**Developing Course Content for CSUF Web Security Submitted by:**

**Austin Thornton**

**892072794**

****

**California State University, Fullerton**

**CPSC-597**

**Spring 2021**

**Advisor: Dr. Doina Bein**

**Reviewer: Dr. Mikhail Gofman**

**Department of Computer Science**

**California State University-Fullerton**

****

**Department of Computer Science**

This project has been satisfactorily demonstrated and is of suitable form.

This project report is acceptable in partial completion of the requirements for the Master of Science degree in Computer Science.

|  |  |  |
| --- | --- | --- |
| Developing Course Content for CSUF Web Security | | |
| Project Title (type) | | |
| Austin Thornton |
| Student Name (type) |
| Dr. Doina Bein |
| Advisor's Name (type) |
|  | |
| Advisor's signature | Date |
| Dr. Mikhail Gofman |
| Reviewer's name |
|  | |
| Reviewer's signature | Date |

Table of Contents

[1 Abstract 5](#_Toc72511231)

[2 Motivation 6](#_Toc72511232)

[3 Related work 6](#_Toc72511233)

[4 Overall Architecture 7](#_Toc72511234)

[4.1.1 Baseline Banking Application 7](#_Toc72511235)

[4.1.2 Baseline Shopping Application 9](#_Toc72511236)

[4.2 Customizations to Expose Vulnerabilities and Plans of Attack 10](#_Toc72511237)

[4.2.1 Security Misconfiguration 10](#_Toc72511238)

[4.2.2 Cross Site Scripting 11](#_Toc72511239)

[4.2.3 Insecure Deserialization 11](#_Toc72511240)

[4.2.4 Using Components with Known Vulnerabilities 12](#_Toc72511241)

[4.3 Customizations to mitigate Vulnerabilities when possible 12](#_Toc72511242)

[4.3.1 Security Misconfiguration 12](#_Toc72511243)

[4.3.2 Cross-site scripting 13](#_Toc72511244)

[4.3.3 Insecure deserialization 13](#_Toc72511245)

[4.3.4 Using Components with Known Vulnerabilities 13](#_Toc72511246)

[4.4 Classroom exercises and walkthroughs 13](#_Toc72511247)

[5 Technical Design and Implementation Details 14](#_Toc72511248)

[5.1 Baseline Applications 14](#_Toc72511249)

[5.1.1 Bank Application 14](#_Toc72511250)

[5.1.2 Shop Application 20](#_Toc72511251)

[5.2 Changes to Design and Implementation 26](#_Toc72511252)

[5.2.1 Security Misconfiguration 26](#_Toc72511253)

[5.2.2 Cross-site Scripting 27](#_Toc72511254)

[5.2.3 Insecure Deserialization 27](#_Toc72511255)

[5.2.4 Using Components with Known Vulnerabilities 27](#_Toc72511256)

[5.3 Software Dependencies for OWASP exercises 28](#_Toc72511257)

[6 Tables 29](#_Toc72511258)

[7 Project Outcomes 32](#_Toc72511259)

[8 Future Work 32](#_Toc72511260)

[9 References 33](#_Toc72511261)

[10 Exercises 34](#_Toc72511262)

[10.1 OWASP 06 – Security Misconfiguration 34](#_Toc72511263)

[10.1.1 Walkthrough – Bank App 34](#_Toc72511264)

[10.1.2 Exercises – Bank App 34](#_Toc72511265)

[10.2 OWASP 06 – Security Misconfiguration 35](#_Toc72511266)

[10.2.1 Walkthrough – Shopping App 35](#_Toc72511267)

[10.2.2 Exercises – Shopping App 35](#_Toc72511268)

[10.3 OWASP 07 – Cross-site scripting 36](#_Toc72511269)

[10.3.1 Walkthrough – Bank App 36](#_Toc72511270)

[10.3.2 Exercises – Bank App 36](#_Toc72511271)

[10.4 OWASP 07 – Cross-site scripting 37](#_Toc72511272)

[10.4.1 Walkthrough – Shopping App 37](#_Toc72511273)

[10.4.2 Exercises – Shopping App 37](#_Toc72511274)

[11 OWASP 08 – Insecure Deserialization 38](#_Toc72511275)

[11.1.1 Walkthrough – Banking App 38](#_Toc72511276)

[11.1.2 Exercises – Banking App 38](#_Toc72511277)

[11.2 OWASP 09 – Using Components with Known Vulnerabilities 39](#_Toc72511278)

