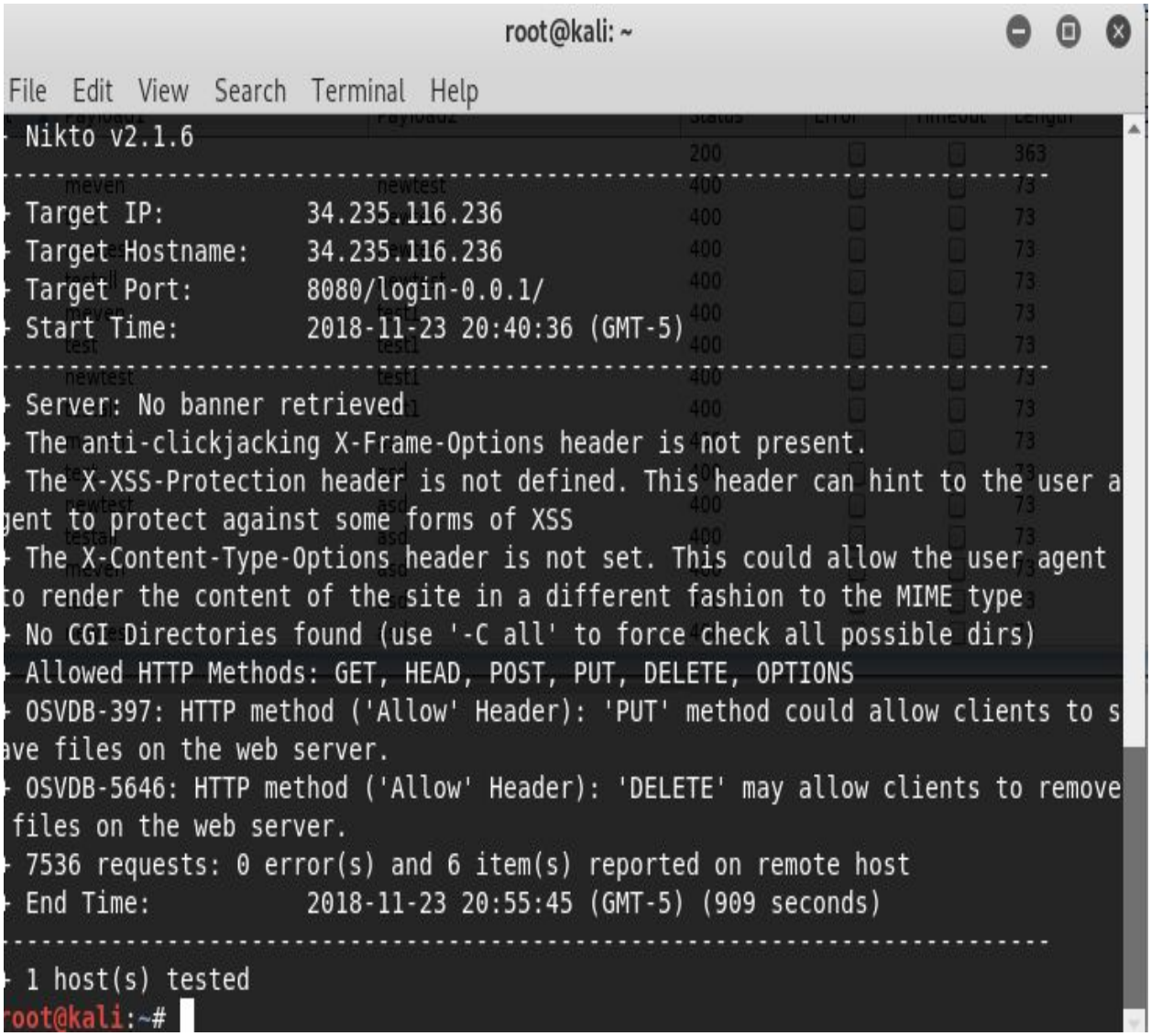
This attack can be triggered when some sensitive data is revealed in the application. In the case of our Rest application, on fetching the transaction the user data is revealed for who that transaction belongs to. Below is the image that demonstrates that.



This attack when combined with the Man in the middle attack and provide the hacker with user data. Although the password stored in this data is encrypted by a unique SALT using Bcrypt, by using intensive resources to figure out the SALT used for a particular user, this could be decrypted. Also, the email Id intercepted would give the hacker information of the registered users.

5. Unprotected APIs

By using Nikto scanner, the potential vulnerabilities in the APIs were detected. The image of the scanned result is as shown below



All of the attacks apart for the 4th one were prevented using the Web Application Firewall (WAF) with OWASP's Top 10 Web Application Vulnerabilities configured.