Open Services

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

Theory:

DNSSEC

DNSSEC improves on authentication in DNS by using digital signatures based on public key cryptography. In DNSSEC, DNS queries are not the things that are encrypted/decrypted, but rather the DNS data itself is signed by the owner of the data. DNSSEC allows for two main improvements on DNS:

- Allows for a resolver to cryptographically ensure that the data they receive actually came from where they requested it from. This is called data origin authentication.
- Allows for a resolver to know that data hasn't been modified while in transport. This is called data integrity protection.

DNS over HTTPS

DNS over HTTPS (DoH) essentially improves on security in DNS embedding DNS in HTTP and sending requests via TLS over TCP. This means that DoH requests are essentially the same as HTTPS requests, meaning that the requests themselves are encrypted. DoH allows for a couple of improvements on DNS:

- Encryption of DNS queries, making them unreadable to third parties who are monitoring internet traffic.
- DoH allows DNS queries to blend in with other HTTPS requests, as they are essentially the same from a mitm perspective.

Differences

- DNSSEC adds data to DNS responses in order to verify authenticity and integrity.
- DoH encrypts DNS communication using HTTPS.
- DNSSEC can make DNS requests take much longer due to the added payload to confirm authenticity and integrity.
- To implement DoH, you need to configure your device to use a specific DNS resolver that supports DoH.
- DNSSEC can be implemented by simply enabling it with your DNS provider.

Practice:

Environment setup:

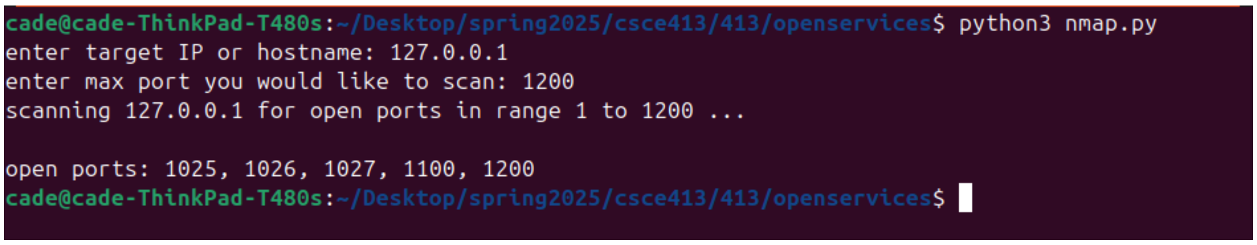
- Ensure python is installed

Running the POC:

- Create 2 terminal tabs
- In one tab, run command: python3 server.py
- In other tab, run command: python3 nmap.py
    - Input 127.0.0.1 as ip to check
    - Input 1200 as max port to check

POC analysis

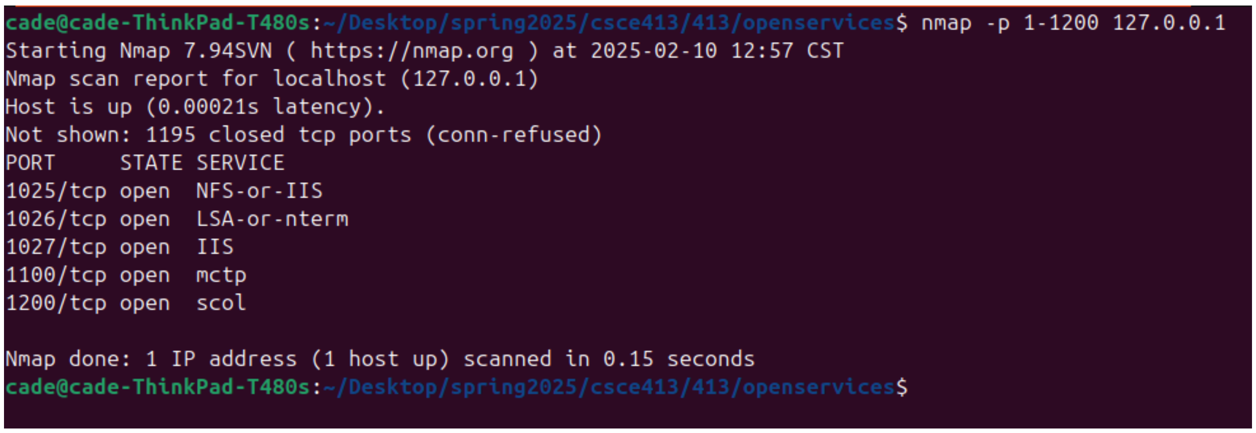
My implementation results:



```
cade@cade-ThinkPad-T480s:~/Desktop/spring2025/csce413/413/openservices$ python3 nmap.py
enter target IP or hostname: 127.0.0.1
enter max port you would like to scan: 1200
scanning 127.0.0.1 for open ports in range 1 to 1200 ...

open ports: 1025, 1026, 1027, 1100, 1200
cade@cade-ThinkPad-T480s:~/Desktop/spring2025/csce413/413/openservices$
```

Nmap results:



```
cade@cade-ThinkPad-T480s:~/Desktop/spring2025/csce413/413/openservices$ nmap -p 1-1200 127.0.0.1
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-02-10 12:57 CST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00021s latency).
Not shown: 1195 closed tcp ports (conn-refused)
PORT     STATE SERVICE
1025/tcp open  NFS-or-IIS
1026/tcp open  LSA-or-nterm
1027/tcp open  IIS
1100/tcp open  mctp
1200/tcp open  scol

Nmap done: 1 IP address (1 host up) scanned in 0.15 seconds
cade@cade-ThinkPad-T480s:~/Desktop/spring2025/csce413/413/openservices$
```

The main difference between these two scans is the fact that nmap provided what service each port that is open is typically used for, even though they are not used for those in this case. However, these two tools show the same five ports as open. It should be noted that I only scanned for openings in ports 1-1200, there may in fact be more ports open on my machine that I am not picking up on, but those are not necessary for the purpose of this assignment.