Graham Dungan
January 20, 2025
UIN: 332001764

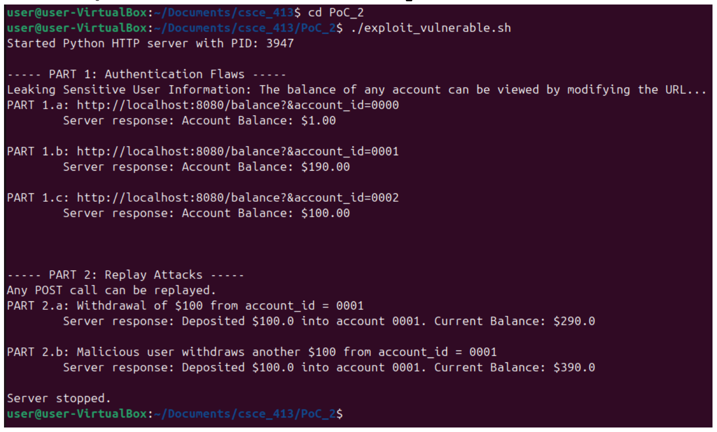
CSCE 413: Software Security
PoC 2

Web Application Creation (40 Points)

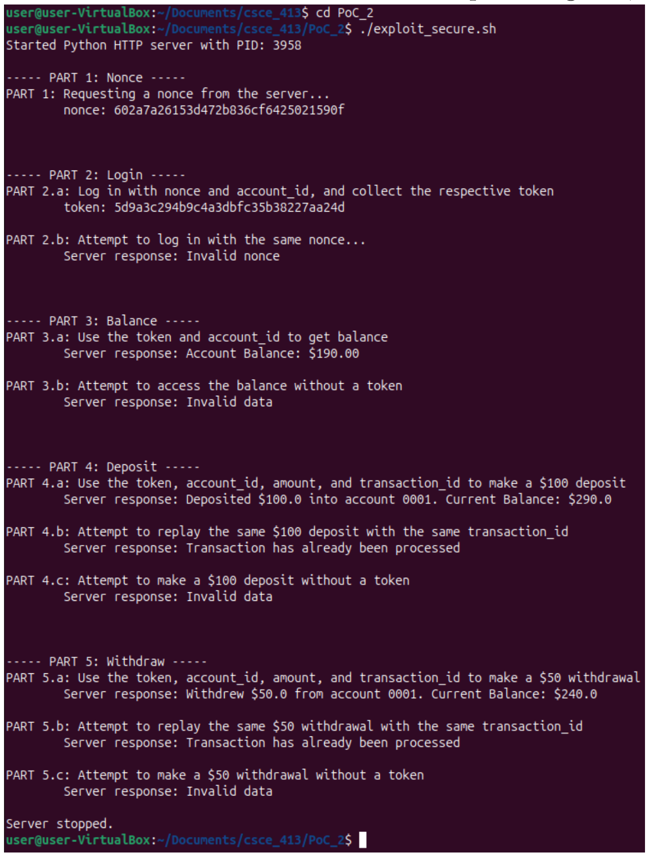
I created a simple vulnerable REST banking API allowing users to check their balance, deposit, or withdraw money. The `/balance` route is a GET call regarded as sensitive information. Users should only be able to get their balance. The `/deposit` and `/withdraw` routes are POST calls that increment or decrement some account's money. Contained alongside this .pdf is;

```
PoC_2/
 ├── vulnerableWebapp.py
 ├── secureWebapp.py
 ├── exploit_vulnerable.sh
 ├── exploit_secure.sh
 └── PoC2_grahamd.pdf
```

