Lab: Applying Cryptography

Jaiden Shoel, April 18th, 2025

Introduction and Materials

The purpose of this lab was to simulate applying Cryptography and to see how it can be used to protect sensitive data and secure web applications. Specifically, we implemented HTTPS on a locally hosted website using OpenSSL to generate SSL/TLS certificates, then configured nginx to serve the site over an encrypted connection. This allowed us to see firsthand how encrypted traffic differs from unencrypted traffic when viewed in a man-in-the-middle (MiTM) attack scenario.

While I didn't complete the lab all the way through on Wednesday (my classmates next to me were having some technical issues), I managed to get to Part 2 (Testing our cryptography), specifically the Wireshark part, so I was content with that, along with the fact that Wyatt stated it was no problem if we weren't able to complete the lab all the way through at the end of class.

Materials

- OpenSSL: to generate a private key, a certificate signing request (CSR), and a self-signed certificate
- Terminal
- VS Code & Docker Desktop
    - Docker and docker-compose: to run our web application and nginx server in isolated containers
    - nginx: to configure HTTPS and serve the web app securely
- Wireshark: to inspect the network traffic before and after encryption

Steps to Reproduce

In a perfect world where there were no technical difficulties the steps to complete the lab, in short, would be:

1. **Install OpenSSL**
    a. In terms of macOS, it is already good to go since OpenSSL is already built in.

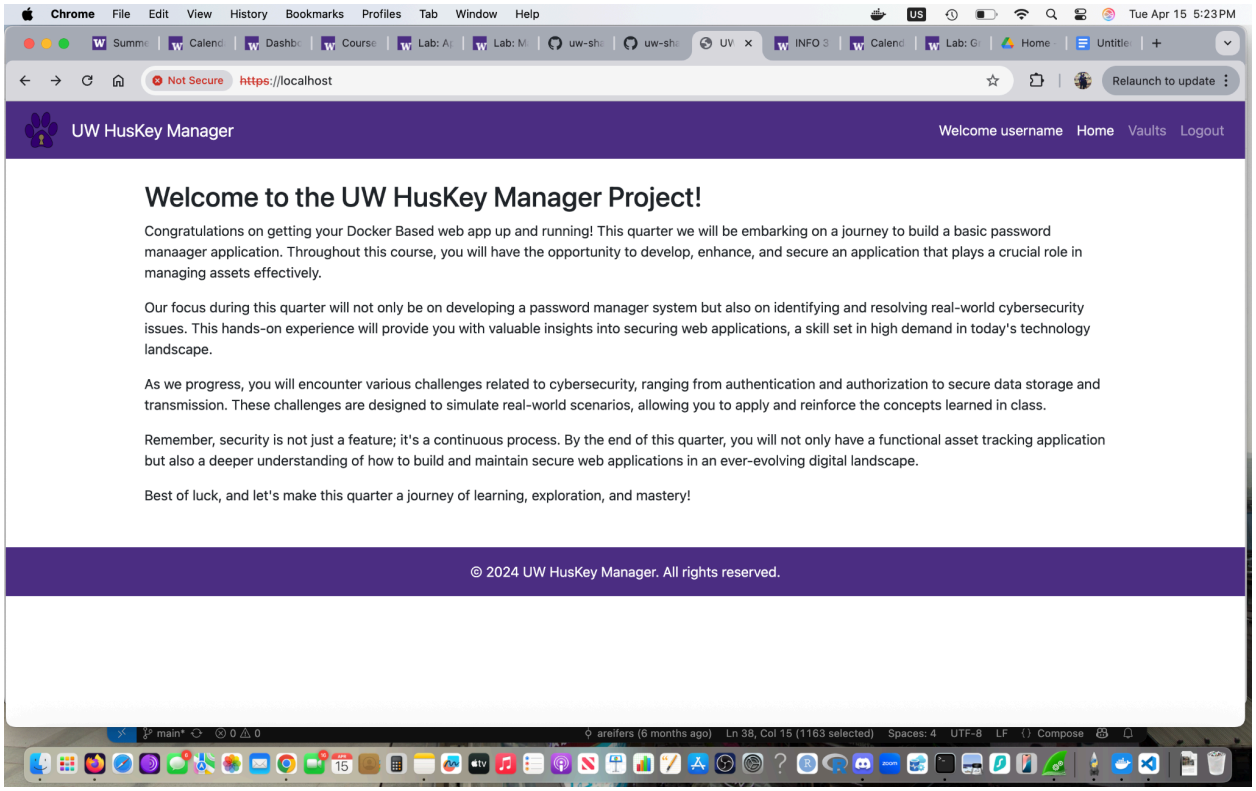
2. **Navigate to your working directory**
    a. Go to your project directory in Terminal (MacOS)