[11.2.1 Walkthrough – Shopping App 39](#_Toc72511279)

[11.2.2 Exercises – Shopping App 41](#_Toc72511280)

[Figure 1 - Database diagram of baseline bank application 18](#_Toc58691007)

[Figure 2 - Sequence diagram of bank transaction 19](#_Toc58691008)

[Figure 3 - Database diagram of baseline shop application 23](#_Toc58691009)

[Figure 4 - Sequence diagram of login process in shop application 24](#_Toc58691010)

[Figure 5- Sequence diagram for view profile / receipt history generation in shop app 25](#_Toc58691011)

# Abstract

The requirement for Cybersecurity continues to increase as the world adds greater integration of computer technology with businesses, society, and cities. “Cybersecurity is the array of measures you take to protect yourself from unauthorized access to your information systems.” (V)

Cybercrime statistics are vastly larger than the average person might be aware of, in 2017, the global cost of cybercrime added up to 600 million dollars. (Spencer) This amount is only expected to increase in the coming years. In addition, companies affected by a cyber-attack are costed about five million dollars on average. (Spencer) To throw salt in the wound, if the attack resulted in a mass data breach of customer records the company may also lose customers and credibility. This leads to the potential for businesses to be shut down due to the heavy price and poor recovery from lack of customer faith.

Despite the statistics being high and the expectation that it will increase does not mean that Cyber Security Professionals are not doing anything to defend against the attacks. While it is important to keep each computers antivirus software up to date, there are many other steps that can be taken to protect against attacks.

The Open Web Application Security Project, abbreviated to OWASP, is a nonprofit foundation that works to improve the security of software. (OWASP) The foundation has created a list based on a broad consensus of the most critical security risks for web applications. This list is called the OWASP Top Ten. While all security risks are important, it is a great starting point for new cyber security students to at least be aware of the OWASP Top 10. Though not all students taking cyber security classes will stick with it as a profession, knowing about them and avoiding them when possible are important coding practices.

The goal of this project is to create a method for students to experiment on the OWASP top 10 web vulnerabilities. The environment will be a set of exercises to allow the student to visually see the vulnerability and learn how to both exploit and fix them. The closest current free tool is the website XSS game. This project is a continuation of former graduate student, Chirstopher Katnic’s graduate project which covered OWASP vulnerabilities 1-5. This Iteration will cover exercises for OWASP vulnerabilities 6-9.

<https://github.com/Nitsua22/Graduate-Project>

# Motivation

As computer technology continues to advance, cyber security must keep changing to keep up. The best way to ensure cyber security continues to advance is to create new tools that will increase the speed and efficiency of training new cyber security learners. However, then it must be considered, “What is the most effective method of training new students?” There are five different methods of learning which can vary from person to person in effectiveness:

* Aural or sound like learning through music
* Verbal such as listening to a lecturer
* Logical which is learning through attributing reasoning to what the student sees
* Visual, learning through pictures and images
* Lastly, learning through physical touch or performing actions yourself

Through those methods, most of them are handled through lectures in a class except for physical learning. There are not many current methods for students to learn web security practices by actually trying out input commands to web applications without it costing a large sum of money to get access to online classes that give access to testing modules. Similar to how GNU was created to give users a free outlet for studying source coding. There should also be a free method for students to study and practice web security.

# Related work

1. OWASP research
   1. Research the topics for each exercise
   2. Create potential questions for each exercise topic
   3. Evaluate the questions on both what is necessary and possible
2. Prototyping
   1. Coding each exercise question
3. Testing
   1. Test all exercises in environment
4. Reevaluation
   1. Evaluating the questions after attempts at coding
   2. If necessary, modify exercises or design entirely different exercise
5. Documentation
   1. Create instructions for accessing and running each exercise as needed

# Overall Architecture

The applications developed for the project are twofold. The first is an application that simulates the features of a bank, and the second is an application that simulates a shopping website. The exercises created for the project will be designed using the two types of applications.