`vulnerableWebapp.py` and `secureWebapp.py` are Python scripts that run the simple HTTP server with a vulnerable and secure version, respectively. The `exploit_vulnerable.sh` script runs through a step-by-step process of how `vulnerableWebapp.py` can be exploited. The `exploit_secure.sh` script runs through a step-by-step process of how `secureWebapp.py` is resistant to the aforementioned exploits. Run the scripts as normal under the `PoC_2/` directory. Ex: `cd PoC_2 && ./exploit_vulnerable.sh`. What follows are screenshots of these scripts being run;



```
user@user-VirtualBox:~/Documents/csce_413$ cd PoC_2
user@user-VirtualBox:~/Documents/csce_413/PoC_2$ ./exploit_vulnerable.sh
Started Python HTTP server with PID: 3947

----- PART 1: Authentication Flaws -----
Leaking Sensitive User Information: The balance of any account can be viewed by modifying the URL...
PART 1.a: http://localhost:8080/balance?&account_id=0000
        Server response: Account Balance: $1.00

PART 1.b: http://localhost:8080/balance?&account_id=0001
        Server response: Account Balance: $190.00

PART 1.c: http://localhost:8080/balance?&account_id=0002
        Server response: Account Balance: $100.00



----- PART 2: Replay Attacks -----
Any POST call can be replayed.
PART 2.a: Withdrawal of $100 from account_id = 0001
        Server response: Deposited $100.0 into account 0001. Current Balance: $290.0

PART 2.b: Malicious user withdraws another $100 from account_id = 0001
        Server response: Deposited $100.0 into account 0001. Current Balance: $390.0

Server stopped.
user@user-VirtualBox:~/Documents/csce_413/PoC_2$
```



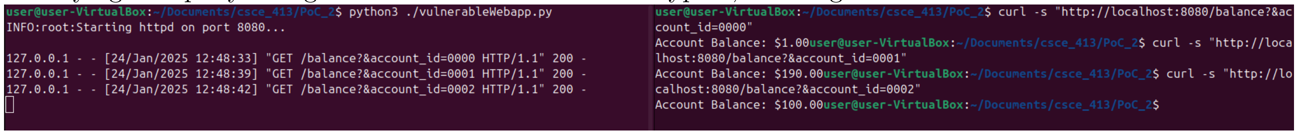
```
user@user-VirtualBox:~/Documents/csce_413$ cd PoC_2
user@user-VirtualBox:~/Documents/csce_413/PoC_2$ ./exploit_secure.sh
Started Python HTTP server with PID: 3958

----- PART 1: Nonce -----
PART 1: Requesting a nonce from the server...
        nonce: 602a7a26153d472b836cf6425021590f



----- PART 2: Login -----
PART 2.a: Log in with nonce and account_id, and collect the respective token
        token: 5d9a3c294b9c4a3dbfc35b38227aa24d

PART 2.b: Attempt to log in with the same nonce...
        Server response: Invalid nonce



----- PART 3: Balance -----
PART 3.a: Use the token and account_id to get balance
        Server response: Account Balance: $190.00

PART 3.b: Attempt to access the balance without a token
        Server response: Invalid data



----- PART 4: Deposit -----
PART 4.a: Use the token, account_id, amount, and transaction_id to make a $100 deposit
        Server response: Deposited $100.0 into account 0001. Current Balance: $290.0

PART 4.b: Attempt to replay the same $100 deposit with the same transaction_id
        Server response: Transaction has already been processed

PART 4.c: Attempt to make a $100 deposit without a token
        Server response: Invalid data



----- PART 5: Withdraw -----
PART 5.a: Use the token, account_id, amount, and transaction_id to make a $50 withdrawal
        Server response: Withdrew $50.0 from account 0001. Current Balance: $240.0

PART 5.b: Attempt to replay the same $50 withdrawal with the same transaction_id
        Server response: Transaction has already been processed

PART 5.c: Attempt to make a $50 withdrawal without a token
        Server response: Invalid data

Server stopped.
user@user-VirtualBox:~/Documents/csce_413/PoC_2$
```

Exploitation of Vulnerabilities (20 Points)

A demonstration of these vulnerabilities can be found by running the provided `exploit_vulnerable.sh` script.

Authentication Bypass

Modifying the query string results in an authentication bypass, allowing users to see other users' information.