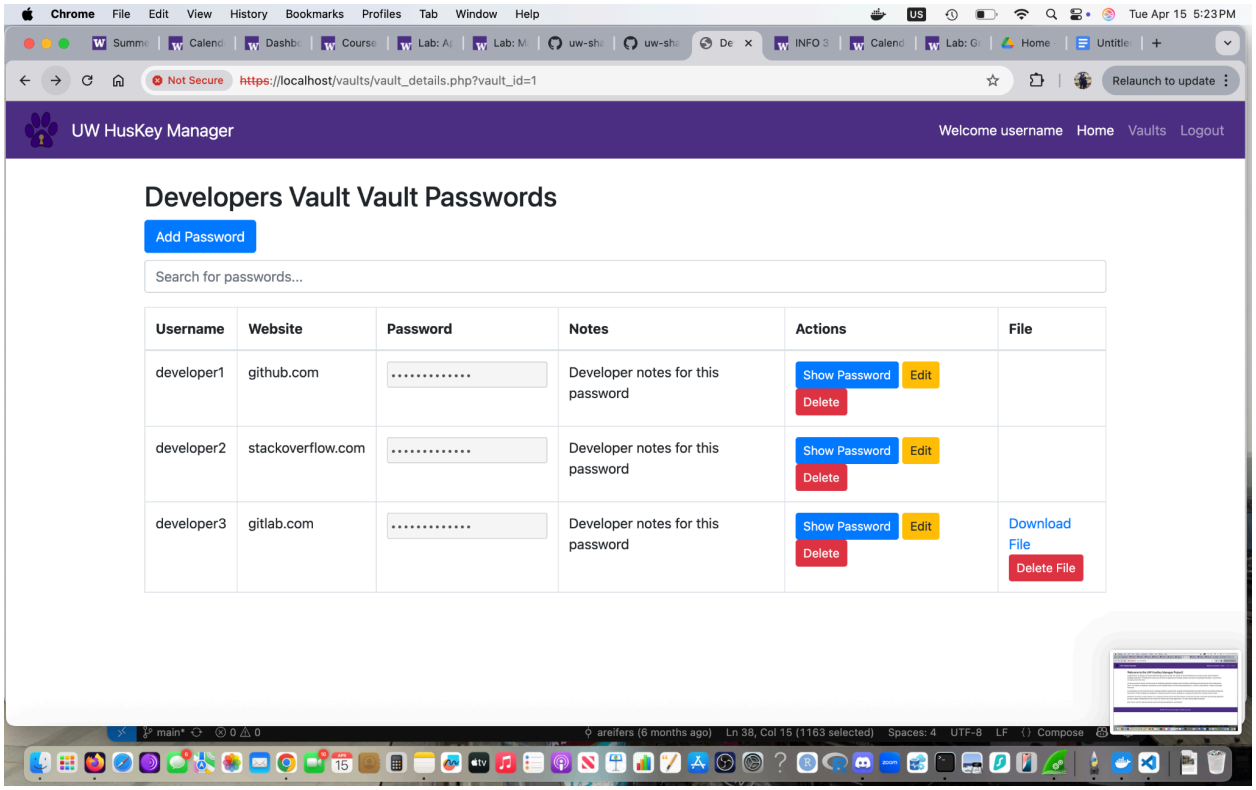
3. **Generate your certificate and key**
    a. Run OpenSSLs commands to create a private key and CSR (see README.txt at https://github.com/uw-shanemiller/spr-25-uw-cybersec-huskey-manager/tree/W3-Encryption)
4. **Set up HTTPS in nginx (In VSCode)**