### Baseline Banking Application

1. Registration of New User Account
   1. Client shall send server the username, password, and other personal details
   2. Server shall parse the data, store it in the database, and authorize the user’s session to login
   3. If the desired username already exists, the server shall return an error, prompting the client to try with a new username
2. Login
   1. Client shall send server the username and password
   2. The server shall parse the data, authenticate with the database, and return a valid session id upon success
   3. If the information is not present in the database, the server will reply with an error to display on the client side
   4. The session shall be used to authenticate the user throughout the rest of the pages in the application, and the application shall require login if the session has expired
3. Create New Bank Account
   1. Client shall send server the login session, and the request to create a new account
   2. The server shall parse the data, create a new money account for the requesting user, set it to $0, and return the new list of accounts to the client to display
4. Deposit Money
   1. Client shall send server the login session, the request to deposit into the account, and the amount to add to the account
   2. The server shall parse the data, add money into the selected account, and return the updated list of accounts to the client to display
5. Withdraw Money
   1. Client shall send server the login session, the request to withdraw from the account, and the amount
   2. The server shall parse the data, subtract money from the selected account, and return the updated list of accounts to the client to display
   3. If the amount is greater than the account total, the server shall instead send back an error to the client to display
6. Transfer Money
   1. Client shall send server the login session, the request to transfer money between accounts, the amount, the sending and receiving account details
   2. The server shall parse the data, subtract money from the sending account, and return the updated list of accounts to the client to display
   3. If the amount is greater than the account total, the server shall instead send back an error to the client to display

### Baseline Shopping Application

1. Registration of New User Account
   1. Client shall send server the username, password, and other personal details
   2. Server shall parse the data, store it in the database, and authorize the user’s session to login
   3. If the desired username already exists, the server shall return an error, prompting the client to try with a new username
2. Login
   1. Client shall send server the username and password
   2. The server shall parse the data, authenticate with the database, and return a valid session id upon success
   3. If the information is not present in the database, the server will reply with an error to display on the client side
   4. The session shall be used to authenticate the user throughout the rest of the pages in the application, and the application shall require login if the session has expired
   5. If the user had a non-empty cart before logging out or the session expiring, the application shall restore the cart upon login
3. Session-Based Item Cart
   1. The server shall send the client a list of items available for purchase, and the client shall display that list
   2. Client shall send server the id of the item that it wishes to add to or remove from the cart
   3. The server shall parse the data, update the user’s cart, and return the updated cart data to the front end
   4. If the user does not have a cart, the server initializes a new one
4. Checkout
   1. The client captures payment and shipping information from the user
   2. The server creates a receipt for payment, including the captured information, the cart details, and other metadata
   3. The server replies with a success message once the data has been saved

## Customizations to Expose Vulnerabilities and Plans of Attack

For each vulnerability, the base application will be modified to expose the vulnerabilities to attack from the user. Additionally, sample plans of attack will be created to instruct the student on the fundamentals of the vulnerability and demonstrate the attack on both applications.

In addition, there will be customizations made to show how to reduce or remove the vulnerabilities.

### Security Misconfiguration

#### Banking App

The bank app was modified in the following ways to expose the vulnerability:

1. The feature to modify the current password of accounts shall be created
   1. When a user attempts to change the password, the application will not confirm the username or password before modifying the result

#### Shopping App

The Shopping app was modified in the following ways to expose the vulnerability:

1. The user accounts shall now include a ‘picture’ attribute
2. A new feature shall be added that allows the user to upload profile pictures
3. A new feature shall be added that allows the user to view the default profile pictures and other uploaded profile pictures
   1. When a user accesses this feature, they will have the ability to move backwards through the applications directory

### Cross Site Scripting

#### Banking App

The bank app was modified in the following ways to expose the vulnerability:

1. The default input sanitization enabled by EJS shall be disabled

#### Shopping App

The Shopping app was modified in the following ways to expose the vulnerability:

1. The default input sanitization enabled by EJS shall be disabled

### Insecure Deserialization

#### Banking App

The bank app was modified in the following ways to expose the vulnerability:

1. The cookie shall be modified to include the value that determines if an account is an administrator or not

The bank app is planned to be modified in the following ways to expose the vulnerability:

1. The setting for the cookie shall be modified to simplify decoding the cookie’s information

#### Shopping App

The Shopping app is planned to be modified in the following ways to expose the vulnerability:

1. The setting for the cookie shall be modified to simplify decoding the cookie’s information
2. The code shall be modified to allow for a reverse shell attack with the method of attacking being through modifying the cookie

### Using Components with Known Vulnerabilities

#### Banking App

An exercise is not yet created for the banking app

#### Shopping App

The Shopping app was modified in the following ways to expose the vulnerability:

1. The express-fileupload module shall be installed at version 1.1.7 to ensure it has the vulnerability.
   1. A copy of the processNested function code shall be added to the exercise folder if there are complications in downloading the older version along with a .txt file explaining it’s destination
2. The application shall be modified the parseNested function of the upload module is enabled
3. Copies of the example POST requests shall be added to the exercise folder

## Customizations to mitigate Vulnerabilities when possible

For each vulnerability, there will be a protected version of the code in addition to other strategies for mitigating the vulnerabilities, however some of the exposures were required to take the already established code and force it be vulnerable so they will not all be in depth.