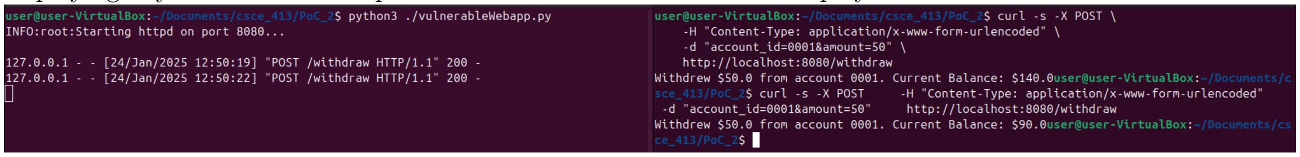
```
user@user-VirtualBox:~/Documents/csce_413/PoC_2$ python3 ./vulnerableWebapp.py
INFO:root:Starting httpd on port 8080...

127.0.0.1 - - [24/Jan/2025 12:48:33] "GET /balance?&account_id=0000 HTTP/1.1" 200 -
127.0.0.1 - - [24/Jan/2025 12:48:39] "GET /balance?&account_id=0001 HTTP/1.1" 200 -
127.0.0.1 - - [24/Jan/2025 12:48:42] "GET /balance?&account_id=0002 HTTP/1.1" 200 -
```

```
user@user-VirtualBox:~/Documents/csce_413/PoC_2$ curl -s "http://localhost:8080/balance?&ac
count_id=0000"
Account Balance: $1.00user@user-VirtualBox:~/Documents/csce_413/PoC_2$ curl -s "http://loca
lhost:8080/balance?&account_id=0001"
Account Balance: $190.00user@user-VirtualBox:~/Documents/csce_413/PoC_2$ curl -s "http://lo
calhost:8080/balance?&account_id=0002"
Account Balance: $100.00user@user-VirtualBox:~/Documents/csce_413/PoC_2$
```

This occurs because there is no means of checking the authenticity, such that we can verify the person accessing the account's balance is the actual user. To fix this would require a proof of identity, as well as route protection.

Replay Attacks

Replaying any withdrawal or deposit calls will result in a successful replay attack.



```
user@user-VirtualBox:~/Documents/csce_413/PoC_2$ python3 ./vulnerableWebapp.py
INFO:root:Starting httpd on port 8080...

127.0.0.1 - - [24/Jan/2025 12:50:19] "POST /withdraw HTTP/1.1" 200 -
127.0.0.1 - - [24/Jan/2025 12:50:22] "POST /withdraw HTTP/1.1" 200 -
```

```
user@user-VirtualBox:~/Documents/csce_413/PoC_2$ curl -s -X POST \
    -H "Content-Type: application/x-www-form-urlencoded" \
    -d "account_id=0001&amount=50" \
    http://localhost:8080/withdraw
Withdrew $50.0 from account 0001. Current Balance: $140.0user@user-VirtualBox:~/Documents/c
sce_413/PoC_2$ curl -s -X POST     -H "Content-Type: application/x-www-form-urlencoded"
 -d "account_id=0001&amount=50"     http://localhost:8080/withdraw
Withdrew $50.0 from account 0001. Current Balance: $90.0user@user-VirtualBox:~/Documents/cs
ce_413/PoC_2$
```

In this instance, an attacker could replay a withdrawal, causing the owner of the account to lose money. Following, anyone can deposit or withdraw money from any account. Fixing this would require some kind of authentication, as well as an expirable token to prevent transactions from occurring more than once.

Fixing the Vulnerabilities (20 Points)

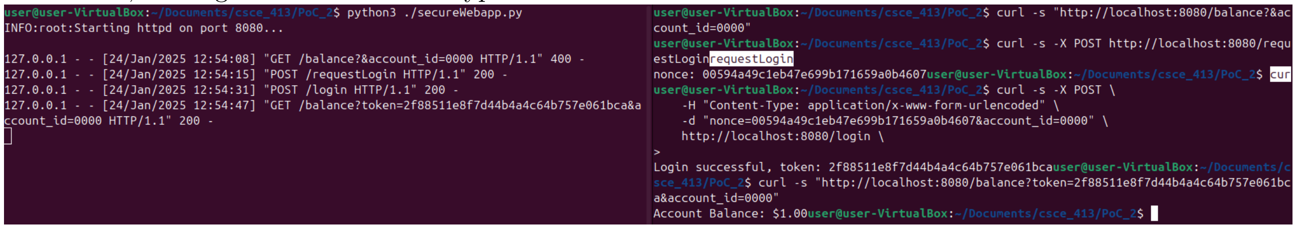
A demonstration of the script protecting against these exploits can be found by running the provided `exploit_secure.sh` script.

