Vulnerable Web Application Framework

Nicholas J. Giannini
University of Rhode Island, ngiannini@my.uri.edu

VULNERABLE WEB APPLICATION FRAMEWORK

BY

NICHOLAS J. GIANNINI

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER SCIENCE AND STATISTICS

UNIVERSITY OF RHODE ISLAND

2015

MASTER OF SCIENCE THESIS

OF

NICHOLAS GIANNINI

APPROVED:

Thesis Committee:

Major Professor	Victor Fay-Wolfe
	Lisa DiPippo
	Haibo He

Nasser H. Zawia
DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND
2015

ABSTRACT

Utilizing intentionally vulnerable web applications to teach and practice cyber security principles and techniques provides a unique hands-on experience that is otherwise unobtainable without working in the real world. Creating such applications that emulate those of actual businesses and organizations without exposing actual businesses to inadvertent security risks can be a daunting task. To address these issues, this project has created *Porous*, an open source framework specifically for creating intentionally vulnerable web applications. The implementation of *Porous* offers a simplified approach to building realistic vulnerable web applications that may be tailored to the needs of specific cyber challenges or classroom exercises.

ACKNOWLEDGMENTS

None of the work in this thesis would have been possible without the constant support of my family, friends, and colleagues. I would like to especially thank Dr. Victor Fay-Wolfe for the opportunities he has provided me over the years as both an undergraduate and graduate student at the University of Rhode Island. He has not only prompted my interest in the fields of digital forensics and cyber security but through his guidance and encouragement has inspired me both academically and personally.

I would also like to thank the members of the Digital Forensics and Cyber Security Center and all those involved with the Department of Computer Science and Statistics. Namely, I would to thank Dr. Lisa DiPippo who has also exhibited guidance and support throughout the entirety of my collegiate career. In addition, I would like to thank Jacob Fonseca for never asking the easy the questions and for always challenging me to do better. His advice and extensive knowledge has been an invaluable resource over the years.

Most importantly, I would like to thank my parents who despite having to put up with me have always been there to steer me in the right direction or to provide the support I needed while pushing me to succeed.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In today's era of digital information, web applications, which commonly act as publicly facing entities for many businesses and organizations, are often the target of malicious attacks by hackers who wish to steal customer data or pivot their way deeper into an organization's internal network. It is essential that education and training for industry professionals has intentionally vulnerable web applications for realistic training in how to secure those applications.

1.1 Statement of the Problem

There are some web applications available that are designed to be deliberately vulnerable for training purposes. However, many of these vulnerable web applications are either outdated, not configurable, are of limited utility for realistic training, or consist of static content that limits them to a single time use in training. Furthermore, most training web applications are created from scratch – a very time-consuming and difficult task with little to no re-use of the development effort shared amongst developers. To solve this problem, this project implemented a web application framework called *Porous* that simplifies the process of developing configurable vulnerable web applications for training purposes by abstracting out the common structure and functionalities that are found in a typical web application. By using *Porous*, application developers will be able to better focus on developing the aspects of an individual web application that makes it unique rather than devoting effort to the basic structure of the vulnerable web application.

1.2 Justification for and Significance of the Study

According to the *Website Security Statistics Report* published by Whitehat Security [1], an organization that provides website risk management solutions, approximately 86% of all surveyed websites have at least one serious vulnerability, with most having far more. Of the vulnerabilities found, only 61% were resolved and required an average of 193 days to resolve from the date of first customer notification. In addition, only 57% of surveyed organizations said they provide some form of software security training to their programmers. These statistics illustrate the overwhelming amount of vulnerabilities that are present in today's web applications as well as the lack of qualified security professionals working within organizations.

There are two use cases for creating intentionally vulnerable web applications within the context of cyber security education. The first use case, tutorials, require stand-alone web applications that reinforce individual lessons or allow students to practice a particular technique. The second use case, for which the *Porous* framework was developed, is cyber challenges. A cyber challenge is an interactive learning environment where students are given hands-on experience practicing cyber security skills without the legal or moral implications that are often associated with using such techniques in the real world. The role of web applications within cyber challenges is that they simulate the web applications of realistic businesses and organizations. Since the individual needs of a business or organization vary drastically, it can be a difficult and time consuming task to build custom web applications for each cyber challenge scenario.

1.3 Goals

The goal of this project is to create a web application framework that may be used to develop intentionally vulnerable web applications. In order to accomplish this goal, the framework must:

1. Simplify the development of vulnerable web applications
2. Provide configurable security features
3. Be evocative of current web application security concerns
4. Be extensible

1.4 Summary of Accomplishments

The result of this project was the creation of the *Porous* web application framework that may be used to develop intentionally vulnerable web applications. The *Porous* framework met the goals specified above in Section 1.3 by implementing an extensible structure that allows for the configuration of security features within the components that are commonly found in web applications.

CHAPTER 2

REVIEW OF LITERATURE

This chapter serves to elaborate on material, both conceptual and technical, that has assisted in the development of the web application framework that corresponds to the goals of Section 1.3 by providing context, inspiration, and foundational information to this research. The chapter will begin by first defining what exactly a web application framework is and how they assist in the development of web applications. Next, it will discuss related works that have had either similar goals or are complementary to this project. Afterwards, there is a discussion of the technologies used in the development of the web application framework for this project. And lastly, the target audiences that this framework is intended for are defined.

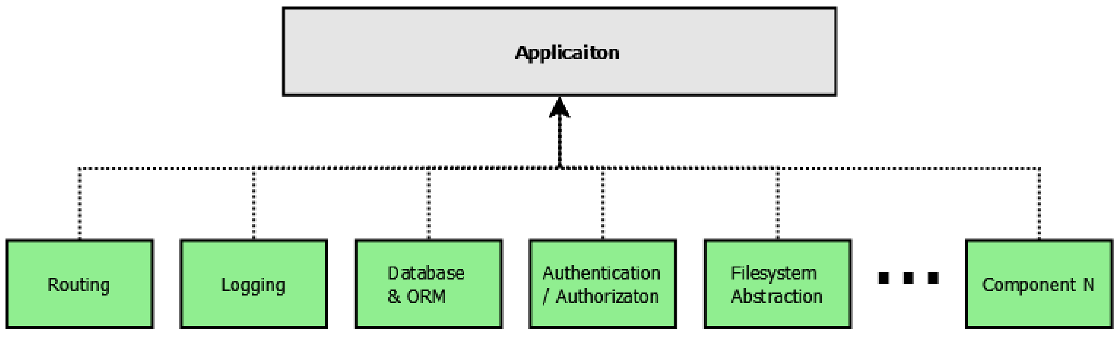
2.1 Web Application Frameworks

A web application framework is a specific type of software framework that is designed for easing the development of web applications and services. Software frameworks are able to ease the development process by providing developers with a structured abstraction that contains interfaces to functionalities common to the types of applications the framework was built to develop. In the case of a web application framework, the framework may provide interfaces to common web application functions such as routing, cookies, session management, and database management. The benefit of using such a framework is that code reusability is encouraged thereby facilitating the rapid development of web applications by allowing the application developer to focus on the business logic of the application instead of the common components. Web application frameworks and software frameworks in general, can be further broken down into *Full-Stack*, *Micro*, and *Monolithic* categories based on how influential they intend to be on the structure of the application and how much they intend to assist the developer. I now describe each of those framework categories.

2.1.1 Full-Stack Frameworks

A full-stack framework is a framework that attempts to provide nearly everything that a developer could possibly need to build an application. It likely includes components that may not be needed by the majority of applications, but by having them available makes it easier for new features to be integrated. Examples of full-stack web application frameworks include Symfony (PHP) [2], Laravel (PHP) [3], Ruby on Rails (Ruby) [4], and Django (Python) [5].

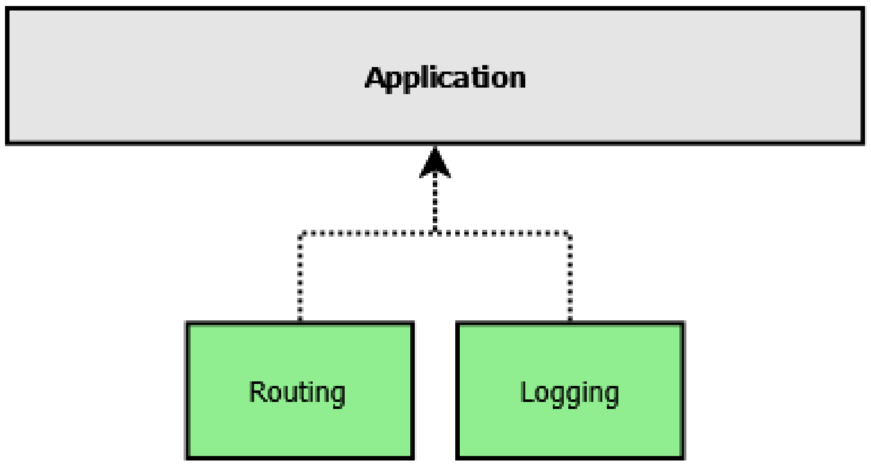
Figure 1. Sample Architecture of a Full-Stack Framework



2.1.2 Micro Frameworks

A micro framework is a framework that attempts to provide only the components that are absolutely necessary for a developer to build an application; or it may focus on providing the functionality of one particular area very efficiently. Micro frameworks tend to be better-suited for smaller applications or for applications that are within the very specific purpose for which the framework was designed. In the case of web application frameworks, a micro framework may be specifically designed for building the public application programming interfaces (APIs) for another service or application. Micro frameworks often need to be extended with additional components in order to make them provide the functionalities required for the web application being developed. Examples of micro web application frameworks includes Slim (PHP) [6], Silex (PHP) [7], Sinatra (Ruby) [8], and Flask (Python) [9].

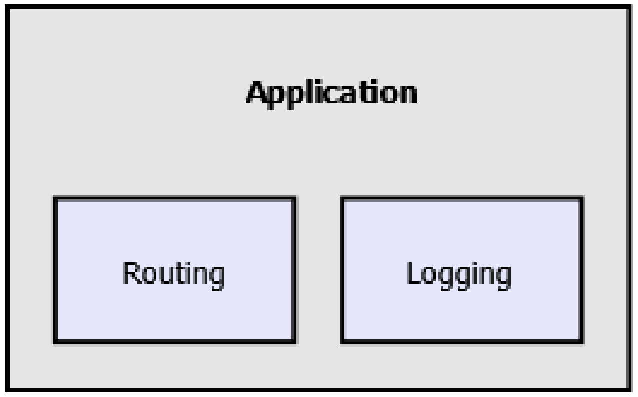
Figure 2. Sample Architecture of a Micro Framework



2.1.3 Monolithic Frameworks

A monolithic framework is a framework in which the components cannot be easily swapped out for different implementations or extended. Both full-stack and micro frameworks can be monolithic, however it is a more common trait in smaller micro frameworks. Slim [6], for example, was developed as a monolithic framework until more recently. The main objective of Slim was to remain lightweight and fast, which was accomplished by having a highly optimized code base. However, this came at the cost of having tightly coupled code that could not easily be extended or modified without affecting the rest of the framework.

Figure 3. Sample Architecture of a Monolithic Framework

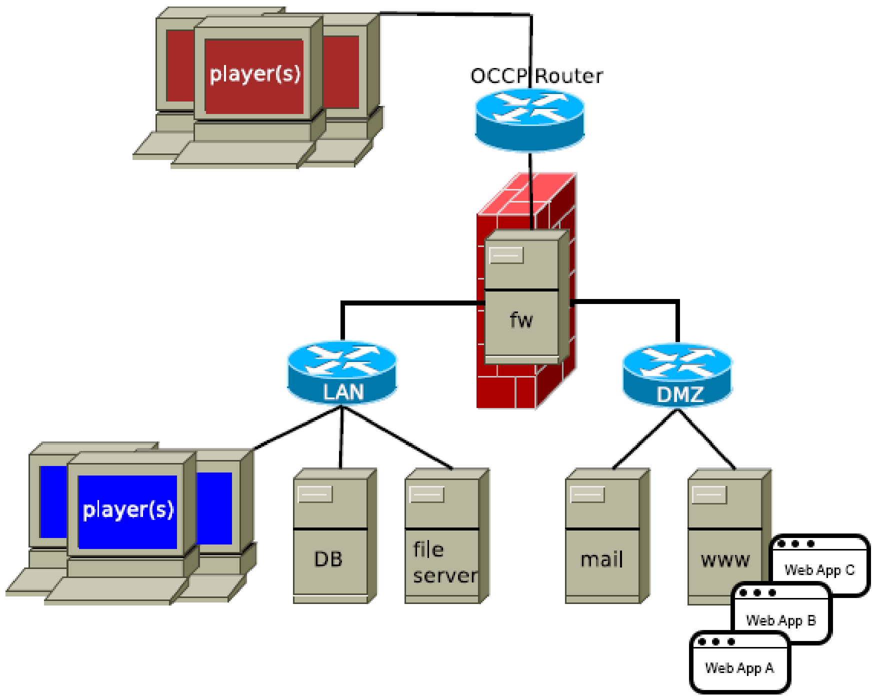


2.2 Related Works

2.2.1 Open Cyber Challenge Platform

As mentioned in Section 1.2, one of the motivating factors for developing a web application framework is developing intentionally vulnerable web applications for use in cyber challenges. Incidentally, creating cyber challenges themselves is a difficult and time-consuming task due to the high level of technical ability required to create and implement an individual scenario to be used as a challenge. To help address this problem, researchers at the University of Rhode Island are actively developing the Open Cyber Challenge Platform (OCCP) [10], which is an open-source platform for building cyber challenges that aims to be extensible, modular, and reusable. Considering the similar goals of both projects, the project being completed for this these can be seen as a complementary asset to the OCCP. As the OCCP aims to simplify the development of cyber challenges, the framework described in this thesis aims to simplify the development of web applications to be used within those challenges. The figure below displays a sample network that the OCCP may generate for a scenario. The web applications developed using the framework may be placed within a web server within such a network design.

