Introduction

Welcome to Module 11! This week, we will learn how to create a secure web application. The major concept with regards to application security is that it is configured and not programmed within an application. SSL (Secure Socket Layer) is the protocol that enables developers to encrypt communications.

Topics and Learning Objectives

Topics

- Security, Security protocols
- Encryption, Decryption
- SSL, TLS
- HTTP vs HTTPS
- CSRF, cookies protection, HTTP header protection

Learning Objectives

By the end of the module, you should be able to:

1. Set up and configure a domain name to point to the web application.
2. Set up and configure SSL to protect web applications.
3. Secure web applications using user credentials (username and password).

Readings & Resources

Reading

Required

- Heroku Dev Center (2022).
    - Custom domain names for apps
    - Heroku SSL
- Wikipedia. (2022). Network address translation.
- Wikipedia. (2022). IPv6.

- Tcpdump. (2022). Man page of tcpdump.

Domain Name Systems

- DNS (Domain Name Systems) is the hierarchical and decentralized naming system used to identify computer reachable through the Internet or other Internet Protocol (IP) networks.
- The resource records contained in the DNS associate domain names with other forms of information. It is hard to remember an IP address of a server, but it is easy to remember a domain name.
- DNS turns domain names into IP addresses, which browsers use to load internet pages. Every device connected to the internet has its own IP address, which is used by other devices to locate the device.

Domain Name Registry

- A domain name registry is a database of all domain names and the associated registrant information in the top level domains of the DNS.
- Most domain name registries operate the top-level and second-level of the DNS.
- To register a domain name you need a service such as a domain name registry.
  - namecheap.com
  - aws.amazon.com
  - azure.com
  - heroku.com

SSL

- SSL stands for Secure Sockets Layer and, in short, it’s the standard technology for keeping an internet connection secure and safeguarding any sensitive data that is being sent between two systems, preventing criminals from reading and modifying any information transferred, including potential personal details.
- The two systems can be a server and a client (for example, a shopping website and browser) or server to server (for example, an application with personal identifiable information or with payroll information).
- It does this by making sure that any data transferred between users and sites, or between two systems remain impossible to read. It uses encryption algorithms to scramble data in transit, preventing hackers from reading it as it is sent over the connection. This information could be

anything sensitive or personal which can include credit card numbers and other financial information, names and addresses.

TLS

- TLS (Transport Layer Security) is just an updated, more secure, version of SSL. We still refer to our security certificates as SSL because it is a more commonly used term, but when you are buying SSL from DigiCert you are actually buying the most up to date TLS certificates with the option of ECC, RSA or DSA encryption.
- HTTPS (Hyper Text Transfer Protocol Secure) appears in the URL when a website is secured by an SSL certificate. The details of the certificate, including the issuing authority and the corporate name of the website owner, can be viewed by clicking on the lock symbol on the browser bar.

SSL Certificates

- SSL certificates have a key pair: a public and a private key. These keys work together to establish an encrypted connection. The certificate also contains what is called the “subject,” which is the identity of the certificate/website owner.
- To get a certificate, you must create a Certificate Signing Request (CSR) on your server. This process creates a private key and public key on your server. The CSR data file that you send to the SSL Certificate issuer (called a Certificate Authority or CA) contains the public key. The CA uses the CSR data file to create a data structure to match your private key without compromising the key itself. The CA never sees the private key.
- Once you receive the SSL certificate, you install it on your server. You also install an intermediate certificate that establishes the credibility of your SSL certificate by tying it to your CA’s root certificate. The instructions for installing and testing your certificate will be different depending on your server.

Purchasing SSL Certificates

- SSL certificates may be purchased from different providers.
- These certificates are valid for 1 year.
- For example, SSL.com sells certificates. They provide four types of certificates for different purposes and different prices
    a. Basic SSL
    b. Wildcard SSL
    c. SAN/UCC SSL
    d. EV SSL
- Another example is Aplus.net. It provides certificates for different purposes and different prices.

Self Signed SSL Certificates

- A self-signed certificate is a digital certificate that’s not signed by a publicly trusted Certificate Authority (CA).
- Self-signed certificates are considered different from traditional CA certificates that are signed and issued by a CA, because self-signed certificates are created, issued, and signed by the company or developer who is responsible for the website or software associated with the certificate.
- At a high-level, these self-signed certificates are based on the same cryptographic key pair architecture used in X.509 certificates. However, these digital certificates do not have the validation of a trusted third-party. This lack of independent validation in the issuance process creates additional risk, which is why self-signed certificates are considered unsafe for public-facing websites and applications.
- Self-signed certificates are excellent for development purposes and local testing.

Self Signed SSL Certificate Usage

- To use SSL certificates in python we will need to install openssl.

```
pip3 install pyopenssl
```

- pyOpenSSL is a Python module that supports TLS connections.
- The Python/Flask application on the next page represents a secure endpoint. Note how SSL is configured and not programmed into the application.