Authentication Bypass

My solution for preventing authentication bypass is a two-step authentication process;

1. The user requests to log in via the `/requestLogin` route. The server responds with a nonce for the user to log in with.
2. The user uses the nonce to log in with their credentials using the `/login` route. The nonce is destroyed, and the server responds with a session token.

This two-step authentication process ensures that a user is provided with a unique login opportunity. Any replay attacks would fail since nonce can only be used once. If the attacker were to obtain the user's nonce before they have logged in, they would need to know the user's credentials and log in faster than the user. The token verifies the authenticity of all user requests for a period of time. Once the time on the token expires, they will need to re-authenticate. Since the token verifies the user's identity, it is no longer possible to access routes without valid credentials, barring an authentication bypass.

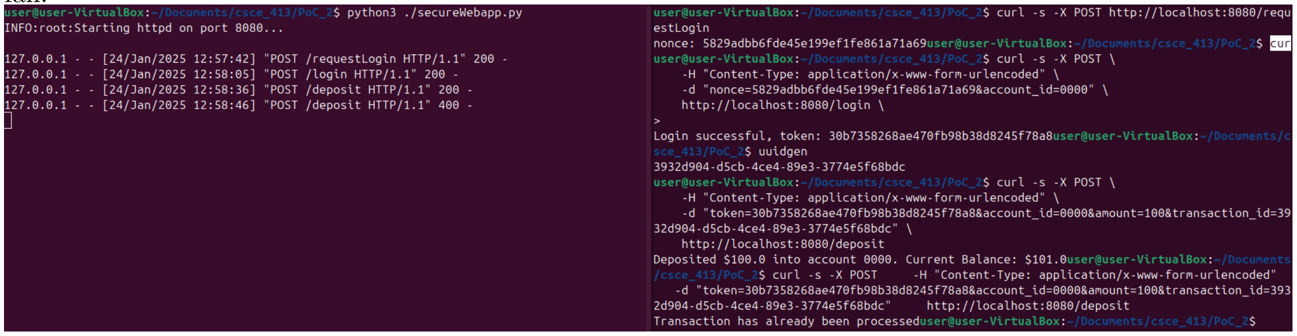


```
user@user-VirtualBox:~/Documents/csce_413/PoC_2$ python3 ./secureWebapp.py
INFO:root:Starting httpd on port 8080...

127.0.0.1 - - [24/Jan/2025 12:54:08] "GET /balance?&account_id=0000 HTTP/1.1" 400 -
127.0.0.1 - - [24/Jan/2025 12:54:15] "POST /requestLogin HTTP/1.1" 200 -
127.0.0.1 - - [24/Jan/2025 12:54:31] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [24/Jan/2025 12:54:47] "GET /balance?token=2f88511e8f7d44b4a4c64b757e061bca&a
ccount_id=0000 HTTP/1.1" 200 -
```

```
user@user-VirtualBox:~/Documents/csce_413/PoC_2$ curl -s "http://localhost:8080/balance?&ac
count_id=0000"
user@user-VirtualBox:~/Documents/csce_413/PoC_2$ curl -s -X POST http://localhost:8080/requ
estLoginrequestLogin
nonce: 00594a49c1eb47e699b171659a0b4607user@user-VirtualBox:~/Documents/csce_413/PoC_2$ cur
user@user-VirtualBox:~/Documents/csce_413/PoC_2$ curl -s -X POST \
    -H "Content-Type: application/x-www-form-urlencoded" \
    -d "nonce=00594a49c1eb47e699b171659a0b4607&account_id=0000" \
    http://localhost:8080/login \
>
Login successful, token: 2f88511e8f7d44b4a4c64b757e061bcauser@user-VirtualBox:~/Documents/c
sce_413/PoC_2$ curl -s "http://localhost:8080/balance?token=2f88511e8f7d44b4a4c64b757e061bc
a&account_id=0000"
Account Balance: $1.00user@user-VirtualBox:~/Documents/csce_413/PoC_2$
```

Replay Attacks

The solution listed above also prevents replay attacks. In the login scenario, a `/login` request needs a unique, one-time-use nonce, so a replay of any request will simply fail. For deposits and withdrawals, the user will generate a unique transaction ID to send to the server. Any attempts at replaying with the same ID will be caught and will fail.