Figure 4. OCCP Network Diagram



2.2.2 Open Web Application Security Project

The Open Web Application Security Project (OWASP) [11] is a community dedicated to the creation of tools, documentation, and technology relating to web application security. For this reason, several of OWASP's projects were reviewed during the completion of this thesis project with the three most significant: The *OWASP Site Generator, the OWASP Top 10,* and *the OWASP testing project*, described below.

2.2.2.1 OWASP Site Generator

The OWASP Site Generator [12] project was created and sponsored by Foundstone and SPI Dynamics during the OWASP Spring of Code in 2007. The project had intentions of creating a tool that could create dynamic websites using predefined vulnerabilities and web architectural elements based on an XML configuration file. The web applications generated by this tool were written in Microsoft's .NET languages. Later efforts sought to expand to include other language options. The project had similar goals of simplifying the development of web applications with configurable security features, however it fell short due to other design decisions. First, the sites that it generated were based on predefined templates, which limited the customization of the websites being developed. Second, the tool was created as a Windows desktop application and generated websites intended to run on Microsoft' IIS web servers. This introduced a dependency on the Windows operating system that could have severely limited the user base of the tool and limited the tool's utility for use in cyber challenges by specifying a platform for deployment.

As of 2008 the project status had changed to inactive, although later efforts to revive the project occurred and failed in 2009.

2.2.2.2 OWASP Top 10

OWASP categorizes its projects as being in different stages of maturity with the most mature projects reaching flagship status, which indicates that the project is not only extremely mature but also has the direct support of OWASP as an organization to continue to maintain and develop. Perhaps its most venerated flagship project is the OWASP Top 10 [13], which is a list of the top ten most statistically common web security concerns. For each of these security concerns, the OWASP provides example vulnerabilities, attacks, reference materials, and suggestions on how to avoid such weaknesses. I used this list as a reference for the types of vulnerabilities to include in *Porous*.

The most recent list of security flaws at the time of this writing comes from the 2013 OWASP Top 10, which includes:

A1. <u>Injection</u>

Any flaw that allows for untrusted data to be sent to an interpreter as part of a command or query, which allows for a malicious user to execute unintended commands or access data without authorization. Example attacks that exploit these flaws include SQL injection, operating system command injection, and LDAP injection.

A2. Broken Authentication and Session Management

Any flaw associated with the authentication of authorized users and the management of sessions. Example attacks that exploit these flaws include password attacks, session fixation, and session hijacking.

A3. Cross-Site Scripting (XSS)

Any flaw that allows for data to be sent to a client web browser without proper validation or escaping, which allows an attack to execute malicious scripts.

A4. Insecure Direct Object References

Any flaw that exposes a reference to the internal implementation of an artifact or asset without proper access controls. These flaws could allow an attacker to view and/or manipulate directories, files, keys, etc.

A5. Security Misconfiguration

Any flaw associated with having configuration settings in an application, database, server, or other entity that are insecure either by default or by a developer's decision.

A6. Sensitive Data Exposure

Any flaw that exposes sensitive data such as customer information, financial data, session identifiers, etc. by not properly handling information or not encrypting it during rest or while in transit.

A7. Missing Function Level Access Control

Any flaw that allows non-privileges users to have access to functions that should only be available to authorized users.

A8. Cross-Site Request Forgery (CSRF)

Any flaw that allows an attacker to trick authenticated users into performing unintended actions through the use of forged HTTP requests.

A9. Using Components with Known Vulnerabilities

Any flaw that results in the exploitation of an application caused by the use of vulnerable libraries, components, frameworks, or software.

A10. Unvalidated Redirects and Forwards

Any flaw that allows an attacker to take advantage of a redirect or forwarding feature in a web application to send a victim to a malicious site or to access unauthorized resources.

2.2.2.3 OWASP Testing Project

The OWASP Testing Project [14] is another flagship project that aims to provide guidelines for the testing of web applications. The project claims to have developed a complete testing framework that can be used as a template for testing applications or to qualify the testing processes or others. It includes testing methodologies for each stage of development, as well as recommended procedures for testing various parts of a web application. I used tests from the OWASP Testing Project to test the components of *Porous*.

2.2.3 Damn Vulnerable Web Application (DVWA)

The Damn Vulnerable Web Application (DVWA) [15], developed by RandomStorm, is an open-source PHP/MySQL web application organized into modules associated with specific vulnerabilities, many of which correspond to those described in the OWASP Top 10. Each of these modules contains a low, medium, and high security example implementation that portrays the vulnerability. I reviewed the modules of the DVWA when building the components of *Porous*.

Unfortunately, when I worked with them, several of the modules in the DVWA had incorrect methods for securing against vulnerabilities [16]. In addition, some modules secured against entirely different vulnerabilities than they claimed. For example, the supposed secure implementation of the Brute Force DVWA module still allows for brute force attacks. Furthermore, the low and medium security implementations of it do not even portray a brute force attack but rather a SQL injection flaw. Due to these aforementioned reasons, I disregarded the DVWA as a reliable source of information for this thesis project.

2.3 Technologies Used

2.3.1 LAMP Stack

According to W^{3}Techs [17], who conduct surveys based on the Alexa top one million websites [18], approximately 82% of website whose underlying technologies are known are written in PHP. Additionally, 58.4% use Apache as the web server and 52.2% use Linux as the operating system on which the server runs. While no exact statistics are provided, MySQL is also claimed to be the most popular open-source relational database management system used to store web application data.

The combination of these technologies make up what is known as the LAMP stack (Linux, Apache, MySQL, PHP), which, based on these statistics, is the most popular set of technologies used for the development and deployment of web applications. Consequently, I used these technologies as the basis for the development of *Porous*.

2.4 Target Audiences

Based on the goals of this project, three target audiences have been identified as potential users of the *Porous* framework: *Framework Developers, Application Developers*, and *Application Users.*

2.4.1 Framework Developers

Framework developers are users who may contribute to the development of the web application framework or its components by extending or modifying it. In order to contribute to the framework and its components, framework developers must have advanced knowledge of object-oriented programming and have the necessary security knowledge to correctly implement components without introducing inadvertent vulnerabilities.

2.4.2 Application Developers

Application developers are users who develop web applications using the *Porous* framework. In order to efficiently develop applications using *Porous*, application developers must have had experience developing web applications with other web application frameworks.

2.4.3 Application Users

Application users are users who interact with the web applications built using the framework. These users may include students or professionals participating in a cyber security course or a cyber challenge. Depending on the course or challenge, application users may be asked to interact with an application in different manners by possibly exploiting or patching the application. Subsequently, the expected knowledge of an application user is dependent on how they will be interacting with the application.

CHAPTER 3

METHODOLOGY

This chapter consists of two sections. The first section, framework architecture, delves into the design decisions and implementation details of the framework's structure. It also introduces any concepts or patterns that were employed in its construction. The second section, testing procedures, describes the experiments and any metrics, both qualitative and quantitative, which were used to measure how effectively the framework was able to satisfy the goals of this project as defined in Section 1.3.

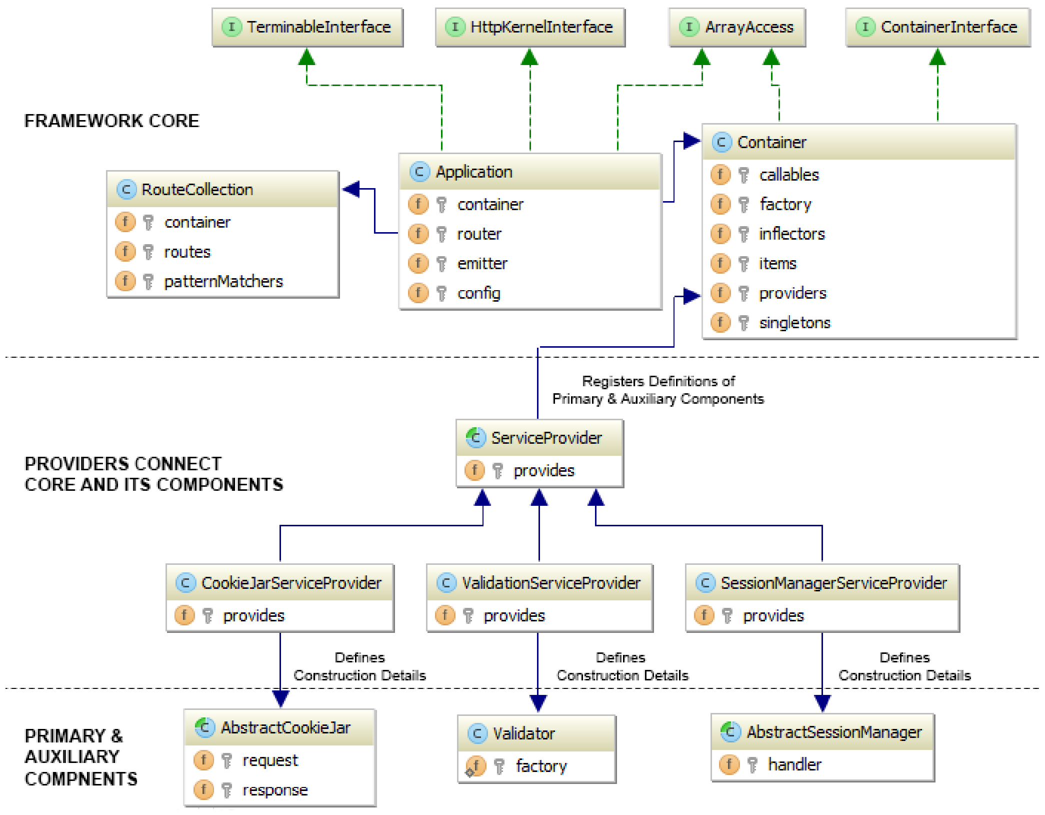
3.1 Framework Architecture

The architecture of the framework can be divided into three conceptual areas: *the core, primary components*, and *auxiliary components*. The core is responsible for bootstrapping, configuring, and running a web application. It also provides the extensibility mechanisms necessary for integrating both the primary and auxiliary components. On its own, the core is actually a fully functional micro framework comparable to the likes of Silex [7] or Slim [6]. Together with the primary and auxiliary components, it lies closer to being a full-stack framework. I had to make the design decision whether to either build on top of one of these existing frameworks or to build an entirely new one. When I was considering building on top of Slim, I discovered that it was too restrictive to meet the goals of the project. Slim aims to be as small as possible and therefore did not offer the extensibility or features that were necessary for Porous to meet the goal of section 1.3. When considering Silex, which is built entirely of Symfony components, I decided that a curation of components from different members of the open-source community would offer greater flexibility. Details of which components I chose, and why, can be seen in Section 3.1.1.

When designing the framework architecture, I made a distinction between *primary components*, which are directly essential for Porous to meet its goals, and *auxiliary components*, which are not essential for Porous to meet its goals. I created a different implementation of each of the primary components in order to provide both a vulnerable and secure version. These implementations may also include configuration options to fine tune their security features. Conversely, I either developed auxiliary components or took them from the community to provide supporting features for the application developer, or to act as dependencies for the core or primary components.

The following diagram displays the overall architecture of the Porous framework. The top section shows the architecture of the framework’s core. The bottom section displays a few of the primary and auxiliary components that are included with the framework. Lastly, the middle section shows how these primary and auxiliary components connect to the framework’s core. The subsections that follow will look further into the implementation details of the framework’s core, the primary components, and the auxiliary components.

