COMP.SEC.300 Secure Programming: Exercise Work Report

Description

The exercise work for the course was done as a programming project. The program is, in its current state, used to share text content over the Tor protocol in a secure manner. It was written in Python, so a Python interpreter is required to run the program as well as a running Tor process on the computer.

Running the program is as simple as executing the main.py file. After the program is running successfully, it will display the Onion Service address. The Onion Service can be accessed with Tor Browser. The Onion Service also requires a private key that is provided when the program runs also. After the service is running, the program also awaits commands from users on the command line where the program was run.

Program commands: "help" to display available commands, "q" to quit, "u" to update displayed message on the web page.

The commands need to be typed on the command line where the program is running and then pressing Enter.

Program Structure

The program consists of the main entry point to the file called "main.py" as well as two classes Tor and Web. The Tor class handles communicating with the Tor process and Web handles the web server required to serve the content over Tor. The program also includes a templates folder for the HTML templates used by the web server.

Secure Programming Solutions

Frameworks & Libraries

The program utilizes established frameworks & libraries to handle important parts of the program such as serving web content and handling cryptographic keys. Cryptography was handled with a Python library called PyNaCl which implements a modern cryptographic library. In this program we used it to produce a public and private key used for client authentication on the Onion Service. Flask, on the other hand, was the web server that was used (although it may not be as critical in this case). However, it does bring one useful feature with itself: the Jinja2 templating engine. It ensures that content displayed on the web page of our Onion Service is escaped properly. This will prevent attacks like XSS.

Security headers & Content Security policy

The fact that content is being served to a web browser by the program, several security headers were used in HTTP responses. A CSP header was implemented also to control what kind of server is being loaded by the browser depending on its origin. There has been comments added to the code where these headers appear in the web.py file.

Controlling Tor Securely

Because the program controls the Tor process programmatically, some precautions were taken there as well. Cookie Authentication was enabled in the Tor process. This means that there is a cookie file on the computer which is needed to control the Tor process. Correct privileges are also required to read this file. If someone found a way to remotely control Tor, they could not do it. Tor listens to a specific port for controlling so this may be a useful security precaution. If a malicious actor had access to control Tor, they could break anonymity and likely cause much other harm.

Other security & privacy practices

The program also implemented security and privacy by other than the programming practices that were used.

Tor

The communication handled by the program is routed over the Tor Protocol. This ensures that the traffic is encrypted as well as provides various precautions to ensure that the visitor and the service itself are anonymous. This means that when a user accesses the service, their identity remains hidden and on top of that, the service itself is hidden; the user does not know which server they are connecting.

From the perspective of the service, so called Ephemeral Hidden Services are used to serve the content. This means that any files used by the service are not written to the disk, but they remain on the RAM. This further ensures that any content that was shared is not stored to be found later by someone.

Tor itself utilizes public and private keys to handle traffic encryption as well as authenticating the server. This itself provides security similar and in some ways better than certificate-based authentication used by regular websites. Utilizing this required no programming.

Client Authentication

Users accessing the Onion Service need to have a private key to authenticate themselves. As was mentioned previously, this private key is given at the start of the program. This prevents an actor who doesn't have right to access the service from connecting. When the service page is opened by Tor Browser, it automatically prompts the user for the key.

The Client Authentication uses x25519 Diffie-Hellman key exchange to establish the client authentication public and private keys. It utilizes elliptic curve cryptography (Curve25519).

Testing

The testing of the program was mostly done manually to ensure that cookies were set properly and that the service works as expected. In the end, it was not a complex program so testing this way was sufficient. Manual testing included testing that the Onion Service works (and that Tor Browser doesn't give any warnings), proper cookies were set in responses, client authentication works (and doesn't accept a wrong key).

Improvements & Potential new features

Improvements

Currently the program is used solely via the command line. This could be a problem in terms of leaking messages because messages entered on the command line such as program output will remain in the history.

An interesting idea could be utilizing Docker containers to run the Tor process and potentially the web server. This might help isolate potential problems like leaking information and isolate the service from the computer used to run the program. It would also allow configuring a separate Tor process.

Potential features

The program could be further improved to be able to share not just text but files also files for example. The initial idea was to implement this as well but due to some issues, it was not implemented yet.