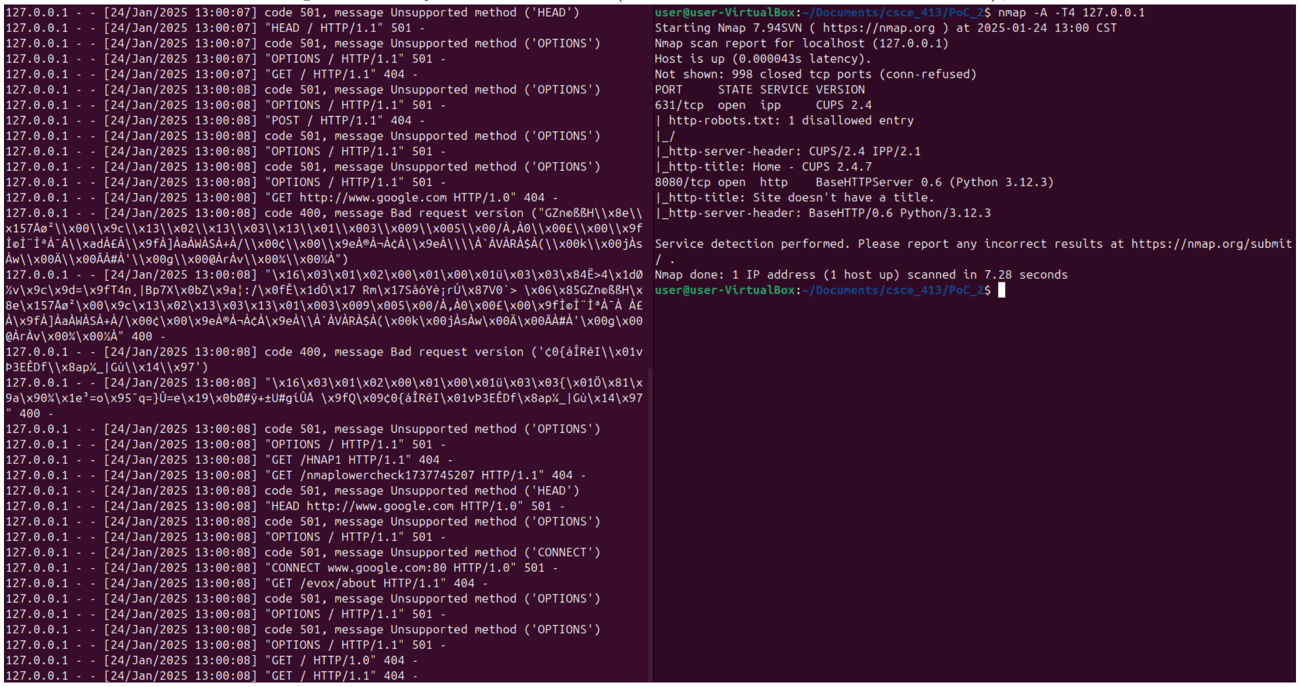


```
user@user-VirtualBox:~/Documents/csce_413/PoC_2$ python3 ./secureWebapp.py
INFO:root:Starting httpd on port 8080...

127.0.0.1 - - [24/Jan/2025 12:57:42] "POST /requestLogin HTTP/1.1" 200 -
127.0.0.1 - - [24/Jan/2025 12:58:05] "POST /login HTTP/1.1" 200 -
127.0.0.1 - - [24/Jan/2025 12:58:36] "POST /deposit HTTP/1.1" 200 -
127.0.0.1 - - [24/Jan/2025 12:58:46] "POST /deposit HTTP/1.1" 400 -
```

```
user@user-VirtualBox:~/Documents/csce_413/PoC_2$ curl -s -X POST http://localhost:8080/requ
estLogin
nonce: 5829adbb6fde45e199ef1fe861a71a69user@user-VirtualBox:~/Documents/csce_413/PoC_2$ cur
user@user-VirtualBox:~/Documents/csce_413/PoC_2$ curl -s -X POST \
    -H "Content-Type: application/x-www-form-urlencoded" \
    -d "nonce=5829adbb6fde45e199ef1fe861a71a69&account_id=0000" \
    http://localhost:8080/login \
>
Login successful, token: 30b7358268ae470fb98b38d8245f78a8user@user-VirtualBox:~/Documents/c
sce_413/PoC_2$ uuidgen
3932d904-d5cb-4ce4-89e3-3774e5f68bdc
user@user-VirtualBox:~/Documents/csce_413/PoC_2$ curl -s -X POST \
    -H "Content-Type: application/x-www-form-urlencoded" \
    -d "token=30b7358268ae470fb98b38d8245f78a8&account_id=0000&amount=100&transaction_id=39
32d904-d5cb-4ce4-89e3-3774e5f68bdc" \
    http://localhost:8080/deposit
Deposited $100.0 into account 0000. Current Balance: $101.0user@user-VirtualBox:~/Documents
/csce_413/PoC_2$ curl -s -X POST     -H "Content-Type: application/x-www-form-urlencoded"
   -d "token=30b7358268ae470fb98b38d8245f78a8&account_id=0000&amount=100&transaction_id=393
2d904-d5cb-4ce4-89e3-3774e5f68bdc"     http://localhost:8080/deposit
Transaction has already been processeduser@user-VirtualBox:~/Documents/csce_413/PoC_2$
```