### Security Misconfiguration

#### Banking App

Ensuring that all security configurations are reviewed and updated.

#### Shopping App

Focusing on keeping the platform to as minimal degree as possible will reduce the changes of security misconfiguration. Avoid installing unnecessary or unused features in applications.

### Cross-site scripting

Avoid allowing untrusted data from being processed through the program. When this is unavoidable, use automatic frameworks that automatically escape XSS when possible. However, not all frameworks are foolproof, learn the limitations when possible and try to compensate.

### Insecure deserialization

Limit what is received as a serialized object as much as possible. Not accepting serialized objects is the only true safe method but this is not always avoidable.  
Implementing integrity checks on any serialized objects will increase the chances of catching an attack before it slips in.

### Using Components with Known Vulnerabilities

Remove unused dependencies, unnecessary features, components, files, and documentation. (OWASP) Check with sources such as Common Vulnerabilities and Exposures(CVE) and National Vulnerability Database(NVD) for components before integrating them to make sure they do not bring and vulnerabilities into the application.

## Classroom exercises and walkthroughs

Classroom exercises for each unit and web application which detail the exploits used to penetrate each application are at the end of this document.

# Technical Design and Implementation Details

## Baseline Applications

### Bank Application

#### Features

|  |  |  |
| --- | --- | --- |
| MID | requireLogin | Middleware to require a valid user credentials before completing the request   * + - 1. If the current request contains user information          1. Forward the request to the next step       2. If the current request does not contain user information          1. Redirect the request to ‘/login’ |
| GET | ‘/login’ | Get the username and password from the form  Query database for usernames/passwords that match the submitted username  if the username from the form has a match in the db…  get the password from the db  hash the password provided from the form  if the passwords match…  render the home page with the account details provided  if the passwords do not match…  render the login page with an error message  if the username from the form does not have a match…  render the login page with an error message |
| GET | ‘/home’ | * + - 1. requireLogin       2. get all bank accounts from db for user          1. if no accounts were found, construct error message, pass to html constructor and render home page          2. if accounts were found, pass account details to html constructor and render home page |
| POST | ‘/register’ | * + - 1. Capture the form submitted data       2. Hash the password provided from the form       3. Query database to insert new user account record       4. If there is already an existing matching username in the database….          1. Render the registration form with an error message       5. If there is no existing matching username in the database…          1. Add a new bank account with balance of 0.00 and relate it to the newly created user record          2. Render the home page with the user details and new account details |
| GET | ‘/logout’ | * + - 1. Reset the current user’s session       2. Redirect the request to ‘/login’ |
| POST | ‘/createNew’ | * + - 1. requireLogin       2. Query database to insert a new bank account into the database for the logged in user       3. If a database error occurs (negative or bad value for initial deposit)…          1. Construct an error message and render the home page with it       4. If no database error occurs….          1. Redirect the response to ‘/home’ |
| POST | ‘/withdraw’ | * + - 1. requireLogin       2. Query database to get the account details of the account which is to be used for transaction       3. If database errors occur (bad account/user details)…          1. Construct an error message and render the home page with it       4. If no database error occurs…          1. Calculate the new balance of the account after subtracting the desired amount from the current balance          2. If the new account balance is less than zero….   Construct an error message and render the home page with it   * + - * 1. If the new account balance is greater than zero…   Query database to update the account balance of the account  Render the home page with the new account details |
| POST | ‘/deposit’ | * + - 1. requireLogin       2. Query database to get the account details of the account which is to be used for transaction       3. If database errors occur (bad account/user details)…          1. Construct an error message and render the home page with it       4. If no database error occurs…          1. Calculate the new balance of the account after adding the desired amount to the current balance          2. If the new account balance is less than the original account balance….   Construct an error message and render the home page with it   * + - * 1. If the new account balance is greater than the original account balance...   Query database to update the account balance of the account  Render the home page with the new account details |
| POST | ‘/transfer’ | * + - 1. requireLogin       2. Query database to get the account details of the two accounts which are to be used for transaction       3. If database errors occur (bad account/user details)…          1. Construct an error message and render the home page with it       4. If no database error occurs…          1. Calculate the new balance of the two accounts after adding the desired amount to the destination account and subtracting the desired amount from the source account          2. If the new account balance is less than the original account balance for the destination account OR If the account balance is less than zero for the source account…   Construct an error message and render the home page with it   * + - * 1. If the new account balance is greater than the original account balance for the destination account AND the account balance is greater than zero for the source account...   Query database to update the account balance of the accounts  Render the home page with the new account details |