Figure 5. Architecture of the Porous Framework



3.1.1 Framework Core

As previously mentioned, the framework core provides the functionality for bootstrapping, configuring, and running a web application. The core is implemented as a class called *Application*, which is the main entry point into any web application built using the framework. An application developer builds applications by calling methods from this class. The Application class contains properties and methods for interacting with the rest of the framework's components, as well as additional helper methods for the application developer. The core's subcomponents, which were taken from the open-source community, include a dependency injection container and a routing layer, which are important aspects in supporting the Porous framework's extensibility

3.1.1.1 Dependency Injection Container

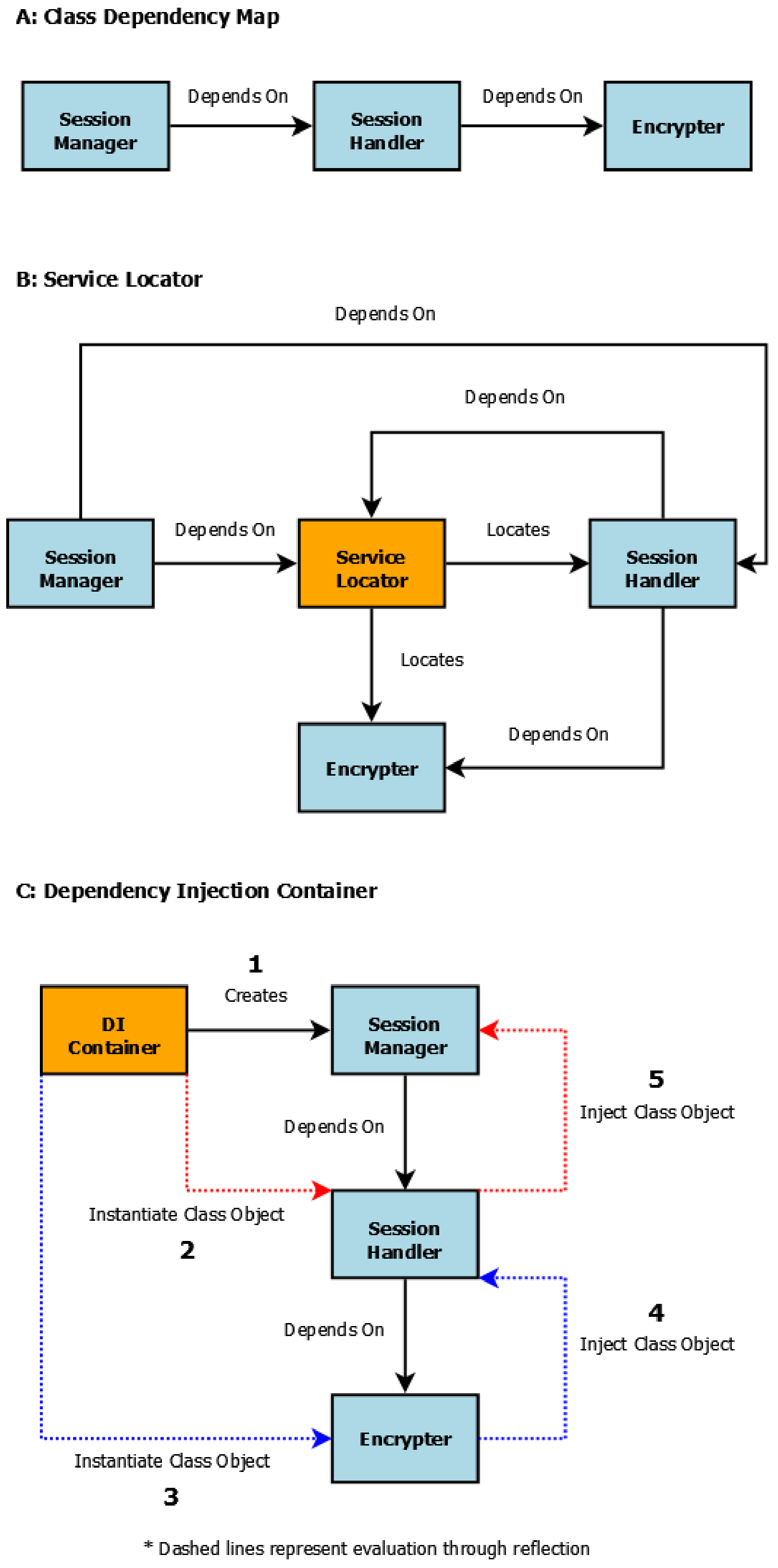
Together, the Application class and the dependency injection container act as the cornerstone of the framework's core by being the connective "glue" that binds the primary components, the auxiliary components, and any configuration settings together in the framework. The Dependency Injection Container allows for new features to be added and for existing components to be swapped out for different implementations. This allows application developers to choose between different vulnerable and secure implementations of components at their own discretion, and is vital to meeting the goals of the thesis by enabling the framework to be extensible and modular.

In order to fully comprehend the dependency injection container's role in the framework, it is first necessary to understand the related object-oriented design principles that it was built to employ. First is the concept of *inversion of control* (IoC) [19], which is the delegation of control over the program flow to some entity. A basic example of this concept is *event driven programming*, where instead of being executed sequentially, certain instructions are executed upon the arrival of different events. In the case of the framework, the concept of inversion of control is implemented through dependency injection, which is a design pattern [19] where the responsibility of locating or constructing dependencies is separate from the code that uses those dependencies. The dependency injection pattern can be implemented by either inverting control to a service locator or to a dependency injection container; the latter of which is done in the framework.

Both service locators and dependency injection containers act as a central repository for dependency definitions and instructions for their construction. Service locators can in certain circumstances be considered an anti-pattern [20] however, since every component that uses the locator is aware of its existence. Therefore, the locator itself is a dependency that must be included in these components. Conversely, with a dependency injection container the components are unaware of the container's existence. Instead, the application uses reflection, which is the ability for a program to inspect itself at runtime, to determine what dependencies need to be constructed and injected.

An example of how these patterns work can be seen in the diagrams of Figure 6 below. Diagram A: *Class Dependency Map* shows the relationship of the classes in the secure implementation of the session manager component in which the *SessionManager* class depends on a *SessionHandler,* which depends on an *Encrypter*. Diagram B: *Service Locator* shows how the injection of these dependencies would work when using the service locator pattern. The session manager would first ask the service locator to locate the session handler. When the session handler is found it asks the locator to locate the encrypter. Diagram C: *Dependency Injection Container* shows how the injection of these dependencies would work when using the dependency injection container pattern. Here, the application would ask the container to construct a session manager. Using reflection, the container would recognize that the session manager requires a session handler parameter to be constructed. The container then checks the service definitions it stores and constructs a session handler. This process is repeated when the container recognizes that the handler requires an encrypter.

Figure 6. Managing Class Dependencies with IoC



To further augment the benefits of using a dependency injection container the *dependency inversion principle* is also employed. The dependency inversion principle, which is one of the five basic object-oriented programming and design principles as identified by Robert C. Martin [21], states that both high-level and low-level components should rely on abstractions, rather than concrete implementations. The significance of this is that by having components depend on an interface or abstract class rather than a specific implementation, the component can be constructed using any other component that adheres to the interface or extends the abstract class.

The diagram in Figure 7 portrays the dependency injection principle being employed within the dependency injection container. This example is identical to that of Diagram C in Figure 6 with the exception that the *SessionHandler* and *Encrypter* classes have been abstracted to interfaces.

Figure 7. DI Container with DIP Principle Applied

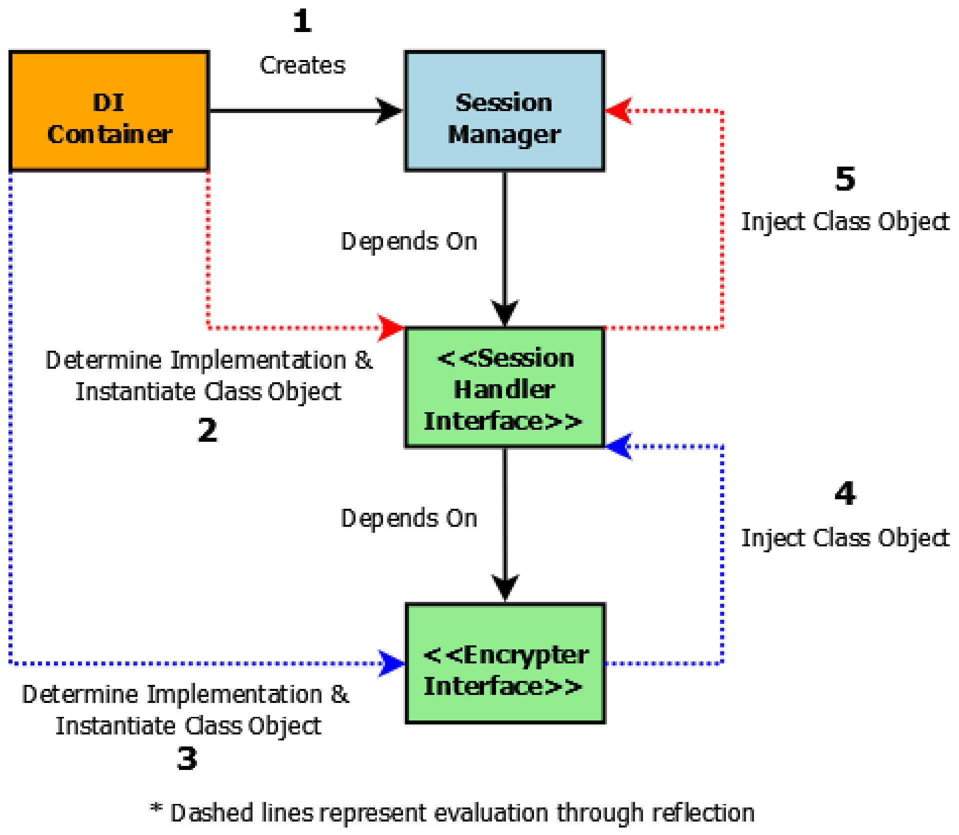
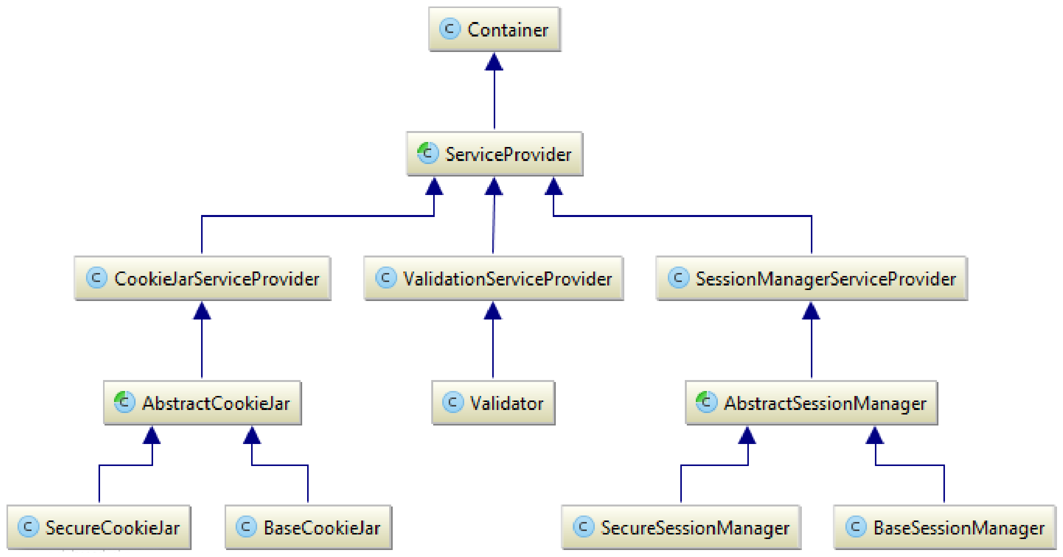

1
Creates
DI Container
Session Manager
5
Inject Class Object
Depends On
Determine Implementation & Instantiate Class Object
2
<<Session Handler Interface>>
4
Inject Class Object
Depends On
<<Encrypter Interface>>
Determine Implementation & Instantiate Class Object
3
* Dashed lines represent evaluation through reflection

Figure 8 shows the relationship between sample component implementations and the provider classes that define their construction details, which are then registered in the container for use in the framework.

