Graham Dungan
February 6, 2025

CSCE 413: Software Security
PoC 7: Open Services

Theory

Explanation of DNSSEC & DNS over HTTPS (DoH) (20 Points)

DNSSEC was created using public key cryptography and digital signatures to fix the implicit lack of trust that DNS depends on. DNSSEC requires DNS zones to have unique public/private key pairs. Each zone publishes its public keys and parent zones sign these, forming a continuous chain of trust from a central authority (authoritative name server). Now, when a user attempts to access a domain, they also receive what is essentially a certificate authenticating the identity of the server in communication.

DNS sends queries in plaintext across the internet, violating the confidentiality of information. To fix this, DoH (DNS over HTTP) encrypts information before it is sent to some secure DNS server that is operated by a trusted partner. This can mitigate some man-in-the-middle attacks, where a third party can intercept and view messages.

DNSSEC differs from DoH as DNSSEC is structured around the concept of authenticating and validating the identity of parties on the internet, not necessarily protecting the transport of data. DoH is built on top of HTTP, and focuses on securing data, not necessarily validating identification.

Analysis of Security Threats and Mitigations (10 Points)

Common threats that can be mitigated by DNSSEC include DNS spoofing and MITM attacks, both of which can be used to perform a multi-stage attack. DNS spoofing is when attackers can inject false DNS records into another server's cache such that they can reroute users to malicious websites. MITM attacks can be followed by DNS spoofing, as all that needs to be done is finding some means of intercepting DNS traffic. DNSSEC can prevent these issues by verifying digital signatures (mitigating spoofing) and, since DNS records are signed, forged responses from MITM attackers would fail to verify their identities.

A DNS spoofing attack was executed in 2018 against a cryptocurrency wallet called MyEtherWallet. Attackers hijacked an AWS DNS server and rerouted requests to a similar-looking phishing website with false credentials. DNSSEC could have been used in this instance to flag the false credentials.

Common threats that can be mitigated by DoH include DNS sniffing, MITM attacks, and DNS blocking. DNS sniffing is the most simple vulnerability of DNS- since messages are sent as plaintext, any server in the middle of communication can view traffic. Since these messages are sent in plaintext, they can also be manipulated quite easily. DoH solves these issues through encryption- hiding data in ciphertext and providing means of repudiation. In the same way that ISPs can perform DNS sniffing to view unencrypted data, ISPs can block certain domains, known as DOS blocking. Reasons for doing this include censorship, throttling (which can be used as a means of censorship), or blocking malicious/dangerous websites. Since DoH is encrypted, DoH queries can avoid DNS blocking.

A real-world example of DoH being used to circumvent DNS blocking is the ChamelDoH implant using DoH to communicate with attacker's servers. The implant uses DoH as a means of reaching command-and-control servers, giving the ability to remote access vulnerable machines.

Pros and Cons of DNSSEC & DoH Adoption (10 Points)

As listed above, DNSSEC provides features that prevent DNS spoofing and MITM attacks. It allows for a chain of trust in DNS responses and establishes central authorities for communicating on the internet. Cons include the difficult management and storage of keys. DNSSEC also does not natively support encryption, adds extra steps for querying incurring a more expensive overhead, and the effectiveness of DNSSEC still depends on the wider adoption of the system.

DoH provides security against DNS sniffing, MITM attacks, and DNS blocking. These features work to contribute to user privacy in public networks and does not require broad changes to end-user programs. However, while secure, DoH adoption is dependent upon larger providers (such as Google, Cloudfare, Mozilla), it bypasses common ISP security controls, and increases overhead, much like DNSSEC.

Practice

Development of Custom NMAP-like Tool (30 Points)

For this assignment, I created a Python program called `portscanner.py`. This application accepts an address and range of ports as input, attempts to make TCP requests for all ports within the range to the specified address, and relays if a successful connection has been made.