Use of NMAP or Vulnerability Scanner (20 Points)

I used NMAP to scan all ports of my local machine (a Linux instance on VirtualBox),



```
127.0.0.1 - - [24/Jan/2025 13:00:07] code 501, message Unsupported method ('HEAD')
127.0.0.1 - - [24/Jan/2025 13:00:07] "HEAD / HTTP/1.1" 501 -
127.0.0.1 - - [24/Jan/2025 13:00:07] code 501, message Unsupported method ('OPTIONS')
127.0.0.1 - - [24/Jan/2025 13:00:07] "OPTIONS / HTTP/1.1" 501 -
127.0.0.1 - - [24/Jan/2025 13:00:07] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [24/Jan/2025 13:00:08] code 501, message Unsupported method ('OPTIONS')
127.0.0.1 - - [24/Jan/2025 13:00:08] "OPTIONS / HTTP/1.1" 501 -
127.0.0.1 - - [24/Jan/2025 13:00:08] "POST / HTTP/1.1" 404 -
127.0.0.1 - - [24/Jan/2025 13:00:08] code 501, message Unsupported method ('OPTIONS')
127.0.0.1 - - [24/Jan/2025 13:00:08] "OPTIONS / HTTP/1.1" 501 -
127.0.0.1 - - [24/Jan/2025 13:00:08] code 501, message Unsupported method ('OPTIONS')
127.0.0.1 - - [24/Jan/2025 13:00:08] "OPTIONS / HTTP/1.1" 501 -
127.0.0.1 - - [24/Jan/2025 13:00:08] "GET http://www.google.com HTTP/1.0" 404 -
127.0.0.1 - - [24/Jan/2025 13:00:08] code 400, message Bad request version ("GZn©ßßH\\x8e\\
x157Åø²\\x00\\x9c\\x13\\x02\\x13\\x03\\x13\\x01\\x003\\x009\\x005\\x00/À,À0\\x00£\\x00\\x9f
Ì©Ì¨ÌªÀ¯À\\xadÀ£À\\x9fÀ]ÀaÀWÀSÀ+À/\\x00¢\\x00\\x9eÀ®À¬À¢À\\x9eÀ\\\\À`ÀVÀRÀ$À(\\x00k\\x00jÀs
Àw\\x00Ä\\x00ÃÀ#À'\\x00g\\x00@ÀrÀv\\x00¾\\x00½À")
127.0.0.1 - - [24/Jan/2025 13:00:08] "\x16\x03\x01\x02\x00\x01\x00\x01ü\x03\x03\x84Ë>4\x1dØ
½v\x9c\x9d=\x9fT4n¸|Bp7X\x0bZ\x9a¦:/\x0fÊ\x1dÕ\x17 Rm\x17SåóYè¡rÚ\x87V0`> \x06\x85GZn©ßßH\x
8e\x157Åø²\x00\x9c\x13\x02\x13\x03\x13\x01\x003\x009\x005\x00/À,À0\x00£\x00\x9fÌ©Ì¨ÌªÀ¯À À£
À\x9fÀ]ÀaÀWÀSÀ+À/\x00¢\x00\x9eÀ®À¬À¢À\x9eÀ\\À`ÀVÀRÀ$À(\x00k\x00jÀsÀw\x00Ä\x00ÃÀ#À'\x00g\x00
@ÀrÀv\x00¾\x00½À" 400 -
127.0.0.1 - - [24/Jan/2025 13:00:08] code 400, message Bad request version ('¢0{áÎRêI\\x01v
Þ3EÊDf\\x8ap¼_|Gù\\x14\\x97')
127.0.0.1 - - [24/Jan/2025 13:00:08] "\x16\x03\x01\x02\x00\x01\x00\x01ü\x03\x03{\x01Ö\x81\x
9a\x90¾\x1e³=o\x95¯q=}Û=e\x19\x0bØ#ÿ+±U#gíÛÅ \x9fQ\x09¢0{áÎRêI\x01vÞ3EÊDf\x8ap¼_|Gù\x14\x97
" 400 -