|  |  |  |
| --- | --- | --- |
| MID | requiresAdmin | Middleware to validate user role before accessing next step   * + - 1. If the currently logged in user has a role == 0 (admin)          1. Forward the request to the next step       2. If the currently logged in user has a role > 0 (non-admin)          1. Construct an error message and render the ‘home’ page with it |

#### Database Design

The Following is a database diagram that depicts the entity relationship of each in the database of the bank baseline application.



Figure 1 - Database diagram of baseline bank application

#### Sequence Diagrams

The following sequence diagram depicts the sequence flow for transactional operations (withdraw, transfer, deposit)



Figure 2 - Sequence diagram of bank transaction

### Shop Application

#### Features

|  |  |  |
| --- | --- | --- |
| MID | requireLogin | Middleware to require a valid user credentials before completing the request   * + - 1. If the current request contains user information          1. Forward the request to the next step       2. If the current request does not contain user information          1. Redirect the request to ‘/login’ |
| GET | ‘/login’ | Get the username and password from the form  Query database for usernames/passwords that match the submitted username  if the username from the form has a match in the db…  get the password from the db  hash the password provided from the form  if the passwords match…  render the home page with the account details provided  if the passwords do not match…  render the login page with an error message  if the username from the form does not have a match…  render the login page with an error message |
| GET | ‘/home’ | * + - 1. requireLogin       2. get all products from db to display on home page          1. if no products were found, construct error message, pass to html constructor and render home page          2. if products were found, pass account details to html constructor and render home page |
| POST | ‘/register’ | * + - 1. Capture the form submitted data       2. Hash the password provided from the form       3. Query database to insert new user account record       4. If there is already an existing matching username in the database….          1. Render the registration form with an error message       5. If there is no existing matching username in the database…          1. Render the home page with the user details and new account details |
| GET | ‘/logout’ | * + - 1. Reset the current user’s session       2. Redirect the request to ‘/login’ |
| GET | ‘/additem | * + - 1. requireLogin       2. Query the database for the item       3. If there is no such item in the database…          1. Construct an error message and render the home page with it       4. If no database error occurs…          1. If there is no matching item in the user’s cart…   Add the item to the cart and set the quantity to 1   * + - * 1. If there is a matching item in the user’s cart…   Increase the quantity of that item in the cart by 1   * + - * 1. Update the user’s cart         2. Render the home page with the updated cart |
| GET | ‘/deleteitem’ | * + - 1. requireLogin       2. Query the database for the item       3. If there is no such item in the database…          1. Construct an error message and render the home page with it       4. If no database error occurs…          1. If there is no matching item in the user’s cart…   Construct an error message and render the home page with it   * + - * 1. If there is a matching item in the user’s cart…   Reduce the quantity of that item in the cart by 1   * + - * 1. Update the user’s cart         2. Render the home page with the updated cart |
| POST | ‘/checkout’ | * + - 1. requireLogin       2. If there are no items in the user’s cart…          1. Construct an error message and render the home page with it       3. If there are items in the user’s cart….          1. Capture the form field details from the front end          2. Calculate the grand total of the receipt          3. Query the database to insert a new receipt with the calculated values and cart items list          4. If there was a database error…   Construct an error message and render the home page with it   * + - * 1. If there was no database error…   Render the thankyoupurchase page with the receipt details |
| GET | ‘/viewprofile’ | * + - 1. requireLogin       2. Query the database for the account details of the logged in user       3. If there was a database error OR no results….          1. Construct an error message and render the view profile page with it       4. If there was a database record….          1. If there was no receipt request….   Render the view profile page with the account details   * + - * 1. If there was a receipt request….         2. Query the database for receipts which match the date range passed in from the form         3. If there was a database error OR no results….   Construct an error message and render the view profile page with it   * + - * 1. If there were one or more results…         2. Render the view profile page with the receipt details |
| GET | ‘/viewcart’ | * + - 1. requireLogin       2. If there is no cart OR there are no items in the cart…          1. Construct an error message and render the viewcart page with it       3. If there is a cart AND there are items in the cart…          1. Render the viewcart page with the cart details |

#### Database Design

The Following is a database diagram that depicts the entity relationship of each in the database of the shopping baseline application.