Given that I do not have much experience with networking and Python, I used Geeks For Geek's Port Scanner using Python as a reference for how I would construct the program. What follows is a code snippet forming the core functionality of the program,

```
for port in range(lower_port,higher_port):
    # Create a TCP socket using IPv4
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Create a 1 second timeout for the new socket
    socket.setdefaulttimeout(0.5)
    # Attempt to establish a TCP connection
    result = s.connect_ex((address,port))

    if result == 0:
        print(f'Port {port} is open')
    s.close()
```

This code snippet forms a for loop for each port between the given port numbers. Using the socket library for Python, it creates a TCP socket to be used in the future request to the specified port. It also has a timeout set to 0.5 seconds such that the attempted connections for each port do not take up too much time. A connection is then attempted, and if the result is zero, then a successful connection had been made.

Comparison with Standard NMAP Tool (20 Points)

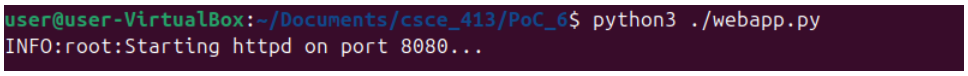
A file named `demo.sh` has been provided for running a comparison between Nmap and the custom port scanner. To run this script;

1. Enter the `PoC7` directory. `cd PoC7`
2. Run the `demo.sh` script. `./demo.sh`

This script simply runs the test webapp `webapp.py` on port 8080, runs the Nmap and custom port scanner programs, and displays their output for comparison. What follows is a manual demonstration and explanation of the port scanning and output.

1. **Start Python Webapp**

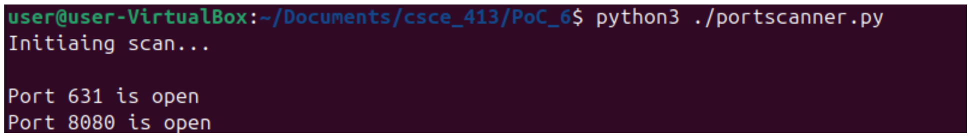
   We can start the python Webapp by simply running `python3 ./webapp.py`. The server will then run on port 8080.



```
user@user-VirtualBox:~/Documents/csce_413/PoC_6$ python3 ./webapp.py
INFO:root:Starting httpd on port 8080...
```

2. **Run Custom Port Scanner**

   Once the Python server is running, the custom port scanner can be ran with `python3 ./portscanner.py`. The port scanner has custom arguments `-a` for specifying an address, and `-p` for specifying a range of ports. These values default to localhost `127.0.0.1` and all ports `1-65535`.



```
user@user-VirtualBox:~/Documents/csce_413/PoC_6$ python3 ./portscanner.py
Initiaing scan...

Port 631 is open
Port 8080 is open
```

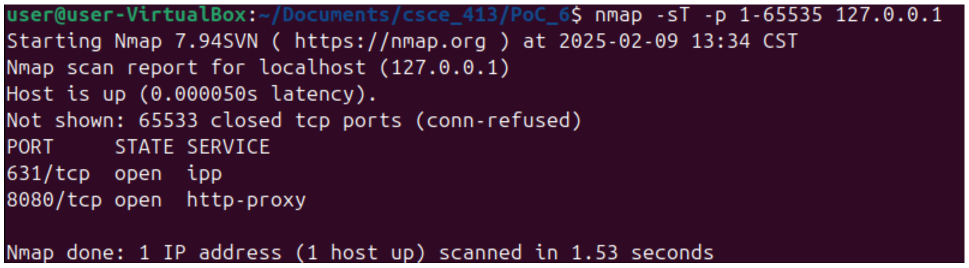
   As seen in this screenshot, the port scanner identified the webserver hosted on port 8080, as well as the port for Internet Printing Protocol, 631.

3. **Run Nmap**

   To compare, we can now run Nmap. Nmap will be run with the settings:

   `nmap -sT -p 1-65535 127.0.0.1`

   The `-sT` argument forces Nmap to perform a full TCP handshake just as the Python program does with the connect statement, as seen on line 7 of `portscanner.py`. The `-p` argument specifies ports just like the Python program defaults to, similarly with the localhost address of `127.0.0.1`.