Figure 8. Diagram of Components Registered to Container



When determining which dependency injection container to use in the framework, the following contenders from the open-source community were considered:

- Illuminate Container [22]: Created by Taylor Otwell and used in the Laravel and Lumen [23] frameworks
- Pimple [24]: Created by Fabien Potencier of SensioLabs and used in the Silex framework
- Container [25]: Created by Phil Bennett and released through The League of Extraordinary Packages (The PHP League)

To decide between these different containers, they were compared for: the features they provided, the completeness of their documentation, the state of their development, the complexity of their use, and overall size of the code base including any dependencies. This comparison is shown in Table 1. From it, Phil Bennett's *Container* was chosen.

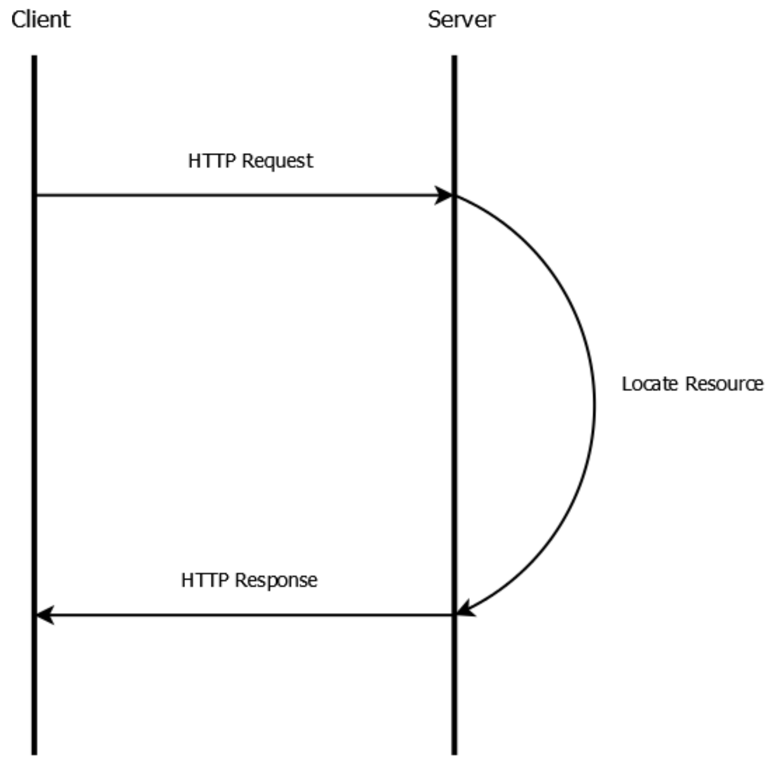
Table 1. Comparison of Dependency Injection Containers

Package	Features	Documentation	Development	Complexity	Size
Container	✓	✓	✓	✓	✓
Illuminate Container	✓	✗	✓	✗	✗
Pimple	✗	✓	✓	✓	✓

3.1.1.2 Routing Layer

Web applications operate through the Hypertext Transfer Protocol (HTTP) [26], which expresses the conventions for a transaction of messages between a client and server using a stateless request-response pattern. More specifically, the procedure for a HTTP transaction is that a user interacts with a client, typically a web browser, to send a request message containing a Uniform Resource Identifier (URI) that identifies both the server being contacted and the resource being requested. The contacted server then attempts to locate the resource and returns an appropriate response message to the client that is typically a HTML document rendered to the user. This interaction is shown in Figure 9.

Figure 9. Flow of a HTTP Transaction



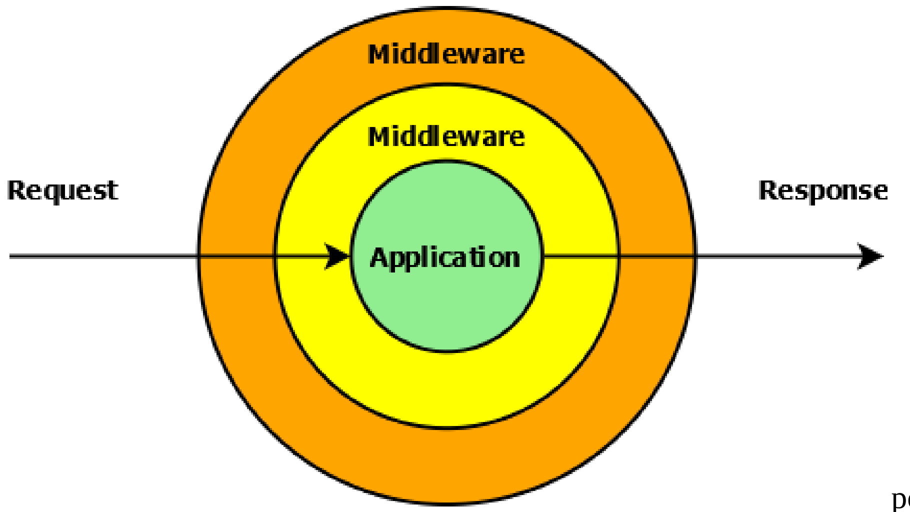
In the traditional architecture of a web application, requests to resources within the application are handled by individual script files that handle any tasks and return a response. The Porous framework uses an alternative approach called the *front-controller pattern*, where all requests are handled at a centralized point and the resource itself is either programmatically generated or delegated to by the central application. The role of the routing layer is to handle the request-response process from the entry point to the web application.

The routing layer was created using a combination of open-course components. The backbone of this layer consists of the *Symfony HttpFoundation* [27] component and the *HttpKernelInterface* from the *Symfony HttpKernel* component [28]. These components, amongst others, are part of a set of decoupled libraries used in the Symfony framework. These specific components are also used in other popular frameworks and projects including Silex, Laravel, Lumen, Drupal [29], and phpBB [30].

In native PHP, data from HTTP request and response messages are stored across several of the language's built-in superglobals, which are predefined variables that are accessible everywhere in an application, including in all scopes. These superglobals however, do not conform to the HTTP specification for HTTP request and response messages and therefore require developers to address any gaps on their own. As a result, the PHP Framework Interoperability Group (PHP-FIG) [31], which is a group of representatives from various projects that creates and votes on standards that are used to promote the reusability and sharing of code between each other's projects, has recently voted to accept the PHP Standard Recommendation 7 (PSR-7) [32]. PSR-7 defines a set of HTTP message interfaces, therefore providing a common, reusable layer for interacting with the HTTP protocol in an object-oriented manner. The HttpFoundation component was largely influential in the creation of this standard and provides implementations of these interfaces. Furthermore, by using the HttpKernelInterface the framework is obligated to handle HTTP transactions by accepting a request object and returning a response object.

The additional impact of using this standardized layer is that project-agnostic middleware for hooking into the request-response process can be used by any framework or project that implements these components. Specifically, any project that uses the HttpKernelInterface can integrate middleware using the decorator pattern. Igor Weidler's Stack library [33] has been included with the routing layer and simplifies the composition of HttpKernelInterface middleware by modeling them as layers being pushed onto a stack. An example of this can be seen in Figure 10. The request object enters the stack and is processed by each middleware that decorates the application. The response object is then processed in a similar fashion.

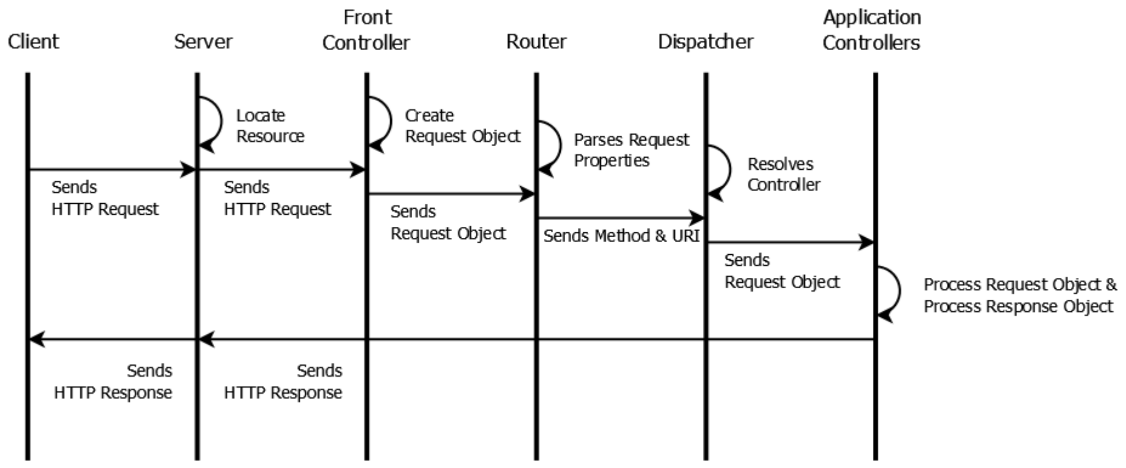
Figure 10. Stack of Middlewares Decorating Application



portrays

The actual handling of the request and response objects created using the HttpFoundation component is performed by Phil Bennett’s *Route* package [34]. The Route package allows for the definition of resource controllers by specifying a HTTP request method and a Uniform Resource Locator (URL). The package’s router inspects incoming requests to obtain this information and passes it to the dispatcher, which then interprets this information to locate and dispatch the controller that then builds and returns a response. As opposed to Figure 9, which shows the flow of HTTP messages through a basic HTTP transaction, Figure 11 shows the flow of HTTP messages as they traverse the routing layer of the application.

Figure 11. Routing Flow through Application



3.1.2 Primary Components

The primary components of the framework are those that are necessary to provide the dynamic logic of a web application. These components include a *cookie jar*, a *session manager*, and a *database manager*.

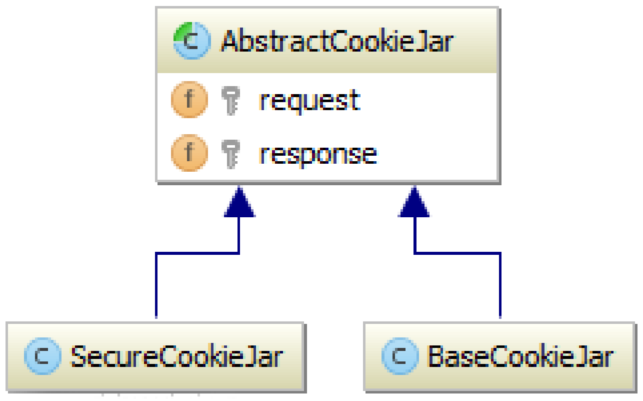
This section will examine both the overall architecture and implementation details of each of these components. Since the goals of the Porous framework included being evocative of current vulnerabilities and to provide configurable security features, there are two implementations of each component: a base implementation with no security features and is therefore vulnerable; and a secure implementation with configurable security features to fine tune how secure the component is. When discussing the vulnerabilities exposed or mitigated by these components, the associated category from the OWASP Top 10 is referenced in parenthesis by its list identifier (A1 – A10).

3.1.2.1 Cookie Jar

The cookie jar component deals with the creation and management of HTTP cookies. HTTP cookies are small pieces of data (limited to 4096 bytes) that are sent in the headers of a HTTP response message to be stored as a text file in a client web browser. They are also sent back to the server on each request in the headers of a request message. The data stored in a cookie may vary based on the individual use cases of the web application. Most commonly, cookies are used to store personalization settings, tracking information from advertisers, remember-me tokens for logging a user back in, and session identifiers, which are used to authenticate users of the web application. Other sensitive information may be stored in a cookie at the discretion of the web application developer. For these reasons, cookies can often be the target of malicious users.

The cookie jar component is the simplest of the three primary components. It is implemented as an abstract class that defines methods for managing instances of the *Symfony Cookie class*, which is part of the HttpFoundation component. The vulnerable and secure implementations of the cookie jar extend this abstract class and are required to implement methods for creating cookies and reading cookie data, as shown in Figure 12.

Figure 12. Architecture of Cookie Jar Component



A cookie consists of a name, a value, and a number of attributes that are given default values if not specified. It should be noted that cookie attributes are not sent back to the server but are only used by the browser to determine if the cookie should be deleted and if the cookie name and value should be sent to the server. The areas of interest regarding the security of cookies includes the value and the attributes. The value of the cookie is the actual data being stored. The attributes include the following:

- *expires* – The time the cookie expires set as a Unix timestamp. If omitted or set to zero, the cookie will expire when the client browser closes.
- *domain* – The domain or subdomain the cookie is available to. Defaults to the domain and all of its subdomains.
- *path* – The path on the server the cookie is available to. Defaults to '/', which indicates that the cookie will be available within the entire domain.

- *httponly* – Whether or not the cookie is accessible by client-side scripting languages such as JavaScript. Defaults to true.
- *secure* – Whether or not the cookie should only be transmitted over a secure HTTPS connection. Defaults to false.

The vulnerable cookie jar implementation creates cookies whose values are stored in a plaintext format, which is human readable. It also leaves all of the attributes with their default settings.