a. In the `docker-compose.yaml`, map your certifications and key like this:
volumes:
   - ./certs/localhost.crt:/etc/nginx/ssl/localhost.crt
   - ./certs/localhost.key:/etc/nginx/ssl/localhost.key
○ Change the port mapping to "443:443" and update your nginx-default.conf:

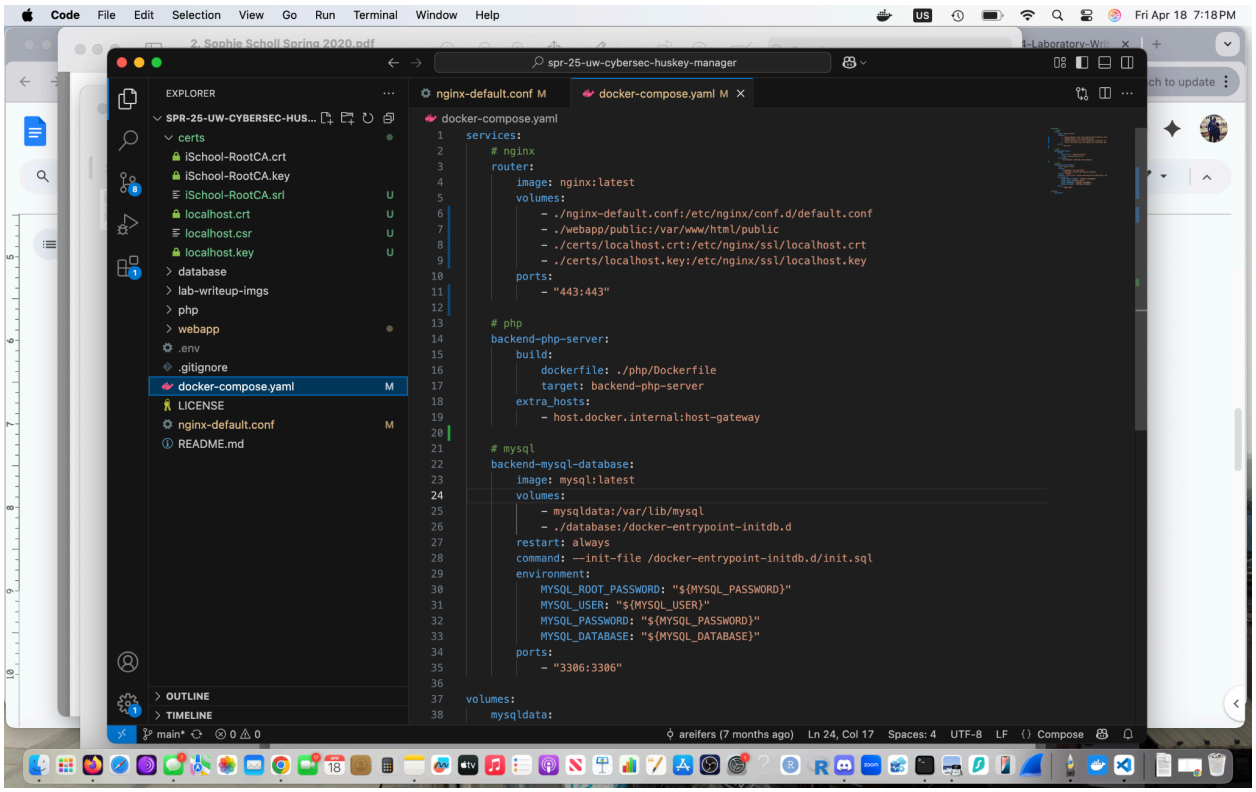
5. **Redeploy the Docker container and verify HTTPS is working**
   a. Access the app via https://localhost:443. A browser warning will appear (since the CA isn't trusted), but just bypass and proceed manually.
   b. Confirm that the application is using encryption by checking the network traffic in browser dev tools (inspect element) or with Wireshark.
6. **Re-run the MITM attack**