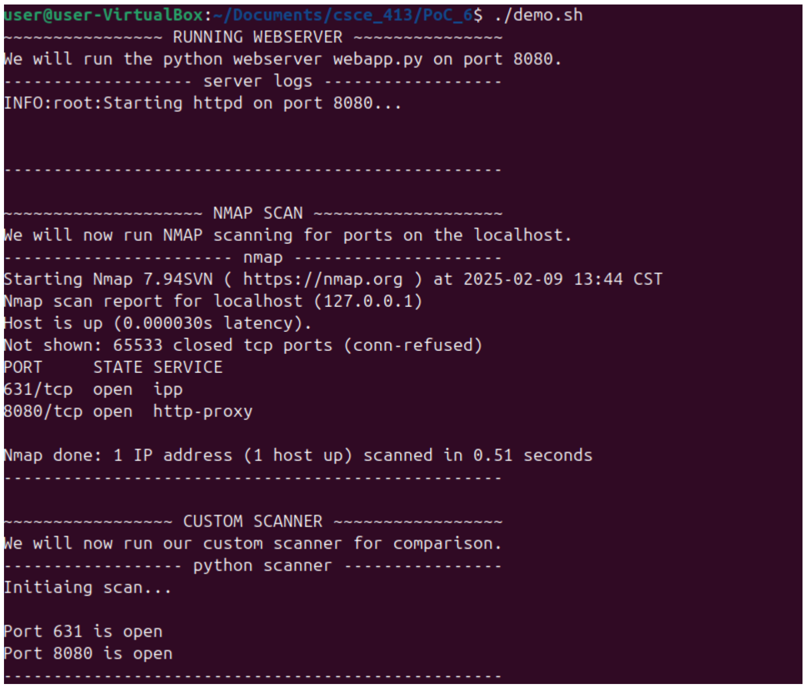


```
user@user-VirtualBox:~/Documents/csce_413/PoC_6$ nmap -sT -p 1-65535 127.0.0.1
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-02-09 13:34 CST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000050s latency).
Not shown: 65533 closed tcp ports (conn-refused)
PORT     STATE SERVICE
631/tcp  open  ipp
8080/tcp open  http-proxy

Nmap done: 1 IP address (1 host up) scanned in 1.53 seconds
```

   It is seen that Nmap has also identified the same ports, alongside the services that they provide.

It is seen that a similar demonstration can be ran with the `demo.sh` script,



```
user@user-VirtualBox:~/Documents/csce_413/PoC_6$ ./demo.sh
~~~~~~~~~~~~~~~~ RUNNING WEBSERVER ~~~~~~~~~~~~~~~
We will run the python webserver webapp.py on port 8080.
------------------ server logs -----------------
INFO:root:Starting httpd on port 8080...


--------------------------------------------------

~~~~~~~~~~~~~~~~~~~ NMAP SCAN ~~~~~~~~~~~~~~~~~~
We will now run NMAP scanning for ports on the localhost.
---------------------- nmap --------------------
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-02-09 13:44 CST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000030s latency).
Not shown: 65533 closed tcp ports (conn-refused)
PORT     STATE SERVICE
631/tcp  open  ipp
8080/tcp open  http-proxy

Nmap done: 1 IP address (1 host up) scanned in 0.51 seconds
--------------------------------------------------

~~~~~~~~~~~~~~~~ CUSTOM SCANNER ~~~~~~~~~~~~~~~~
We will now run our custom scanner for comparison.
----------------- python scanner ---------------
Initiaing scan...

Port 631 is open
Port 8080 is open
--------------------------------------------------
```

Documentation (10 Points)

See above.

External Resources

- DNS Security: Threat Modeling DNSSEC, DoT, and DoH
- DNSSEC Overview
- DNSSEC – What Is It and Why Is It Important?
- What DNS over HTTPS (DoH) Is and How to Enable in Windows 10