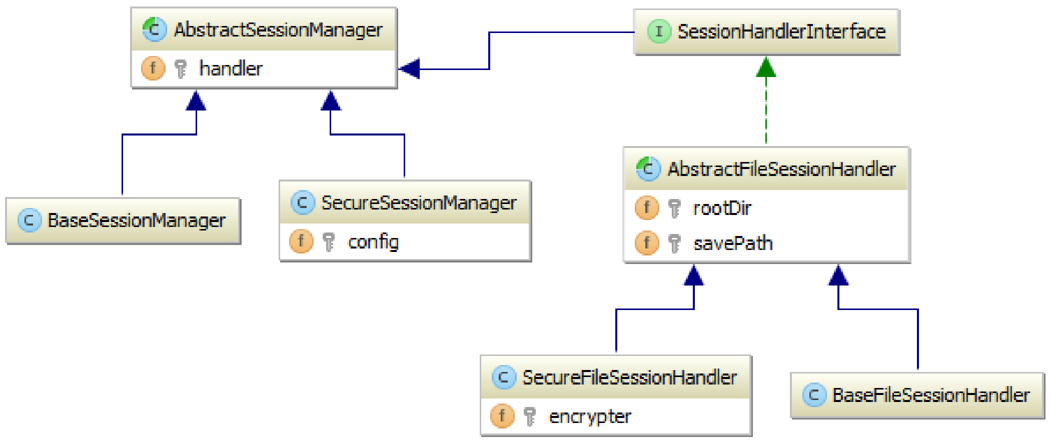
The secure cookie jar implementation is slightly more complex in that it has configuration settings to optionally encrypt and optionally sign a cookie's value using the Cryptography auxiliary component described in Section 3.1.3.3. Encrypting the cookie's value obfuscates its data into ciphertext that is no longer human readable. In order to read the cookie's value it would first need to be decrypted back to plaintext using the same cipher and key used to encrypt it. Encrypting the cookie's value mitigates possible attacks due to sensitive data exposure (A6). Signing the cookie's value creates a unique digital signature of the data. Any change to the cookie's data would result in a different digital signature. By appending this signature to the cookie value the cookie jar component can check if a cookie's data has been altered by unauthorized sources. This mechanism mitigates cross-site scripting (XSS) (A3) attacks in which a malicious user would store JavaScript to be executed in the browser inside of the cookie. Lastly, the secure implementation enables the *httponly* attribute by default, which also mitigates XSS (A3) attacks by preventing scripting languages from accessing the cookie.

3.1.2.2 Session Manager

HTTP is a stateless protocol, meaning that data is not preserved when making subsequent requests to a web application. In order to maintain state, sessions are employed, which store an identifier cookie on the client side that refers to the actual session data stored on the server side. The session manager component is therefore responsible for supervising how both the session identifiers and session data are stored. The data stored in sessions is typically used to authenticate users within a web application. The security of session data is therefore imperative as authenticated users may have access to sensitive information or may be given the authorization to perform additional functions not intended for unauthorized users.

The session manager has two parts, the manager itself and the session handlers. The manager acts as a wrapper around the functions PHP provides for working with sessions, and also provides additional helper methods that allow the application developer to add additional security mechanisms. The manager is implemented as an abstract class that is extended by the vulnerable and secure manager implementations. The vulnerable implementation simply calls the built-in PHP functions, while the secure implementation adds additional logic that is considered to be best practices when working with these functions. For instance, when starting a session on a new request, the secure implementation can optionally regenerate the session identifier, which can help prevent session fixation attacks since the old identifier is no longer tied to the session data. Additionally, both implementations include methods for the application developer to optionally expire a session after a period of idleness, bind a session to the IP address and/or user agent of a client, and generate a cross-site request forgery (CSRF) token. The diagram in Figure 13 shows the architecture of the session manager component. It should be noted that additional abstract session handlers exist, however only the *AbstractFileSessionHandler* is shown for simplicity. Additional information regarding these session handlers is provided in the text that follows.

Figure 13. Architecture of Session Manager Component



The actual storage of session data and construction of session identifiers is controlled by PHP's session handlers, whose methods are kept internal and are not exposed to developers. The functions PHP provides for developers call these internal methods to perform their prescribed tasks. As of PHP 5.4.0, the *SessionHandlerInterface* was introduced, which allows developers to create custom session handlers by overriding these internal methods. The session manager is constructed by first passing in an implementation of the *SessionHandlerInterface*. These custom handlers allow developers to control where and how the data is stored and how the session identifier cookie is created. The Porous framework provides four custom session handlers. These include both a vulnerable and secure implementation of a file-based handler, and a vulnerable and secure implementation of a database-based handler. The secure implementations of the file and database handlers can be optionally configured to encrypt the session data and can optionally encrypt, sign, and set attributes to the session identifier cookie similarly to what was discussed in the previous section. Additionally, the storage path of file-based sessions can be set.

The security mechanisms described in this section can help mitigate vulnerabilities of sensitive data exposure (A6), flaws due to broken authentication and session management (A2), and cross-site request forgery attacks (A8).

3.1.2.3 Database Manager

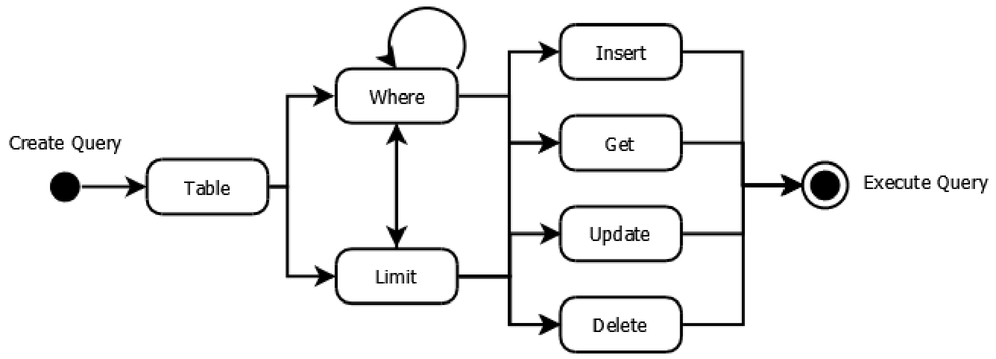
As explained by Anthony Ferrara, a Developer Advocate at Google, a common model for web applications is to present them as a union of *n*-tiers [35] that are responsible for conceptually different processes. Almost all web applications utilize at least two tiers that enable their dynamic nature. The first tier is the *application server* that controls the logical operations of the web application. The second tier is the *database* that is used to store the actual data used by these logical operations. The database manager component acts as an abstraction layer for communication between the application logic and the data stored in the database.

The implementation of the database manager consists of three subcomponents: the *database connector*, the *query builder,* and the *compiler*. The database connector employs the factory pattern to create data source names (DSN), which are formatted strings that describe a connection to a data source. The connector passes this DSN to a PHP Data Object (PDO) to create a database connection. PDO supports drivers for a number of different database types; however the factory only supports creating DSNs for MySQL, PostgreSQL, and SQLite at this time.

The query builder is the main subcomponent with which application developers interact. It is implemented as a fluent interface in which methods are chained together to build an object whose properties represent the different clauses of a structured query language (SQL) statement. The utilization of a fluent interface gives developers a readable API for building queries. The query builder is also responsible for passing compiled queries to the database connection to be executed. The query builder is also implemented as an abstract class whose subclasses are responsible for determining how the query is to be executed. The vulnerable implementation uses the PDO’s *query* method, which simply takes a raw SQL statement and executes it on the database server. The secure implementation uses PDO’s *prepare* and *execute* methods for creating prepared statements and then binding the values of variables to these statements that are then executed. A prepared statement is analogous to a template that is precompiled in the database driver, and therefore cannot be modified when variables are passed into it. As a result, prepared statements are immune to SQL injection vulnerabilities (A1), which can cause sensitive data exposure (A6) by allowing malicious users to execute statements that may read or modify data in the database.

Below, Figure 14 presents a subset of the state machine for the fluent query builder. Each state represents a method that is chained onto the requisite methods. Each query begins by specifying the table you are working with. There are then numerous methods that may be chained together to specify the data you are working with. A query is completed by calling a method that indicates what is to be done with the data.

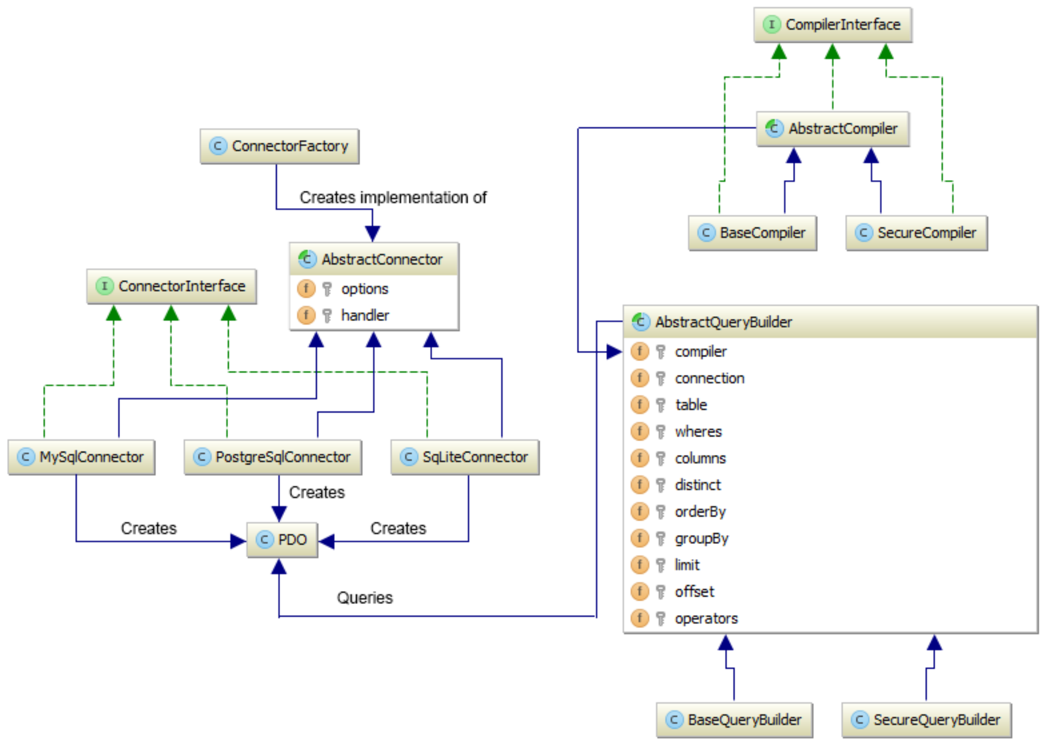
Figure 14. Subset of Fluent Query Builder State Machine



The last subcomponent is the compiler, which takes the properties of a query object and translates them into a SQL statement. The compiler consists of an interface that describes the methods needed to generate different types of SQL queries. The vulnerable and secure implementations implement these methods to return raw SQL statements and prepared statements with bound parameters, respectively.

The overall architecture of the database manager component can be seen in Figure 15, which also shows the relationships between the database manager's subcomponents. To recap, the *ConnectionFactory* creates an implementation of an *AbstractConnector*, which specifies a data source to be used when creating a PDO connection. Query objects are then generated using a fluent interface from an implementation of the *AbstractQueyBuilder*. These objects are then compiled to SQL statements by an implementation of the *AbstractCompiler* and executed.

Figure 15. Architecture of Database Manager Component



3.1.3 Auxiliary Components

The last part of the framework's architecture is the addition of auxiliary components that act as either dependencies for other components in the framework or provide supplementary functionality for application developers to use. These auxiliary components include the *Event Manager*, *Logger*, *Cryptography*, *Validation*, and *Template Engine*.

3.1.3.1 Event Manager

The event manager component allows application developers to hook into the web application by using the Publish-subscribe design pattern for event-driven programming. By default, the Porous framework has event listeners registered to listen for events that occur during the request-response cycle - including when a request is received, a response is created, and when a response is sent back to the client. The event manager may be used to automate logging when events occur, or to support the business needs of more complex applications.

The event manager used in the Porous framework is the open-source package aptly called *Event*, which was developed by Frank de Jonge [36]. Other event managers considered for inclusion in the framework included Symfony's *Event Dispatcher* [37] and Sabre's *Event Emitter* [38]. Symfony's Event Dispatcher is by far the most popular solution as it is used by the Symfony framework. However, it introduces several dependencies that would substantially increase the overall size of the Porous framework. Comparatively, neither Event nor Sabre's Event Emitter require any additional dependencies. I chose Event over Event Emitter due to its more exhaustive documentation and due to having more than double the install base (~81,000 installations vs ~39,000 at the time of this writing).