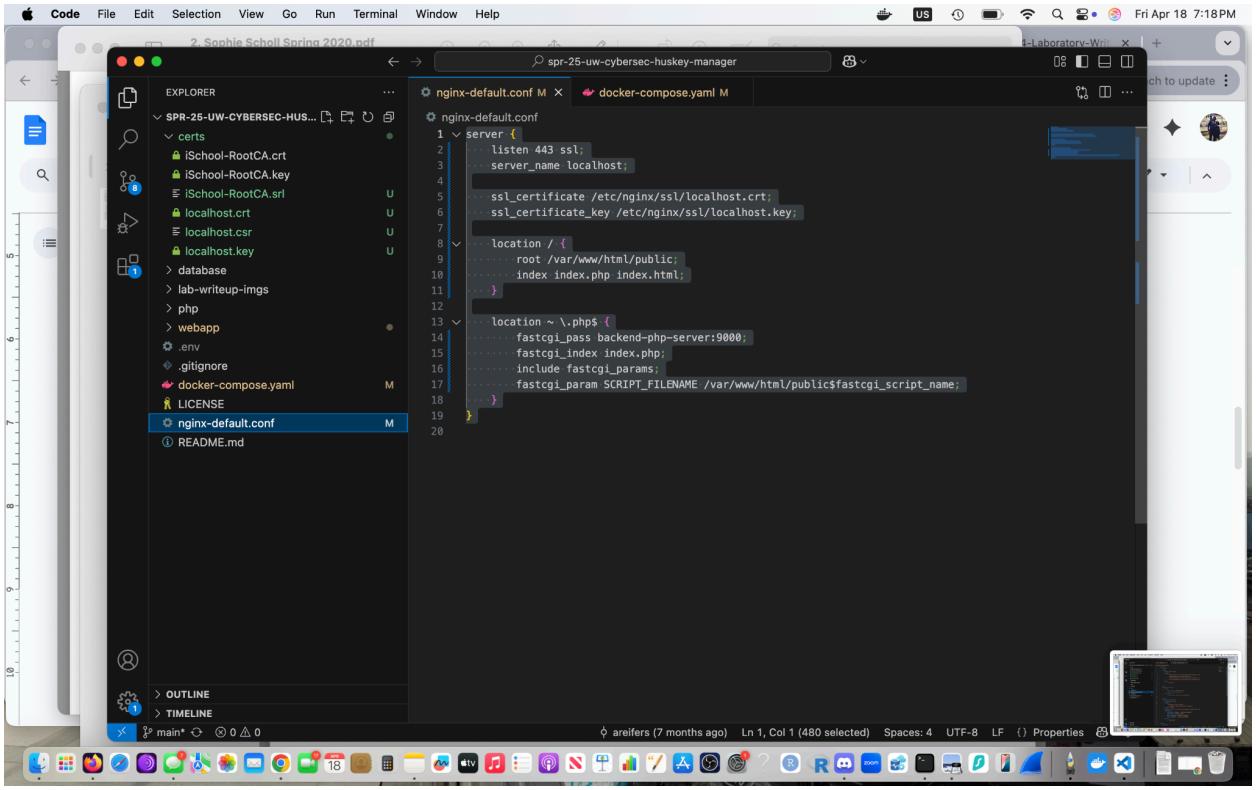
Results of completing Step 4 in Part 1 (Got into the localhost):