Creating a Secure Application

```
from flask import Flask, jsonify

app = Flask(__name__)

@app.route("/")
def main_page():
     return jsonify({"secured": "Hello world"})

if __name__ == "__main__":
     app.run(debug=True, ssl_context='adhoc')
```

Generating a Self Signed SSL

- The application on the previous page has security enabled but it is missing the SSL certificate.
- The missing SSL certificate will have to be created using the openssl command.

```
openssl req -x509 -newkey rsa:4096 -nodes -out cert.pem keyout
priv_key.pem -days 3650
```

- The above command creates a self signed certificate that is good for 3650 days or 10 years. The private key is in file named “priv_key.pem” and public key is “cert.pem”.
- To use the above keys you need to modify app.run.

```
app.run(debug=True, ssl_context=('cert.pem', 'priv_key.pem’))
```

- All endpoints will no longer be accessed though http but https.

Let’s Encrypt

- Let’s Encrypt is a nonprofit certificate authority providing TLS certificates.
- To create a fee TLS certificate you can follow the instructions on the Let’s Encrypt Getting Started page.
- You should understand that the free TLS certificates are valid for 90 days. After 90 days the TLS certificates expire and have to be renewed.

Heroku SSL

- Heroku provides a PaaS for hosting your applications. Heroku provides SSL as an option.
- SSL can be configured on Heroku PaaS using the steps on the Heroku SSL page.

CSRF Protection

- CSRF (Cross-Site Request Forgery) is an attack that uses the victim’s credentials to perform undesired actions on behalf of the victim. This vulnerability can become more severe if chained with XSS or Mis-Configured CORS.

- By default, the flask framework has no CSRF protection, but we can use Flask-WTF extension to enable the CSRF protection.

```
from flask_wtf.csrf import CSRFProtect
csrf = CSRFProtect(app)
```

- We have to keep some bullet points in mind while making the routes to keep our app protected from CSRF attacks.
    - GET requests should be used to retrieve data from the web.
    - Sensitive routes that change information should be performed with POST requests in the proper form submission.
    - Requests that change the state should be mandated with a CSRF token generated by the server and sent to the user's browser.
    - The Origin and Referrer header must be validated.

Cookies Protection

- Applications that run on the browser have most common attack vectors of cookies. Let's discuss all the options in Flask and Flask extensions such as Flask-Login and Flask-WTF to protect against cookie attack vectors.
- We should always use HTTPS rather than HTTP. Nowadays it's not a big issue to use SSL. We have to add these lines to protect against cookie attack vectors in our Flask configuration.
- See example code below:

```
app.config.update(
     SESSION_COOKIE_SECURE=True,
     SESSION_COOKIE_HTTPONLY=True,
     SESSION_COOKIE_SAMESITE='Lax',
)

# We can set secure cookies in response
response.set_cookie('key', 'value', secure=True, httponly=True,
samesite='Lax')
```

HTTP Header Protection

- We have to use proper HTTP headers to protect our apps. There are some HTTP headers which can be used to implement some sort of security.
- You can set these according to your app but we will mention those headers so you know which HTTP headers to use in case you develop an app. Below is the configuration to set an HTTP header and its value in the response.
- See example code below:

```
@app.after_request
def apply_caching(response):
     response.headers["X-Frame-Options"] = "SAMEORIGIN"
     response.headers["HTTP-HEADER"] = "VALUE"
     return response
```

HTTP Header Protection Options

- Content-Security-Policy (CSP)
- Strict-Transport-Security (HSTS)
- X-Permitted-Cross-Domain-Policies
- X-Frame-Options
- X-XSS-Protection
- X-Content-Type-Options
- X-Download-Options
- Public-Key-Pins (HPKP)

Video

Please watch the following video, Implementing Application Security. This video demonstrates how to enable user security for your web application.

Implementing Application Security
TMU Video

Summary

Application security is configured and not programmed within an application. There are multiple components to creating a secure application. The first component is to implement user security. This allows users to access routes only if they are logged in using a valid user account. The second component is to implement encryption using HTTPS access for your routes. The third component is to implement CSRF, cookies and HTTP header protection.

Assessments

Each week, you'll find here reminders of assignments or labs that you should be working on. For Week 11:

- Lab 11: Securing a web application using user credentials (5% of the final grade) is due in 7 days, i.e., the end of day Friday of Week 11.

Reminder

You have a weekly Zoom session. The date/timing for your Zoom session can be found in the Course Schedule in the Course Outline. The link to the Zoom session will be provided separately to you by your instructor. The Zoom session will consist of 20–30 minutes of lecture, followed by questions and answers (Q&A), followed by a lab period. You may ask questions about the previous or current week’s content or labs.

Please bring any questions to the weekly Zoom session. However, if you have any questions during the week outside of the Zoom session, you may post them in the Discussion Board. Your instructor may respond in the Discussion board or during the weekly Zoom session.

Click-n-reveal: **Where is the Discussion board?**

Click on the “Communication” area at the top main course menu, and select Discussions from the dropdown menu.

References

None for this week.