3.1.3.2 Logger

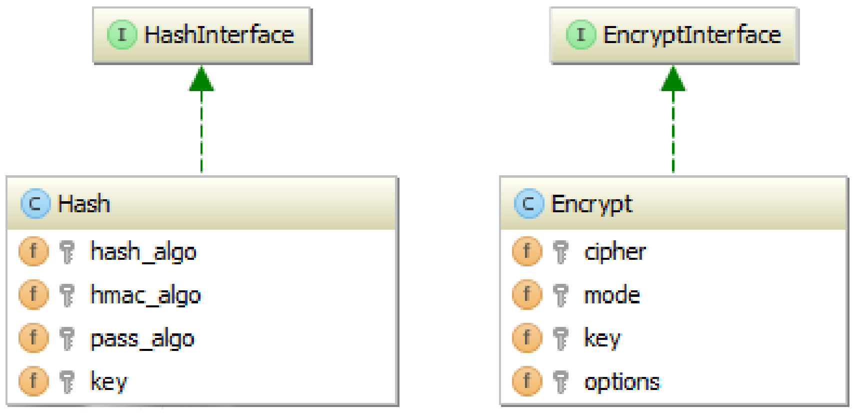
The logger component was included to allow developers to log different events that occur within a web application. Specific use cases may include to record error messages, track data, or to provide and audit trail for different actions. The log files generated by a web application could be used to support application users participating in a cyber challenge or to help provide additional information for cyber challenge moderators.

The logging package chosen for inclusion in the framework is Monolog [39], which was developed by Jordi Boggiano. Monolog is the most popular logging library available in PHP.

3.1.3.3 Cryptography

The cryptography component is the only auxiliary component that was developed rather than taken from the open-source community. This component is a dependency for the secure implementations of both the cookie jar and session manager components. The component includes interfaces for encryption and hashing methods as well as an implementation of each interface, as shown in Figure 16.

Figure 16. Architecture of Cryptography Component



The included Encrypt class acts as a wrapper around PHP's MCrypt extension [40] that can be configured to perform the different types of encryption supported by MCrypt. By default, the Encrypt class is configured to perform encryption using the Rijndael algorithm [41], which is used by the Advanced Encryption Standard (AES) [42] selected by the U.S. National Institute of Standards and Technology (NIST).

The included Hash class acts as a wrapper around PHP's *hash*, *hash_hmac*, and *password_hash* functions and includes helper methods for comparing hash values.

The *hash* function creates a digital signature of data, the *hash_hmac* function creates a keyed-hash message authentication code (HMAC), which can be used to sign data, and the *password_hash* function is used specifically for hashing passwords using the bcrypt algorithm, which is based on the Blowfish cipher [43]. By default, the Hash class is configured to use the SHA-256 algorithm [44] for both the *hash* and *hash_hmac* functions. While the *password_hash* function uses bcrypt, the Hash class can override the method to use a less secure hashing algorithm such as MD5 in order to introduce vulnerabilities for applications built using this framework.

3.1.3.4 Validation

The implementations of the primary components are able to introduce and mitigate a wide range of vulnerabilities. However, they do not support any form of data validation, which can be imperative when properly securing a web application. Since the validation of data is reliant on the context and type of data being validated, it is left to the application developer to properly perform. To secure a web application it is expected that all user input is filtered when output. The validation library I chose for inclusion in the Porous framework is Respect's *Validation* developed by Henrique Moody [45]. According to Chris Cornutt, PHP security expert and member of Hewlett-Packard's Global Cyber Securtiy Group, Respect's *Validation* library has become one of the de-facto standards for doing data validation in PHP [46].

3.1.3.5 Template Engine

A template engine was included in the framework to assist application developers in creating the HTML documents for their web applications. Three different template libraries were considered for the framework including SensioLab's *Twig* [47], Illuminate's *Blade* [48], and the PHP League's *Plates* [49]. Both Twig and Blade are compiled templates, while Plates is a native template engine that was inspired by Twig. When considering these options, size was a major determining factor in choosing which engine to include. With its dependencies Illuminate's Blade is ~3 MB in size, Twig ~1 MB, and Plates only ~40 KB. Ultimately, I chose Plates for the Porous framework due to its size and minimal learning curve.

3.2 Testing Procedures

The previous section described the methodologies used to implement the Porous web application framework. This section will identify the experiments that were conducted and how they were used to evaluate the effectiveness of this solution in meeting the goals defined in Section 1.3.

3.2.1 Primary Component Vulnerability Tests

The first set of experiments conducted were used to assess the presence of vulnerabilities in the framework's primary components. Each of these experiments, unless otherwise noted, were conducted twice: once for the vulnerable implementation, and once for the secure implementation with its security features configured. In order to test for vulnerabilities in these components, tests directly from the OWASP Testing Guide (OTG) [14] were used when applicable. These tests are referenced by their identifier in the form of OTG-CATEGORY-###. It should be noted that in many cases only portions of an OTG test were completed since a considerable amount of them rely on an application's business logic rather than a component's implementation. In cases where no OTG test was available to assess a vulnerability, additional procedures were established by considering information from the OWASP Top 10 or through general understanding of security principles and the underlying technologies.

A summary of these tests can be seen in the table below:

Table 2. Summary of Primary Component Tests

Cookie Jar Tests	
Test	**Description**
Testing for Cookies Attributes (OTG-SESS-002)	Tests the appropriate setting of cookie attributes.
Testing for Information Leakage (Segment of OTG-SESS-001)	Tests the human readability of cookie data.
Testing for Tamper Resistance (Segment of OTG-SESS-001	Tests for the ability to modify cookie data.
Session Manager Tests	
Test	**Description**
Testing for Cookies Attributes (OTG-SESS-002)	Tests the appropriate setting of session cookie attributes.
Testing for Information Leakage (Segment of OTG-SESS-001)	Tests the human readability of the session identifier and stored session data.
Testing for Tamper Resistance (Segment of OTG-SESS-001	Tests for the ability to modify session cookie data.
Testing for Session Fixation (Segment of OTG-SESS-003)	Tests the regeneration of session identifiers.
Testing for Cross Site Request Forgery (OTG-SESS-005)	Tests for possibility of CSRF attacks.
Testing for Session Validity (OTG-SESS-007)	Tests the ability to verify a session's authenticity.
Database Manager Tests	
Test	**Description**
Testing for SQL Injection (OTG-INPVAL-005)	Tests the possibility of SQL injection.

3.2.1.1 Cookie Jar Tests

To test for vulnerabilities in the different implementations of the cookie jar component the following experiments were performed:

3.2.1.1.1 Testing for Cookie Attributes (OTG-SESS-002)

The purpose of this experiment was to verify that appropriate default settings for cookie attributes were applied to cookies generated by the cookie jar component. More specifically, this experiment tested for the enabling of the *httponly* attribute, which when enabled, prevents access to cookies from client-side scripting languages such as JavaScript. The omission of this attribute introduces a vulnerability to cross-site scripting (XSS) attacks.

It should be noted that the referenced test OTG-SESS-02 also recommends testing the *secure*, *domain*, *path*, and *expires* attributes. However, the setting of these attributes is dependent on the context of the cookie within the application, and is therefore left to the application developer to implement correctly. Therefore, rather than testing for an appropriate default setting, I tested the ability to set these attributes instead.

To perform this experiment, a web application was created that generated a cookie. The attributes and their values were then inspected by observing the HTTP response headers sent by the web application using the OWASP Zed Attack Proxy (ZAP) [50], an intercepting proxy and web application penetration testing tool.

3.2.1.1.2 Testing for Information Leakage (Segment of OTG-SESS-001)

The purpose of this experiment was to check for the possibility of information leakage by storing the cookie's data in plaintext, a human readable format. The storing of data in plaintext introduces a vulnerability to sensitive data exposure through network eavesdropping or local machine access.

To perform this experiment a web application was created that generated a cookie with a name of "`foo`" and a value of "`bar`". Using ZAP, the HTTP response headers were inspected to check the readability of the cookie's value.

3.2.1.1.3 Testing for Tamper Resistance (Segment of OTG-SESS-001)

This purpose of this experiment was to test a cookie's resistance to malicious attempts of modification. A lack of resistance to such modification introduces a vulnerability to an exploit known as cookie tampering (also known as cookie poisoning), which may be used to perform a variety of attacks.

To perform this experiment, a web application was created that generated a cookie named "`foo`" with a value of "`bar`". The web application then rendered the value of the cookie by printing it in the client web browser, Google Chrome. The value of the cookie was then modified using the Google Chrome extension EditThisCookie. Once modified, the resource that rendered the cookie's value was refreshed and the value printed in the browser was inspected.

3.2.1.2 Session Manager Tests

The tests in this section were performed to test for vulnerabilities in the session manager component and its session handlers. Additionally, tests were performed to verify that the helper methods provided for application developers were functioning properly.

3.2.1.2.1 Testing for Cookie Attributes (OTG-SESS-002)

The purpose and procedure of this experiment is identical to that of experiment 3.2.1.1.1, but within the context of a session cookie, which is handled independently of the cookie component. In addition to the *httponly* attribute, the configuration of the *secure* attribute was tested.

3.2.1.2.2 Testing for Information Leakage (Segment of OTG-SESS-001)

The purpose and procedure of this experiment is identical to that of experiment 3.2.1.1.2, but within the context of a session cookie. In addition, the possibility of information leakage of session data stored on the server was tested by its readability as plaintext. Depending on the handler, the data was inspected either in the session files or the database table in which sessions were stored.

3.2.1.2.3 Testing for Tamper Resistance (Segment of OTG-SESS-001)

The purpose and procedure of this experiment are identical to that of experiment of 3.2.1.1.3, but within the context of a session cookie.

3.2.1.2.4 Testing for Session Fixation (Segment of OTG-SESS-003)

The purpose of this experiment was to test if sessions are vulnerable to *fixation attacks*. A session fixation attack occurs when an attacker forces a session identifier for a web application upon a victim. When the victim authenticates themselves with the web application, the same session identifier is used. Since the attacker knows what the identifier is, the attacker is able to hijack the session by using the now authenticated identifier.

To perform this experiment, a web application was created that requires a user to authenticate themselves using a predetermined set of credentials to access an administrative area. Firefox was then used to access the application and start a session. The session cookie was then copied into Google Chrome. The application was then logged into from Chrome using the predefined credentials. It was then checked if the administrative area could be accessed through Firefox.

3.2.1.2.5 Testing for Cross Site Request Forgery (OTG-SESS-005)

The purpose of this experiment was to test for the possibility of cross site request forgery (CSRF) attacks by verifying a web application's trust in requests from users that are made to it. Specifically, this experiment tests the Session Manager's methods for generating and verifying a CSRF token that is stored in a session's data.

To perform this experiment a web application was created that allows authenticated users to delete a database entry by submitting a form. After logging into the application a second web application was accessed that contained a hidden form that forges a request to the first application. It was then checked if the first web application honored the request and deleted the database entry.

3.2.1.2.6 Testing for Session Validity (Includes OTG-SESS-007)

The purpose of this experiment was two-fold: to test the session manager's ability to identify a user based on their user-agent and/or IP address and to test the session manager's ability to invalidate a session after a defined period of inactivity. This functionality provides a rudimentary defense against session hijacking attacks.

To perform this experiment, a web application was created that starts a session. Using OWASP ZAP, HTTP requests were then created that contained headers with user-agents and IP addresses that were different than the ones that started the session. It was then checked if the web application invalidated the session due to these changes. Additionally, the timeout functionality was tested for correctness by setting a predefined idle time and checking for a timeout after the prescribed amount of time.

3.2.1.3 Database Manager Tests

To test for vulnerabilities in the different implementations of the database manager component the following experiments were performed:

3.2.1.3.1 Testing for SQL Injection (OTG-INPVAL-005)

The purpose of this experiment was to determine if the queries compiled by the database manager component were vulnerable to SQL injection attacks, which could allow malicious users to read or modify the contents of a database potentially exposing sensitive data or causing harm to an organization.

To perform this experiment, a web application was created with the functionalities to create, read, update, and delete data from a database containing dummy data. Each of these operations were tested using SQLMap [49], an open-source penetration testing tool that automates the detection and exploitation of SQL injection flaws.

CHAPTER 4

FINDINGS

This chapter presents the results that were gathered by performing the experiments described in Section 3.2.

4.1 Primary Component Vulnerability Results

This set of experiments set out to assess the presence of vulnerabilities in the proposed vulnerable and secure implementations of each of the primary components. A summary of these results is shown in the table below followed by individual results for each of these experiments.

