Chapter 8: Creating a Vulnerable Web Application

In this laboratory exercise, the student will be introduced to the concepts surrounding web applications and will create a vulnerable web application.

8.1 Laboratory Exercise

Web Application Creation

8.1.1 Specifications

This lab will be completed on an Ubuntu Server LTS 16.04.6 machine running Nginx version 1.14.0.

8.1.2 Learning Objectives

- Understand basics of web servers
- Configure a secure web server
- Create a web application

8.1.3 Mapping to NIST Nice Framework

This laboratory exercise is intended to increase the student's familiarity with webpages, web serving, and the interaction between PHP, MySQL, and Nginx. The student will need this information to successfully complete the web application hardening exercise. This laboratory exercise maps to the following KSAs from the NIST NICE Framework:

- Cybersecurity and Privacy Principles (K0004)
- Cyber Threats and Vulnerabilities (K0005)
- Application Vulnerabilities (K0006)
- Data Administration (K0020)
- Database Management Systems (K0023)
- Programming Language Structures and Logic (K0068)
- Query Languages (K0069)
- Application Security Threats and Vulnerabilities (K0070)
- Database Access Applicaiton Programming (K0197)
- Database Theory (K0420)
- Conducting Queries (S0013)
- Generate Queries (S0037)
- Writing Code (S0060)
- Develop Secure Software (A0047)
- Maintain Databases (A0176)

8.1.4 Necessary Background and Expected Completion Time

This laboratory exercise can be completed by students with varying background and experience. The following categories should help identify approximately how much time (in minutes) will be necessary to complete the laboratory exercise, for a student meeting the criteria for the respective experience level.

- Beginner: A student in this category has little to no experience with Linux, the Linux terminal, or Nginx.

- Intermediate: A student in this category has experience with Linux and the Linux terminal but has little to no experience with Nginx or web serving concepts.
- Advanced: A student in this category has experience with Linux and the Linux terminal. Additionally, this student has experience with web service concepts.

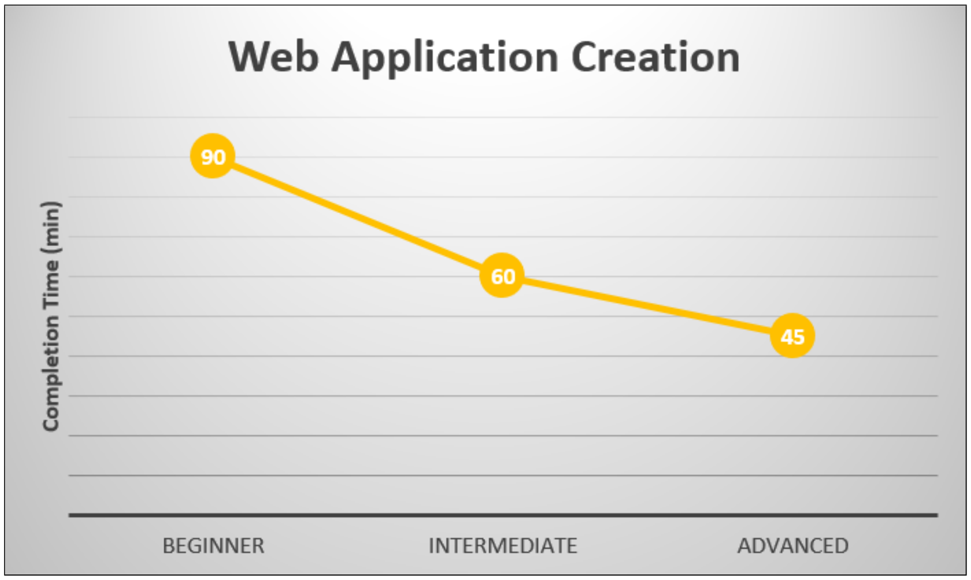

Figure 8.1: *Web Application Creation Laboratory Exercise Expected Completion Time (min)*

8.1.5 Configuration and Setup

The machine used in this laboratory exercise is an installation of Ubuntu Server LTS 18.04.1. Additionally, Nginx version 1.14.0 is installed on the machine to serve the webpage. The initialization script also installs Nginx, PHP, and MySQL. The machine is configured using the initialization script, listing **8.1**.

Listing 8.1: *vw-initializationscript.sh*

```bash
#!/bin/bash

sudo dpkg --configure -a


#install nginx
sudo apt-get install nginx -y

#install php
sudo apt-get install php-fpm php-mysql -y

#install MySQL
sudo apt-get install mysql-server -y
```

8.1.6 Challenges

1. *Stop Serving Page*

   Often times you will need to perform some sort of maintenance on your web server. Typically, this means you will need to stop serving your webpage for a brief moment.

   For this task, stop and then restart the web server. Verify that stopping the web server by trying to visit the site while the server is stopped.