Figure 3 - Database diagram of baseline shop application

#### Sequence Diagrams

The following sequence diagram depicts the sequence flow for the checkout process.



Figure 4 - Sequence diagram of login process in shop application

The following sequence diagram depicts the sequence flow for the profile view / receipt lookup flow.



Figure 5 - Sequence diagram for view profile / receipt history generation in shop app

## Changes to Design and Implementation

### Security Misconfiguration

#### Banking Application

A new feature was added to change passwords in order to protect the admin account.

|  |  |  |
| --- | --- | --- |
| POST | ‘/password’ | * + - 1. requireLogin       2. Get the username and password from the form       3. Hash the password provided from the form       4. Query database to compare the username and password       5. If the username and current password do not match          1. Render the login page with an error message       6. If the username and current password both match the database          1. Change the password for the corresponding username to the new password in the database          2. Render the home page with the notification of password change complete |

#### Shopping Application

A new feature was added to give the users a profile picture and the ability to change the profile picture by uploading a new image. To accomplish this, the node module express-fileupload was added.

|  |  |  |
| --- | --- | --- |
| POST | ‘/upload’ | * + - 1. requireLogin       2. checkIdWithUser       3. Send the new image to the profile picture folder       4. Query database to insert new profile picture name for current user       5. Redirect user back to viewprofile          1. If the file is not an image, it will display image fail when returned to viewprofile |

### Cross-site Scripting

#### Banking Application

None necessary

#### Shopping Application

None necessary

### Insecure Deserialization

#### Banking Application

The node module client-sessions will need to be replaced with another type of cookie module that does not require the use of a secret for the cookie

#### Shopping Application

The node module client-sessions will need to be replaced with another type of cookie module that does not require the use of a secret for the cookie

### Using Components with Known Vulnerabilities

#### Banking Application

None

#### Shopping Application

The shopping application also uses the same file upload feature added in Security misconfiguration. This time the module for express-fileupload has the setting for parseNested set to enabled.

## Software Dependencies for OWASP exercises

The applications have been built with:

* JavaScript
* HTML
* CSS.

Node.JS is used as the JavaScript Library. The following packages are installed for all exercises:

* Npm - a software registry which provides a mechanism to download and maintain updates for the rest of the packages
* Nodemon - a utility that automatically stops and restarts the nodejs server when one of the files are changed
* Express - a framework that provides much of the core web application features that is implemented throughout the back-end of the application
* Bcrypt - a library that includes hashing functions to be used when saving passwords or other sensitive data
* Body-parser - a library that provides features to enable the back-end to consume data from the client
* Uuid- a library that provides features to create uuids, to be used in session management
* Client-sessions - a library that provides an encrypted client session utility
* Cookie parser - a library that provides client-side cookies to save temporary data
* Helmet-csp - a library with a suite of features that enable content security policies, which are used to thwart cross-site scripting attacks
* EJS - a front-end library that provides features allowing html templates to be populated dynamically with data received from the back end

MariaDB is used for managing the applications databases and requires MySQL server running on the server-side

# Tables

Table 1 - OWASP Top 10

|  |  |  |
| --- | --- | --- |
| # | Title | Description |
| 1 | Injection (SQL, NoSQL, OS, LDAP, etc.) | Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker’s hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization |
| 2 | Broken Authentication | Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users’ identities temporarily or permanently. |
| 3 | Sensitive Data Exposure | Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser. |
| 4 | XML External Entities (XEE) | Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks. |
| 5 | Broken Access Control | Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users’ accounts, view sensitive files, modify other users’ data, change access rights, etc. |
| 6 | Security Misconfiguration | Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched/upgraded in a timely fashion. |
| 7 | Cross-Site Scripting (XSS) | XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim’s browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites. |
| 8 | Insecure Deserialization | Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks. |
| 9 | Using Components with Known Vulnerabilities | Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts. |
| 10 | Insufficient Logging & Monitoring | Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring. |

# Project Outcomes

The outcome for this iteration of the project are the foundations of 6 additional exercises among the vulnerabilities 6-9 along with the along with the videos recorded to walk through the vulnerabilities. The information for creating and populating the databases of each exercise are also prepared but must be performed manually before the start of each exercise.

# Future Work

The Next iteration the project will include the completion of the exercises for OWASP vulnerabilities #8 and #9. In addition, the exercises will be created for OWASP vulnerability #10.