Table 3. Summary of Primary Component Tests Results

Cookie Jar Tests	
Test	**Pass / Fail**
Testing for Cookies Attributes (OTG-SESS-002)	Pass
Testing for Information Leakage (Segment of OTG-SESS-001)	Pass
Testing for Tamper Resistance (Segment of OTG-SESS-001	Pass
Session Manager Tests	
Test	**Description**
Testing for Cookies Attributes (OTG-SESS-002)	Pass
Testing for Information Leakage (Segment of OTG-SESS-001)	Pass
Testing for Tamper Resistance (Segment of OTG-SESS-001	Pass
Testing for Session Fixation (Segment of OTG-SESS-003)	Pass
Testing for Cross Site Request Forgery (OTG-SESS-005)	Pass
Testing for Session Validity (OTG-SESS-007)	Pass
Database Manager Tests	
Test	**Description**
Testing for SQL Injection (OTG-INPVAL-005)	Pass

4.1.1 Cookie Jar Results

To revisit the exact details of each of the following experiments for the cookie jar component refer to section 3.2.1.1.

4.1.1.1 Testing for Cookie Attributes (OTG-SESS-002)

This experiment set out to verify that the appropriate default configuration settings for cookie attributes were applied to each implementation of the cookie jar component. The results of this experiment were gathered by visual inspection of the HTTP response headers sent by the web application using the OWASP ZAP tool. Upon inspection of these headers, I determined that each vulnerable and secure implementations set the *httponly* attribute by having it set to false and true, respectively. Additionally, both implementations able to correctly set the *secure*, *domain*, *path*, and *expires* attributes on demand.

As a result of this experiment, I concluded that the vulnerable and secure implementations of the cookie jar component can accurately expose and mitigate vulnerabilities associated with setting of cookie attributes including certain instances of XSS attacks (A3).

4.1.1.2 Testing for Information Leakage (Segment of OTG-SESS-001)

This experiment set out to check for the possibility of information leakage caused by the storage of data in a cookie’s value in a human readable plaintext format. The results of this experiment were gathered by visually inspecting the HTTP response headers using the OWASP ZAP tool. Upon inspection of these headers I saw that the vulnerable implementation of the cookie jar showed the expected value of “bar”. Contrarily, the secure implementation obfuscated the data by utilizing the Cryptography component to encrypt and encode the cookie’s value. The exact values that were stored in the cookies tested during this experiment are in the table below.

Table 4. Testing for Information Leakage Comparison

Implementation	**Cookie Value**
Vulnerable	`bar`
Secure	`GmNegLfvVYlhG1gde4vs5NVIrkw01WUv2FEWcGuuI0c%3D`

As a result of this experiment, I concluded that the vulnerable and secure implementations of the cookie jar component can accurately expose and mitigate sensitive data exposure vulnerabilities (A6) associated with the storing of cookie data in plaintext.

4.1.1.3 Testing for Tamper Resistance (Segment of OTG-SESS-001)

This experiment set out to determine if a cookie was resistant to malicious attempts of modification. The results of this experiment were gathered by visually inspecting the output of the web application that rendered the cookie value in the web browser. Upon inspection of the rendered web page I saw that a cookie created using the vulnerable implementation could be modified and have its value rendered as normal. Contrarily, the value of a cookie created using the secure implementation was not rendered in the browser. Instead, the web application simply ignored the cookie altogether. This was due to the fact that the signature generated from the modified data could not be validated against the signature that was generated by the original data. In order to successfully modify a cookie’s data an attacker would need to reverse engineer the algorithm used to create the signature. By default, the cookie jar component uses an HMAC code to sign the data that would require the attacker to also gain access to the key used in the algorithm. However, the cookie jar can also be configured to use a weaker algorithm such as MD5, which is easily recognizable. If recognized, the attacker could then modify the cookie. The values rendered in the browser for the cookie data before and after modification are in the tables below.

Table 5. Cookie Data before Modification

Implementation	Stored Value	Value Displayed
Vulnerable	`bar`	`bar`
Secure (HMAC)	`bar--14b473a0d902a7a38187d2b2bc2e092d63050b110c9d9fe04be342cf97581eb5`	`bar`
Secure (MD5)	`bar--37b51d194a7513e45b56f6524f2d51f2`	`bar`

Table 6. Cookie Data after Modification

Implementation	Stored Value	Value Displayed
Vulnerable	`qux`	`qux`
Secure (HMAC)	`qux`	
Secure (HMAC)	`qux--14b473a0d902a7a38187d2b2bc2e092d63050b110c9d9fe04be342cf97581eb5`	
Secure (MD5)	`qux--37b51d194a7513e45b56f6524f2d51f2`	
Secure (MD5 – Recognized)	`qux--d85b1213473c2fd7c2045020a6b9c62b`	`qux`

As a result of this experiment, I concluded that the vulnerable and secure implementations of the cookie jar component accurately expose and mitigate vulnerabilities associated with the tampering of cookie data at a granular level.

4.1.2 Session Manager Results

To revisit the exact details of each of the following tests for the session manager component, refer to section 3.2.1.2.

4.1.2.1 Testing for Cookie Attributes (OTG-SESS-002)

This experiment set out to verify that the appropriate default configuration settings were set for the attributes of a session cookie. The results of this experiment are nearly identical to those found in Section 4.1.1.1. The exception to these results is due to the additional requirement that the secure implementations of the session handlers are configured to have session cookies sent over an encrypted connection by enabling the *secure* attribute. Upon visual inspection, I determined that the setting of this attribute was correct for both the vulnerable and secure implementations of the session handlers.

As a result of this experiment, I concluded that the vulnerable and secure implementations of the session handlers can correctly expose or mitigate vulnerabilities associated with the cookie attributes of a session cookie. These include possible vulnerabilities due to broken authentication and session management (A2), XSS (A3), and sensitive data exposure (A6).

4.1.2.2 Testing for Information Leakage (Segment of OTG-SESS-001)

This experiment set out to identify the possibility of information leakage by both the session cookie and the session data stored on the server. The results pertaining to the session cookie are identical to those in Section 4.1.1.2. The results of testing for information leakage in the session data was concluded in a similar manner by visually inspecting the location in which the session data was stored.

The vulnerable implementations of the session handlers store session data as key-value pairs in a human readable plaintext format. Conversely, the secure implementations of the session handlers store data in an obfuscated format that has been serialized as well as encrypted and encoded like the secure cookies. The exact data stored by the session handlers can be seen in the table below.

Table 7. Comparison of Stored Session Data

Implementation	**Session Data**
Vulnerable	`s:52:"username|s:7:"johndoe";email|s:16:"jdoe@example.com";";`
Secure	`s:108:"lvbdVMs9VMmMulAOkbrsGr00QMfVf/c8k0VodUfJtmMkdIW6ZDoL/6iS8Ut8Xfdp/gQoioxkAx1Q7Hlo2Rrgu5uf7lqIL0RcJOO0ZcDP8qM=";`

As a result of this experiment, I concluded that the vulnerable and secure session handlers correctly store data to expose and mitigate vulnerabilities pertaining to the leakage of information from session data. This includes possible vulnerabilities due to broken authentication and session management (A2), XXS (A3), and sensitive data exposure (A6).

4.1.2.3 Testing for Tamper Resistance (Segment of OTG-SESS-001)

This experiment set out to determine if a session cookie was resistant to malicious attempts of modification. The results of this experiment are identical to those in Section 4.1.1.3. However, additional implications of these results include the possible exposure to and mitigation of session hijacking attacks.

4.1.2.3.1 Testing for Session Fixation (Segment of OTG-SESS-003)

This experiment set out to specifically determine if sessions could possibly be vulnerable to certain incarnations of session fixation attacks. The results of this experiment were gathered by visually inspecting session cookies to see if their identifiers were regenerated and by attempting to bypass the authentication mechanism of the login form to directly access the administrative section of the web application. With the vulnerable implementation of the session manager, the session identifier is never regenerated. Therefore, when a session was authenticated in one browser it was also authenticated in the other browser that shared the same session identifier. Contrarily, with the secure implementation the session identifier was regenerated on each request, which invalidated the old session identifier after logging into the application.

As a result of this experiment, I concluded that the vulnerable and secure implementations of the session manager component appropriately expose and mitigate session fixation attacks (A2) that are reliant on the regeneration of session identifiers.

4.1.2.4 Testing for Cross Site Request Forgery (OTG-SESS-005)

This experiment set out to determine if the methods in the session manager could be used to verify a request coming into the application through the use of a CSRF token. The results of this experiment showed that these methods were working as intended since the forged request was not honored by the application. A CSRF token is stored in a user’s session data and is regenerated on every request. This token must appear as a hidden form field in any form that is submitted to the application in order to be verified. Since the forged request did not contain the session's token it was ignored by the application.

As a result of this experiment, I concluded that these methods do function correctly and may be used by the application developer to prevent CSRF attacks.

4.1.2.5 Testing for Session Validity (Includes OTG-SESS-007)

This experiment set out to determine if the methods in the session manager component could be used to check the validity of a session by associating it with client-specific data. The results of this experiment were gathered by inspecting the behaviors of the web application when HTTP request headers were forged with different user-agent strings and IP addresses. The results of this experiment showed that these methods were working as intended since the web application denied access to an authorized-only area of the web application. The timeout functionality of the session manager also worked as intended. When a request was made after an allowed period of idle time of three minutes, the session was destroyed.

As a result of this experiment, I concluded that these methods do function correctly and may be used by the application developer to introduce additional security features to a web application.

4.1.3 Database Manager Results

To revisit the exact details of each of the following tests for the database manager component refer to section 3.2.1.3.

4.1.3.1 Testing for SQL Injection (OTG-INPVAL-005)

This experiment set out to determine if the queries compiled by the database manager component were vulnerable to SQL injection attacks (A1). This experiment was broken down into separate tests for each of the major query operations: select, insert, update, and delete. The results of running SQLMap against different parts of a web application that uses these queries showed that the vulnerable implementation was vulnerable to SQL injection in all four cases. More specifically, SQLMap found the queries vulnerable to Boolean-based blind injection, error-based injection, AND/OR time-based blind injection and union query injection. As expected, SQLMap was unable to perform any injection attacks on the secure implementation of the component. As a result, I concluded that the vulnerable and secure implementations of the database manager component correctly expose or mitigate SQL injection vulnerabilities.

CHAPTER 5

DISCUSSION

5.1 Conclusions

The previous chapter presented the results that were collected by testing the web application framework designed and implemented using the methodologies described in Chapter 3. This chapter will now reflect on those methodologies and review the testing results to make implications regarding whether or not the methodologies used to develop the web application framework were able to achieve the goals defined in Section 1.3.

5.1.1 Goal 1 Conclusions

The first goal of this project was to develop a web application framework that is able to simplify the development of vulnerable web applications. The framework developed in this thesis was able to simplify the development of web applications by providing abstractions and interfaces to common web functionalities including routing and the management of sessions, cookies, and databases. This is evidenced by the creation of the primary components, which include methods for providing these functionalities. Furthermore, many of these abstractions were due to the inclusion of popular community projects and standards set by the PHP-FIG that were created specifically for simplifying the development of PHP web applications. Based on the application developer target audience, which was defined to have had experience with developing applications using native PHP and other web application frameworks, the developers creating applications should already have a familiarity with some of these components such as the HttpFoundation component and the HttpKernelInterface.

In addition, the Porous framework specifically sought to simplify the development of vulnerable web applications. This was achieved by building on these components and making abstractions to different security implementations of components. This was verified by the various primary component tests.

5.1.2 Goal 2 Conclusions

The second goal of this project was to develop a web application framework that provides configurable security features for introducing vulnerabilities to web applications. This goal was achieved by the inclusion of both a vulnerable and secure implementation of each primary component. As described in the methodologies sections for each component, the base implementation contains no security options by default. The secure implementations however, provide configuration options to granularly control the security mechanism for each component. For instance, the cookie jar component provides the options of whether or not to encrypt a cookie, sign a cookie, and what algorithms are used to do either of these tasks. The session component provides these same options for its identifier as well as options to regenerate the identifier and change its default name. The session data may also be optionally encrypted and stored in non-default locations. Lastly, the database component can be configured to use raw SQL statements or prepared statements. Each of these configuration options were again tested during the primary component tests and are configurable by setting these options in a main configuration file for the application.

5.1.3 Goal 3 Conclusions

The third goal of this project was to develop a web application framework that is evocative of current web application security concerns. This has been evidenced throughout the primary component tests that reference the categories of vulnerabilities that the various security mechanisms expose or mitigate from the OWASP Top 10.

The test results show which of the categories of vulnerabilities can be directly exposed and mitigated by configuring the primary components of the framework. The table below provides a mapping of these categories to the components that are affected.

