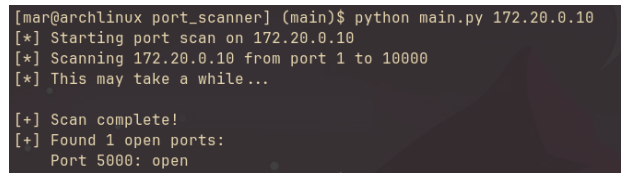# Executive Summary

Part 1: Reconnaissance

Using the initial starter code to verify socket and docker networking working properly, I was able to discover port 5000 on `172.20.0.10` was open.
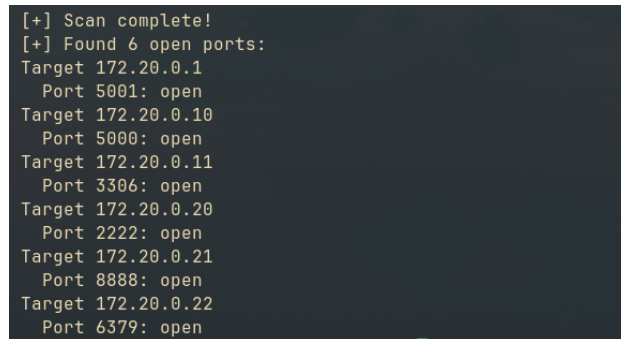

Figure 1: Basic socket test to find port 5000

After implementing some input handling with `argparse`, CIDR handling with `ipaddress`, and threading, I ran `python main.py --target 172.10.0.0/24 --ports 1-10000 --threads 10000` and got the results below.

Figure 2: Some open ports on the `172.10.0.0/24`

Implementing some level of banner grabbing by going through some common probing techinques onto the same subnet and ports on Figure 3. This is ran wit all ports `1-65535` to get all ports I'm not certain why `172.20.0.1:2222` appears to allow a socket connection now when Figure 2 doesn't have it open. Regardless, based on these banners, the services for these ports are

- `172.20.0.1:5001` - `http`

- `172.20.0.10:5000` - `http`

- `172.20.0.11:3306` - `mysql`

- `172.20.0.11:33060` - Unknown

- `172.20.0.20:2222` - `ssh`

- `172.20.0.21:8888` - `http`

- `172.20.0.22:6379` - `telnet`

where the last one knowledge that in `telnet`, `get` is a command and can verify by trying to connect via `telnet`, seen in Figure 4.

```
[+] Scan complete!
[+] Found 8 open ports:
Target 172.20.0.1
  Port 5001: open
    Banner: HTTP/1.1 200 OK
Server: Werkzeug/3.1.5 Python/3.11.14
Date: Sat, 07 Feb 2026 21:58:09 GMT
Content-Type: text/html; charset=ut
  Port 2222: open
Target 172.20.0.10
  Port 5000: open
    Banner: HTTP/1.1 200 OK
Server: Werkzeug/3.1.5 Python/3.11.14
Date: Sat, 07 Feb 2026 21:58:52 GMT
Content-Type: text/html; charset=ut
Target 172.20.0.11
  Port 3306: open
    Banner: J
8.0.45Zg)At0QW1!7|gUmysql_native_password
  Port 33060: open
    Banner:
        ?
Target 172.20.0.20
  Port 2222: open
    Banner: SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.13
Target 172.20.0.21
  Port 8888: open
    Banner: HTTP/1.1 200 OK
Server: Werkzeug/3.1.5 Python/3.11.14
Date: Sat, 07 Feb 2026 21:59:46 GMT
Content-Type: application/json
Con
Target 172.20.0.22
  Port 6379: open
    Banner: -ERR wrong number of arguments for 'get' command
```

Figure 3: Banner grabbing on `172.10.0.0/24`


```
[mar@archlinux csce413_assignment2] (main)$ telnet 172.20.0.22 6379
Trying 172.20.0.22...
Connected to 172.20.0.22.
Escape character is '^]'.
```

Figure 4: Telnet connection on `172.10.0.22:6379`

Connecting to `172.20.0.11:33060` with `nc`, all I get is the `?` as seen in the banner and no other information. I was not able to determine what this service could possibly be.

Accessing the SSH server, credentials are shown when connecting with `ssh`, I was able to read out the flag `FLAG{h1dd3n_s3rv1c3s_n33d_pr0t3ct10n}`.

Viewing the website at `172.20.0.10:5000`, it suggests going to the route `/api/secrets`, which returns the flag `FLAG{n3tw0rk_tr4ff1c_1s_n0t_s3cur3}`. It notes this is the api token.