127.0.0.1 - - [24/Jan/2025 13:00:08] code 501, message Unsupported method ('OPTIONS')
127.0.0.1 - - [24/Jan/2025 13:00:08] "OPTIONS / HTTP/1.1" 501 -
127.0.0.1 - - [24/Jan/2025 13:00:08] "GET /HNAP1 HTTP/1.1" 404 -
127.0.0.1 - - [24/Jan/2025 13:00:08] "GET /nmaplowercheck1737745207 HTTP/1.1" 404 -
127.0.0.1 - - [24/Jan/2025 13:00:08] code 501, message Unsupported method ('HEAD')
127.0.0.1 - - [24/Jan/2025 13:00:08] "HEAD http://www.google.com HTTP/1.0" 501 -
127.0.0.1 - - [24/Jan/2025 13:00:08] code 501, message Unsupported method ('OPTIONS')
127.0.0.1 - - [24/Jan/2025 13:00:08] "OPTIONS / HTTP/1.1" 501 -
127.0.0.1 - - [24/Jan/2025 13:00:08] code 501, message Unsupported method ('CONNECT')
127.0.0.1 - - [24/Jan/2025 13:00:08] "CONNECT www.google.com:80 HTTP/1.0" 501 -
127.0.0.1 - - [24/Jan/2025 13:00:08] "GET /evox/about HTTP/1.1" 404 -
127.0.0.1 - - [24/Jan/2025 13:00:08] code 501, message Unsupported method ('OPTIONS')
127.0.0.1 - - [24/Jan/2025 13:00:08] "OPTIONS / HTTP/1.1" 501 -
127.0.0.1 - - [24/Jan/2025 13:00:08] code 501, message Unsupported method ('OPTIONS')
127.0.0.1 - - [24/Jan/2025 13:00:08] "OPTIONS / HTTP/1.1" 501 -
127.0.0.1 - - [24/Jan/2025 13:00:08] "GET / HTTP/1.0" 404 -
127.0.0.1 - - [24/Jan/2025 13:00:08] "GET / HTTP/1.1" 404 -
```

```
user@user-VirtualBox:~/Documents/csce_413/PoC_2$ nmap -A -T4 127.0.0.1
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-01-24 13:00 CST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000043s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT     STATE SERVICE VERSION
631/tcp  open  ipp     CUPS 2.4
| http-robots.txt: 1 disallowed entry
|_/
|_http-server-header: CUPS/2.4 IPP/2.1
|_http-title: Home - CUPS 2.4.7
8080/tcp open  http    BaseHTTPServer 0.6 (Python 3.12.3)
|_http-title: Site doesn't have a title.
|_http-server-header: BaseHTTP/0.6 Python/3.12.3

Service detection performed. Please report any incorrect results at https://nmap.org/submit
/ .
Nmap done: 1 IP address (1 host up) scanned in 7.28 seconds
user@user-VirtualBox:~/Documents/csce_413/PoC_2$
```

It is seen that NMAP has correctly identified that HTTP server is running on port 8080 and that it is being run through Python.