Table 8. OWASP Top 10 Vulnerabilities Exposed and Mitigated

OWASP Top 10 Category	**Components Affected**
A1 – Injection	Database
A2 – Broken Authentication and Session Management	Cookie, Session
A3 – Cross-Site Scripting (XSS)	Cookie, Session
A4 – Insecure Direct Object References	---
A5 – Security Misconfiguration	Cookie, Session, Database
A6 – Sensitive Data Exposure	Cookie, Session, Database
A7 – Missing Function Level Access	---
A8 – Cross-Site Request Forgery	---
A9 – Using Components with Known Vulnerabilities	Cookie, Session, Database
A10 – Unvalidated Redirected and Forwards	---

Based on these results, the configuration of the primary components directly addresses six of the ten categories. The remaining four categories may be addressed by the business logic of the application.

A4 – Insecure Direct Object References refers to flaws that expose a reference to resources without any proper restrictions. The example attack provided by the OWASP Top 10 for this vulnerability is when an application uses unverified data to access the account information for another user. The solution to preventing this type of an attack would be to validate that the account information for the logged in user matches the account information being modified. This could be handled by using a combination of logic from the session manager and/or auxiliary validation component.

A7 – Missing Function Level Access refers to flaws that allow non-privileged users to access functions that should only be available to authorized users. The example attack provided by the OWASP Top 10 for this vulnerability is when an attacker accesses a URL that should only be available to authorized users. Again, this can be prevented using either the session manager or validation component to ensure that a user accessing a URL has the correct level of authorization to access the resource.

A8 – Cross-Site Request Forgery refers to flaws where an attacker tricks authenticated users into performing unintended actions through the use of forged HTTP requests. While the session manager component does contain methods for generating and verifying CSRF tokens it does not support the injection of these tokens into the HTML of a rendered web page as a hidden form field. It would be up to the application developer to individually add these hidden form fields to every form they use in their application.

A10 – Unvalidated Redirects and Forwards refers to flaws where an attacker takes advantage of a redirect or forward feature within a web application to send a victim to a malicious location or to access unauthorized resources. The solution to this category of flaws would be to not use redirects or forwards that allow for user parameters. If allowed, then the user input should be validated using the validation component.

Overall, based on this information the Porous framework can be seen as evocative of current web application security concerns through direct and indirect use of the primary and auxiliary components

5.1.4 Goal 4 Conclusions

The fourth goal of this project was to create a web application framework that is extensible. Proof that this goal was achieved can be seen in the architecture of the Porous framework's core. The dependency injection container allows components to be added and swapped into the framework by storing definitions that provide the construction details of each component. Each component, both primary and auxiliary, was added to the framework by writing these service provider definitions and registering them with the container. Additionally, the routing component, which is based on the HttpFoundation component and the HttpKernelInterface allows for community middleware to be added to the framework. Together, the dependency injection container and routing layer provide the extensibility that was desired to meet this goal.

5.2 Future Work

The Porous framework developed for this project was able to successfully meet each of the four goals described above. However, at this point the Porous framework is still in its infancy and additional work can and should be done to bring this project to its fullest potential. Over the course of developing this framework the following areas of future work are seen as parts of the framework that may be developed at a later time.

5.2.1 Further Evaluation

Perhaps the most important consideration when drawing conclusions from the results of the experiments conducted on this framework is that no implementation is ever going to be completely secure against all vulnerabilities. Additional testing should be done to check for vulnerabilities that were overlooked or missed during the completion of this project. It would be beneficial for other developers and security experts to audit the code of this framework in order to locate any of these vulnerabilities and provide any additional insights on how to prevent them.

5.2.2 Additional Components and Implementations

Another future extension to this project would be to construct additional components or implementations of existing components. This would allow for added customization for application developers who may be seeking particular functionalities for the web applications that they build. Suggested components to be added to the framework would be authentication and authorization libraries for managing users.

5.2.3 Intrusion Detection System

The next area of future work would be the possible integration of an intrusion detection system. An intrusion detection system could be of use when web applications built using the framework are used in a cyber challenge. This, combined with the event manager component, could provide application users or challenge moderators with a means of logging the exact exploitations that take place within the vulnerable components of the framework. Existing intrusion detection systems may be looked at as possible candidates for inclusion in later releases of the framework.

5.2.4 Command Line Interface

A command line interface to the framework would be a nicety for application developers by providing them with tools to generate keys, application templates, and content. Additionally, a command line interface could include the functions for database migrations and seeding, which would accelerate the process of some of this other content generation.

5.3 Conclusion

In conclusion, the Porous web application framework was successful in meeting its goals. It has the potential to be a significant contributor to the open-source community and cyber security communities by allowing them to develop realistic vulnerable web applications relatively simply and have these web applications be easily extensible to facilitate reuse. While this project was successful, it should be seen as just the beginning as the framework should continue to grow over time.

BIBLIOGRAPHY

[1] WhiteHat Security, "Website Security Statistics Report," WhiteHat Security, May 2013.

[2] Sensio Labs, "Symfony Framework," Sensio Labs, [Online]. Available: https://symfony.com/. [Accessed 18 June 2015].

[3] Laravel, "Laravel: The PHP Framework for Artisans," [Online]. Available: http://laravel.com/. [Accessed 18 June 2015].

[4] Ruby on Rails, "Ruby on Rails," [Online]. Available: http://rubyonrails.org/. [Accessed 18 June 2015].

[5] Django, "Django," Django, [Online]. Available: https://www.djangoproject.com/. [Accessed 18 June 2015].

[6] J. Lockheart, "Slim Framework," Slim Framework, [Online]. Available: http://www.slimframework.com/. [Accessed 18 June 2015].

[7] Sensio Labs, "Silex The PHP micro-framework based on the Symfony2 Components," [Online]. Available: http://silex.sensiolabs.org/. [Accessed 18 June 2015].

[8] Sinatra, "Sinatra," [Online]. Available: http://www.sinatrarb.com/. [Accessed 18 June 2015].

[9] "Armin Ronacher," [Online]. Available: http://flask.pocoo.org/. [Accessed 18 June 2015].

[10] University of Rhode Island, "Open Cyber Challenge Platform," [Online]. Available: https://opencyberchallenge.net/. [Accessed 18 June 2015].

[11] Open Web Application Security Project, "Open Web Application Security Project," [Online]. Available: https://www.owasp.org/index.php/Main_Page. [Accessed 18 June 2015].

[12] Open Web Application Security Project, "OWASP Site Generator," [Online]. Available: https://www.owasp.org/index.php/OWASP_SiteGenerator. [Accessed 18 June 2015].

[13] Open Web Applicaiton Security Project, "OWASP Top 10," [Online]. Available: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project. [Accessed 18 June 2015].

[14] Open Web Application Security Project, "OWASP Testing Project," [Online]. Available: https://www.owasp.org/index.php/OWASP_Testing_Project. [Accessed 18 June 2015].

[15] RandomStorm, "Damn Vulnerable Web Application," [Online]. Available: http://www.dvwa.co.uk/. [Accessed 18 June 2015].

[16] Github, "Github RandomStorm DVWA Issues," [Online]. Available: https://github.com/RandomStorm/DVWA/issues. [Accessed 18 June 2015].

[17] Q-Success, "W3Techs - World Wide Web Technology Surveys," [Online]. Available: http://w3techs.com/. [Accessed 18 June 2015].

[18] Alexa, "Alexa," Amazon, [Online]. Available: http://www.alexa.com/. [Accessed 15 June 2015].

[19] M. Fowler, "Inversion of Control Containers and the Dependency Injection pattern," [Online]. Available: http://www.martinfowler.com/articles/injection.html. [Accessed 18 June 2015].

[20] M. Seemann, "Service Locator is an Anti-Pattern," [Online]. Available: http://blog.ploeh.dk/2010/02/03/ServiceLocatorisanAnti-Pattern/. [Accessed 15 June 2015].

[21] R. C. Martin, "Design Principles and Design Patterns," [Online]. Available: http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf. [Accessed 18 June 2015].

[22] Illuminate, "Github Illuminate Container," [Online]. Available: https://github.com/illuminate/container. [Accessed 18 June 2015].

[23] "Lumen," [Online]. Available: http://lumen.laravel.com/. [Accessed 18 June 2015].

[24] Sensio Labs, "Pimple A Simple Dependency Injection Container," [Online]. Available: http://pimple.sensiolabs.org/. [Accessed 18 June 2015].

[25] The League of Extraordinary Packages, "Container," [Online]. Available: http://container.thephpleague.com/. [Accessed 18 June 2015].

[26] Network Working Group, "Hypertext Transfer Protocol -- HTTP/1.1," World Wide Web Consortium, [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616.html. [Accessed 15 Jun 2015].

[27] SensioLabs, "The HttpFoundation Component," SensioLabs, [Online]. Available: http://symfony.com/doc/current/components/http_foundation/introduction.html.

[Accessed 15 June 2015].

[28] Sensio Labs, "The HTTPKernel Component," [Online]. Available: http://symfony.com/doc/current/components/http_kernel/introduction.html. [Accessed 18 June 2015].

[29] Drupal, "Drupal," [Online]. Available: https://www.drupal.org/. [Accessed 18 June 2015].

[30] phpBB, "phpBB," [Online]. Available: https://www.phpbb.com/. [Accessed 18 June 2015].

[31] PHP Framework Interop Group, "PHP Framework Interop Group," [Online]. Available: http://www.php-fig.org/. [Accessed 15 June 2015].

[32] PHP Framework Interop Group, "HTTP Message Interfaces," [Online]. Available: http://www.php-fig.org/psr/psr-7/. [Accessed 15 June 2015].

[33] Igor Wiedler, "Stack," [Online]. Available: http://stackphp.com/. [Accessed 18 June 2015].

[34] The League of Extraordinary Packages, "Route," [Online]. Available: http://route.thephpleague.com/. [Accessed 18 June 2015].

[35] A. Ferrara, "N-Tier Architecture - An Introduction," [Online]. Available: http://blog.ircmaxell.com/2012/08/n-tier-architecture-introduction.html. [Accessed 18 June 2015].

[36] The League of Extraordinary Packages, [Online]. Available: http://event.thephpleague.com/2.0/. [Accessed 18 June 2015].

[37] Sensio Labs, "The EventDispatcher Component," [Online]. Available: http://symfony.com/doc/current/components/event_dispatcher/introduction.html. [Accessed 18 June 2015].

[38] Sabre, "Sabre EventEmitter," [Online]. Available: http://sabre.io/event/eventemitter/. [Accessed 18 June 2015].

[39] J. Boggiano, "Github Monolog," Github, [Online]. Available: https://github.com/Seldaek/monolog. [Accessed 18 June 2015].

[40] The PHP Group, "Mcrypt," [Online]. Available: http://php.net/manual/en/book.mcrypt.php. [Accessed 15 June 2015].

[41] J. Daemen and V. Rijmen, "AES Proposal: Rijndael," National Institute of Standards and Technology, [Online]. Available:

http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf. [Accessed 15 June 2015].

[42] "Announcing the Advanced Encryption Standard (AES)," National Institute of Standards and Technology, [Online]. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf. [Accessed 15 June 2015].

[43] B. Schneier, "Description of a New Variable-Length Key, 64-bit Block Cipher (Blowfish)," [Online]. Available: https://www.schneier.com/paper-blowfish-fse.html. [Accessed 15 June 2015].

[44] W. Penard and T. van Werkhoven, "On the Secure Hash Algorithm Family," Utrecht University, [Online]. Available: http://www.staff.science.uu.nl/~werkh108/docs/study/Y5_07_08/infocry/project/Cryp08.pdf. [Accessed 15 June 2015].

[45] H. Moody, "Respect Validation," Respect, [Online]. Available: http://respect.li/Validation/. [Accessed 18 June 2015].

[46] C. Cornutt, "Effective Validation with Respect," websec.io, [Online]. Available: http://websec.io/2013/04/01/Effective-Validation-with-Respect.html. [Accessed 15 June 2015].

[47] SensioLabs, "Twig," [Online]. Available: http://twig.sensiolabs.org/. [Accessed 15 June 2015].

[48] T. Otwell, "Github Illuminate View," [Online]. Available: https://github.com/illuminate/view. [Accessed 15 June 2015].

[49] J. Reinink, "Plates Native PHP Templates," The PHP League, [Online]. Available: http://platesphp.com/. [Accessed 15 June 2015].

[50] Open Web Application Security Project, "OWASP Zed Attack Proxy," [Online]. Available: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project. [Accessed 18 June 2015].

[51] "SQLMap," [Online]. Available: http://sqlmap.org/. [Accessed 18 June 2015].