Curling these http services, at `172.20.0.21:8888`, it appears to be some sort of api route. Here it says that there's a flag route that also needs a token with hint to intercept network traffic, which is likely refering the the flag I got noted to be api token. Based on this description, it likely for part 2 regarding analyzing network traffic.
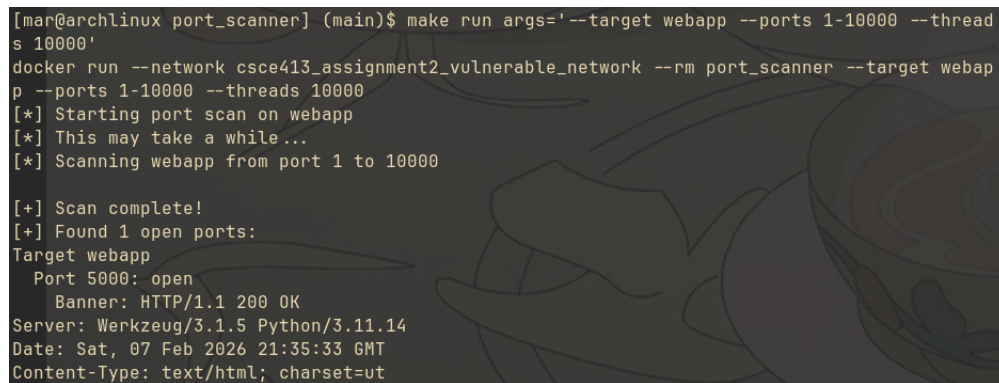
```json
{
  "authentication": {
    "alternative": "?token=<token> query parameter",
    "header": "Authorization: Bearer <token>",
    "hint": "The token can be found by intercepting network traffic...",
    "type": "Bearer token"
  },
  "endpoints": [
    {
      "description": "API information",
      "method": "GET",
      "path": "/"
    },
```

```
    {
      "description": "Health check",
      "method": "GET",
      "path": "/health"
    },
    {
      "description": "Get flag (requires authentication)",
      "method": "GET",
      "path": "/flag"
    },
    {
      "description": "Get secret data (requires authentication)",
      "method": "GET",
      "path": "/data"
    }
  ],
  "message": "This is a hidden API service. Authentication required.",
  "port": 8888,
  "service": "Secret API Server",
  "status": "running",
  "version": "1.0"
}
```

To show that domain names work, I set up running the python script within a docker container so it can resolve the domain names, which are the container names in the network. This can be seen in Figure 5.

Figure 5: Domain name resolution of `webapp`

To summarize,

- `172.20.0.1:5001` - `http` using Werkzeug/3.1.5 with Python 3.11.14 (Flask server)

- `172.20.0.10:5000` - `http` using Werkzeug/3.1.5 with Python 3.11.14 (Flask server). Has corresponding flag `FLAG{n3tw0rk_tr4ff1c_1s_n0t_s3cur3}`

- `172.20.0.11:3306` - `mysql`

- `172.20.0.11:33060` - Unknown

- `172.20.0.20:2222` - `ssh` with OpenSSH 8.9 on Ubuntu. Has corresponding flag `FLAG{h1dd3n_s3rv1c3s_n33d_pr0t3ct`

- `172.20.0.21:8888` - `http` using Werkzeug/3.1.5 with Python 3.11.14 (Flask server)

- `172.20.0.22:6379` - `telnet`

Part 2: MITM Attack

As suggested by the instructions, I determine the network id of my docker network to be `286b60758bd5` and thus ran the command `sudo tcpdump -i br-286b60758bd5 -A -s 0 'port 3306'`. This command listens to this bridge and looks for packets with information about `port 3306`, which from recon only the mysql server has open. Running this and looking at the website on `localhost:5001`, I went to the api routes `/api/users` and `/api/secrets` and can see that these packets are being sent in plaintext, as seen in Figure 6. Log can be seen in `mitm/tcpdump_mysql.txt`. As intended in this assignment, the flags are consider
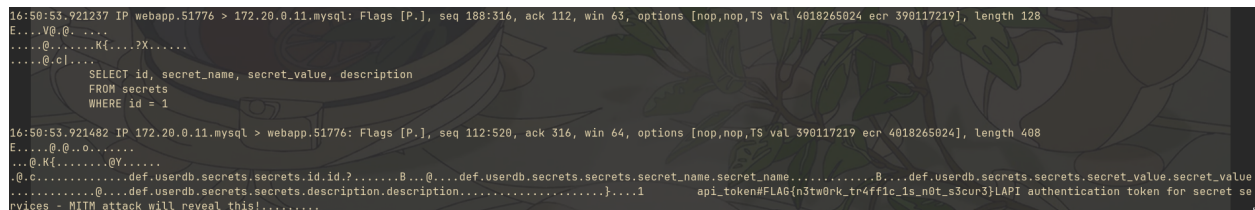

Figure 6: Some of the tcpdump packets caught

sensitive and thus can obtain sensitive data through packet sniffing.

To use this in conjugation with the info obtained in recon to get the third flag, I constructed a curl command with credentials to query for the flag.

```
curl 172.20.0.21:8888/flag \
  -H "Authorization: Bearer FLAG{n3tw0rk_tr4ff1c_1s_n0t_s3cur3}"
```

Obtaining the flag, `FLAG{p0rt_kn0ck1ng_4nd_h0n3yp0ts_s4v3_th3_d4y}`.

Discussion on Real World Impacts

Plaintext traffic is generally considered not secure and same here with MySQL traffic being plaintext. This can allow people to do this packet sniffing/MITM attack to obtain information about the database through the queries such as table/column names. Similarly, the results of the query can be sensitive such as social security numbers (possibly only masked), hashed passwords, FERPA data like grades, HIPPA like medical data, etc.

Part 3: Security Fixes

Port Knocking

Honeypot

Remediation Recommendations

Conclusion