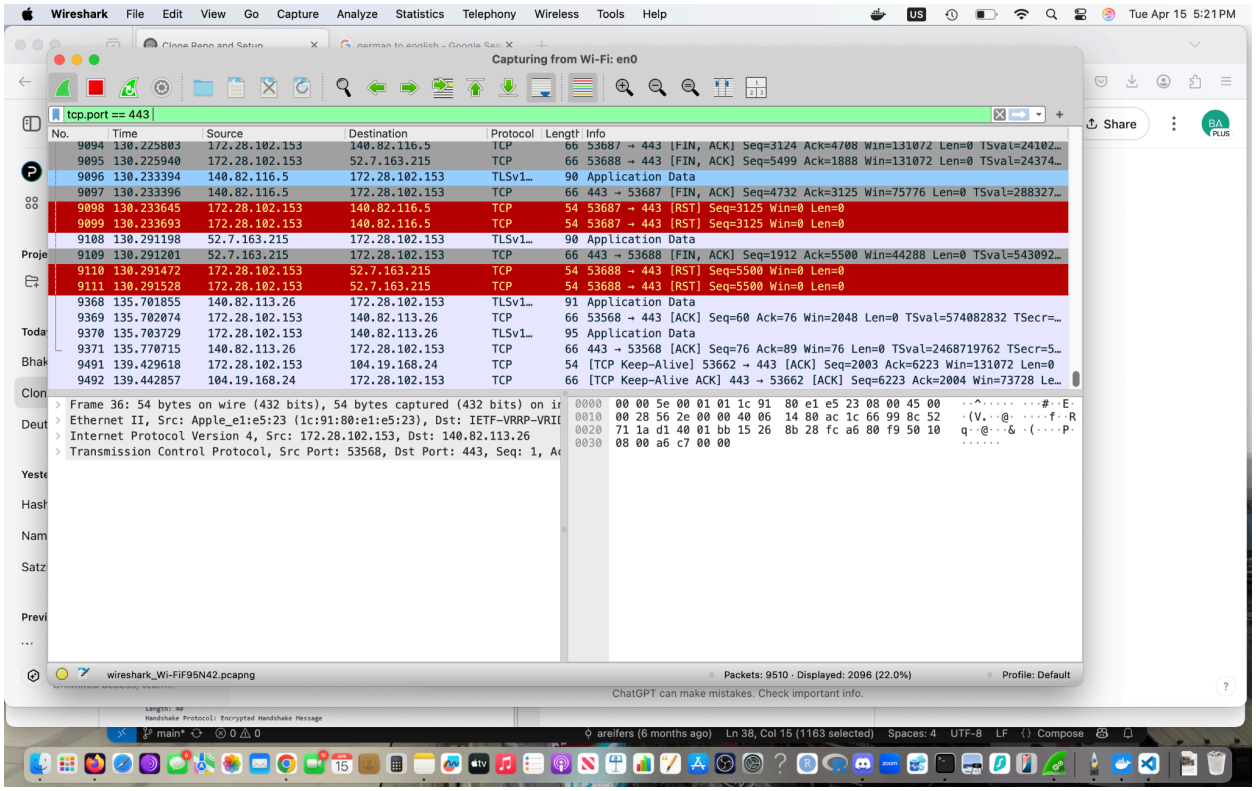


Screenshots of my VSCode, which allowed me to access the HTTPS https://localhost:443:




Code File Edit Selection View Go Run Terminal Window Help
Fri Apr 18 7:18 PM
spr-25-uw-cybersec-huskey-manager
EXPLORER
SPR-25-UW-CYBERSEC-HUS...
certs
iSchool-RootCA.crt
iSchool-RootCA.key
iSchool-RootCA.srl
localhost.crt
localhost.csr
localhost.key
database
lab-writeup-imgs
php
webapp
.env
.gitignore
docker-compose.yaml
LICENSE
nginx-default.conf
README.md
OUTLINE
TIMELINE
nginx-default.conf M
docker-compose.yaml M
nginx-default.conf
server {
listen 443 ssl;
server_name localhost;
ssl_certificate /etc/nginx/ssl/localhost.crt;
ssl_certificate_key /etc/nginx/ssl/localhost.key;
location / {
root /var/www/html/public;
index index.php index.html;
}
location ~ \.php$ {
fastcgi_pass backend-php-server:9000;
fastcgi_index index.php;
include fastcgi_params;
fastcgi_param SCRIPT_FILENAME /var/www/html/public$fastcgi_script_name;
}
}
main*
areifers (7 months ago)
Ln 1, Col 1 (480 selected)
Spaces: 4
UTF-8
LF
Properties