The last iteration of this project for full release will be the completion of all 20 web applications exercises along with the instructional content created and ready to be deployed using another student’s project which was to create the rapid deployment infrastructure that these web applications will eventually sit on.

Additionally, the web applications for this project can be reworked to have a more convincing look-and-feel, and additional exercises should be created to help students further explore how the vulnerabilities in this project may appear in similar web applications.

# References

Alzahrani, M. (2018). Auditing Albaha University Network Security using in-house Developed Penetration Tool. Journal of Physics., 978, 012093.

D. Xu, W. Xu, M. Kent, L. Thomas and L. Wang (2015), "An Automated Test Generation Technique for Software Quality Assurance," in IEEE Transactions on Reliability, vol. 64, no. 1, pp. 247-268.

LuHill, Felicity (2017) Overview of learning styles. Retrieved May 2, 2020. https://barbershopbooks.org/methods-learning/

NCSC, National Cyber-Security Centre (2017). Penetration Testing. Retrieved March 3, 2020, from https://www.ncsc.gov.uk/guidance/penetration-testing

OWASP, The open web security project (2017) OWASP Top 10 – 2017 Retrieved March 10, 2020. from https://www.owasp.org

Posix (2020, July 28) Real-world KS – 1. Retrieved May 17, 2020, from https://blog.p6.is/Real-World-JS-1/

Spencer, J. (2019). Why is Cyber Security Important in 2019? Retrieved March 10, 2020, from https://securityfirstcorp.com/why-is-cyber-security-important/

V, D. (2019, September 5). Importance Of Cybersecurity for Businesses. Retrieved March 10, 2020, from https://www.techtimes.com/articles/245233/20190904/importance-of-cybersecurity-for-businesses.html

Website is down and cannot obtain the information for the reference, used for information on the table for the explanations of each vulnerability:

https://www.creditfreeze.info/factors-driving-api-growth-in-industry/

# Exercises

## OWASP 06 – Security Misconfiguration

### Walkthrough – Bank App

1. Login with Admin account using username=admin password=admin
2. Admin account can access the config page
   1. Obvious security vulnerability the password remains easy to guess
3. Change the password of the admin account
4. Logout
5. Create new account and log into new account
6. Go to change password again
   1. Instead of inputting the new account username, input admin then a new password
7. Log out of the new account
8. Try logging in with the admin account using the new password

### Exercises – Bank App

1. Update the bank application to confirm the username if the original password is correct.
2. Optional: add the functionality to also change the username of the accounts
3. Optional: add additional security to usernames and passwords allowed
   1. Example: password must 1 uppercase letter and 1 lowercase letter

## OWASP 06 – Security Misconfiguration

### Walkthrough – Shopping App

1. Create a new account or login into any known account
2. Navigate to the View profile page
3. Choose select file and pick a .jpeg file
4. Press upload
5. From the View profile page select to view profile picture gallery
6. The gallery takes you to a directory listing all the profile images
7. If selected, the “..”, will take you further out in the directory
   1. The user is given full access to node packages and other code for the application
   2. From there an attacker can work to find other potential vulnerabilities

### Exercises – Shopping App

1. Modify the Shopping application to prevent the user from moving out of the profile picture directory
2. Optional: Modify the Shopping application to display the images instead of a directory listing

## OWASP 07 – Cross-site scripting

### Walkthrough – Bank App

1. The application does not sanitize user input to escape from XSS input
2. Create a new account and give it the First name of <script>alert(1)</script>
3. When you log in to this account and it displays reach a menu that displays the first name of the account it will trigger alert with the value of 1
   1. If you try and put the alert into the username, last name, or address section instead, it will not trigger any alerts unless we can get to the admin page



### Exercises – Bank App

The application uses EJS which has the ability when writing the code to use methods to automatically escape cross-site scripting attacks

1. Modify the Bank app to escape XSS input
2. Optional: create a function that takes the input values and filters out potential cross-site scripting characters

## OWASP 07 – Cross-site scripting

### Walkthrough – Shopping App

1. The application does not sanitize user input to escape from XSS input
2. Create a new account and give it the username of <body onload=alert("1")>;
3. When you log in to this account and it displays reach a menu that displays the username of the account it will trigger alert with the value of 1
4. From there figure out which inputs can trigger alerts and what cannot
   1. For this application its possible to trigger a XSS alert from the username, first name, last name, and address because they are shown on the view profile page