2. *Identify Key File Locations*

   Websites are made up of files on a web server. The most common file is the `index.html` file which is typically the landing page when a user visits your website. There are a few other locations where critical files are kept. Identifying these locations will allow you to successfully configure your web server.

   For this task, identify the directory location for the following elements of your web server:

   - Web Page Content - that is the actual web pages which are being served and all the files associated with them.
   - Web Server Configuration - that is where all the configuration files for your web server are located.

3. *Enable HTTPS*

   Web servers often have users input usernames and passwords and other sensitive data. For this reason, it is very important to ensure that the data being sent to and from a web server are secure. In order to secure this data in transit, one can use HTTPS. In order to properly configure HTTPS, there are a few different things which must be accomplished. First, the user must generate SSL keys and a certificate. Once they have done this, the web server must be configured to use those keys and certificate. In a real enterprise environment, the certificate would be validated by what is known as a Certificate Authority (CA) but for now, it can be left as a self-signed certificate.

For this task perform the following:

- Use OpenSSL to generate a certificate and key.
- Configure Nginx to listen on port 443.
- Configure Nginx to use the certificate and key.

This can be a challenging task to accomplish as the necessary syntax is very specific. You should use the internet to complete this task and see the solutions and/or the guided walkthrough if needed.

4. *Disable Unnecessary HTTP Methods*

   HTTP is a protocol which defines how to format messages to navigate the internet. There are many HTTP methods, but only a few are commonly used. The three most commonly used methods are GET, POST, and HEAD. Other than these methods, there are multiple others; specifically, two methods DELETE and TRACE leave web servers vulnerable. The DELETE method can be used to delete a specified resource from a web server similar to the way that the GET method retrieves a resource. The TRACE method can be used to identify what resources are being transmitted to the other end of a request chain. Typically, this information would be used for debugging purposes but provides an attacker the opportunity to trace where traffic is flowing.

   For this task, disable all HTTP methods other than the GET, HEAD, or POST methods. Once again, for this task, the internet is a good resource. Refer to the solution manual if needed.

5. *Prevent Giving Away Server Content Details*

   HTTP error codes are a common way of conveying what went wrong when an error occurs. Common error codes that most users may recognize are:

   - 401, unauthorized access
   - 404, resource not found
   - 403, permission denied
   - 405, method not allowed

   These error codes, while convenient, can convey a lot of detail about your directory structure and your data to an attacker. Unless it is absolutely critical that you let the user know what type of error occurred, it likely suffices to just let them know that an error occurred. For example, if the attacker receives a 403 error code, they will know that content exists in that location and that the content is likely of some value since it is being protected. **For this task, route error 401, 403, 404, and 405 error codes to the same page that a 404 error routes to by default. Once again, make use of the internet and reference the solution manual as necessary.**

6. *Creating a Web Application*

   For this challenge, follow along with the code provided from the GitHub repository to create the files for the vulnerable web application. Try to identify which lines in the code correspond to the tasks in this challenge. Running the initialization script for the web application hardening laboratory exercise, listing 9.1, will perform the tasks in this challenge. If you use the initialization script, listing 9.1 to configure the vulnerable web application, discuss with peers or colleagues how you would have approached each task if starting from scratch.

   - Ensure that the machine has Nginx, MySQL, and PHP installed.
   - Edit configuration files so the landing page is index.php.
   - Edit configuration files so that Nginx interfaces with PHP, allowing the server to serve PHP pages.

- Create a MySQL database with the following data:
  - Database named **test**.
  - Table named **employees** with fields **name** and **password**, both of type VARCHAR.
  - Table named **startdates** with fields **name** and **date**, both of type VARCHAR.
  - Input the following (name,password) pairs into the employees table:
    (liam,password)
    (emma,superman)
    (william,mustang)
    (sophia,trustno1)
    (mason,hockeymason)
    (mia,curiousgeorge).
- Create an **index.html** with the following specifications:
  - Accept a username input
  - Accept a password input
  - Include a submit button
  - Send the inputs to **auth.php**
- Create an **auth.php** with the following specifications:
  - Interact with MySQL database using *root* user.
  - Check if user credentials were correct.
  - Display a link to **insertdate.php?name=$n**, where $n is the name of all users matching the query.
  - If no matches, send to **invalid.html**.
- Create an **insertdate.php** with the following specifications:
  - Accept a username input
  - Accept a startdate input
  - Include a submit button

  - Send the inputs to **insert.php**

- Create an **insert.php** with the following specifications:
  - Interact with MySQL database using *root* user.
  - Insert an entry into the MySQL **startdates** table with the **name** and **startdate** fields set to the values received from the **insertdate.php** page.
  - Link back to **index.html**.
- Create an **invalid.html** which states invalid input and links back to the **index.html** page.