How far I got with Part 2 (was able to filter IP's with the command tcp.port == 443):



Class Principles

1. What layers of the OSI model remain in clear text following our encryption implementation?

Even after setting up HTTPS, I identified that not everything in our network traffic is hidden. The Network (Layer 3) and Transport (Layer 4) layers still remain visible. This includes IP addresses and port numbers, which routers and other networking devices need to read in order to forward traffic. It seemed like the actual encrypted data , like the web page content or form inputs, exististed at higher layers such as Application, Presentation, and Session layers which is where HTTPS applies its encryption according to what we learned.

2. Which CIA Properties are protected that were previously violated during the MITM attack?
Before implementing encryption, our traffic was completely exposed during a Man-In-The-Middle (MITM) attack. Switching to HTTPS helps protect:

1. **Confidentiality**, because any sensitive data being transmitted (like passwords or messages) is now encrypted and unreadable to attackers.

2. **Integrity**, because HTTPS includes mechanisms to detect tampering. If the data is changed in transit, the recipient would be able to tell something is wrong.

3. Why is encryption alone insufficient for establishing a fully trusted connection?

While encryption ensures data is hidden from eavesdroppers, it doesn't guarantee who you're communicating with. In this lab, we used our own self-signed certificate, something that browsers don't trust by default. That's why we still saw browser pop up warnings and we had to bypass it manually even though encryption was working. To truly establish trust, we need a recognized certificate authority (CA) to vouch for the website's identity. Without that layer of verification, an attacker could still set up an encrypted connection to a malicious site and trick users into thinking it's safe.

Hacker Mindset and Conclusion

The hacker mindset most definitely applies to this lab because it forced me to really engage with the systems and tools at a technical and conceptual level. In addition to following the steps in the README.txt, I also had to think about how HTTPS actually works, where its weaknesses are, and how an attacker might still try to break it. Questions of "what if I tried this instead?" or "what would happen if I didn't trust this certificate?" ultimately reflects the kind of contrarian and curious thinking that hackers rely on. It wasn't just about protecting data but it was also about understanding the capabilities that are engraved into security tools and how they could be tested or manipulated.

On the same token, to complete the lab, I think a mix of committed and creative thinking was required. Some of the terminal commands or config edits didn't work perfectly on the first try, so I had to troubleshoot, look things up, talk with my classmates and be patient while debugging small errors in

the docker-compose.yaml and the nginx-default.conf. From this, I thought the lab wasn't just technical persistence but was also about thinking flexibly and understanding the smallest misconfigurations can break the entire HTTPS setup. It actually reminded me of times when I was debugging my java codes in the cse classes I took. This process and time of debugging pushed me to not just follow instructions blindly, but to actually try to learn how and why things worked and why we were doing them.

Looking in the future, there are definitely creative ways to apply what I learned here. For example, understanding how certificate trust works could help and give me more perspective to build more secure web apps and educate users on certificate warnings. From a contrarian standpoint, it could involve intentionally misconfiguring certs or mimicking CA behavior in a controlled environment to test how systems or users respond. I know there's a specific position at companies where there job is to do exactly this called the Red Team. This lab gave me a better appreciation and recognition for the thin line between trust and risk in internet communication.

In terms of breaking or spoofing the cryptography we implemented, it's clear that while our data was encrypted, our certificate wasn't from a trusted authority and that's a big vulnerability. An attacker could do exactly what we did, create their own self-signed cert, run a fake site, and potentially trick users who ignore browser warnings. This shows that cryptography can be bypassed when trust isn't properly established, and that encryption alone doesn't guarantee security. This is why, as I stated in the class principles section, a valid cert from a real trusted CA and not just any certificate is essential to building strong secure systems.