### Exercises – Shopping App

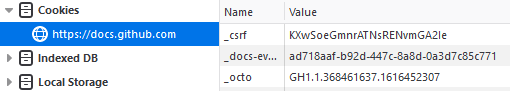
The application uses EJS which has the ability when writing the code to use methods to automatically escape cross-site scripting attacks

1. Modify the shopping app to escape XSS input
2. Optional: create a function that takes the input values and filters out potential cross-site scripting characters

# OWASP 08 – Insecure Deserialization

\*Note: Current version will not allow for simple cookie decoding. A large portion of development was spent on attempting to modifying the main code to allow for cookies that could be easily decoded. It was not until the near the end of development that it would be a better solution to rewrite the code and implement a new module for handling cookies that do not require a secret key. This will need to be implemented in the next iteration\*

### Walkthrough – Banking App

1. Create a new account or log into any account that is not the admin account.
2. Right-click the web browser and select inspect
3. Select the memory tab on the new panel
4. Expand the cookies option  
   
5. From there select the cookie for the exercise application
6. Run the cookie through a base64 decode to see the values stored
7. Modify the the value for the role\_id to be 0
8. Run the new value through the base64 encoder
9. Go back to the cookie menu and right-click the cookie and enter the new encoded cookie
10. Select the Config tab from the home menu

### Exercises – Banking App

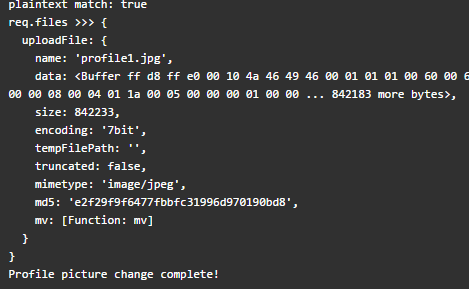
1. Modify the Shopping application to prevent the user from moving out of the profile picture directory to have a more difficult cookie to decode
2. Optional: implement a check that reconfirms the user’s cookie is unmodified

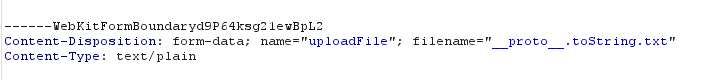
## OWASP 09 – Using Components with Known Vulnerabilities

### Walkthrough – Shopping App

The npm module that was added for OWASP 06 that allowed for file uploading, if the correct versions is installed, has the flaw when a setting is enabled called parseNested. When this is set to true, it is possible to cause a pollution of the string operator. If a POST request is sent with the name of the file being \_\_proto\_\_.toString, it can lead to the high jacking and pollution of operators. It does not have to be \_\_proto\_\_.toString specifically, this can be used with any operation that has \_\_proto\_\_ at the start.

Burp suite community edition or another software capable of sending POST requests will be required.

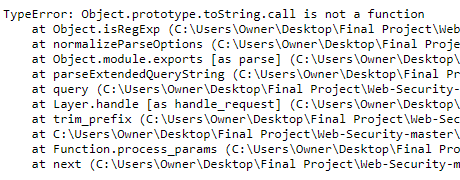
1. Launch Burp suite
   1. Use temporary project
   2. Use Burp defaults
2. Select the Proxy tab
   1. Choose Open Browser
   2. Leave intercept off for the moment
3. On Burp suite browser go to localhost:3000
4. Login to any profile or create a new one
5. Navigate to the view profile page
6. Upload any regular image
7. Observe the console for the details of the file
8. On Burp suite, turn intercept on
9. Select a file to upload
   1. For simplicity select the file \_\_proto\_\_.toString.txt to upload
10. After pressing the upload button, observe the POST request on Burp suite



* 1. At the Content-Disposition line, change the name to equal “\_\_proto\_\_.toString”



1. Then use the Forward button on Burp suite to send the POST request (it may need 2 presses)
2. Now the method, Object.prototype.toString has been polluted because a something that is not a function has been placed into it.
3. Every request sent to the application will now result in a 500 error



\*Note: The application uses EJS, because of this it is possible to create a reverse shell by polluting using \_\_proto\_\_.outputFunctionName. However, researching and implementing the current state of the vulnerability took far longer than anticipated due to this a consistent process for this was not developed before the end of this iteration and will need to be implemented in the next\*

### Exercises – Shopping App

1. Locate the express-fileUpload module and modify the processNested function to avoid processing anything starting with \_\_proto\_\_
2. Optional: Update the Shopping Application to only take in files for images such as .png or .jpeg