A Secure Framework for Communication and Data Processing in Web Applications

Suprakash Sudarsanan Nair and Karuppasamy Mariappan

1 Department of Information Technology, Kalasalingam Academy of Research and Education, Anand Nagar, Krishnankoil 626126, Tamilnadu, India
2 Department of Computer Applications, Kalasalingam Academy of Research and Education, Anand Nagar, Krishnankoil 626126, Tamilnadu, India; karuppasamy.m1987@gmail.com

Abstract: Web applications are widely used, and the applications deployed on the web do not always satisfy all the security policies. This may arise due to less secure configurations, less knowledge in security configurations, or due to insecure coding practices. Even though a lot of practices are available, a lot of security loopholes are still available for hackers to steal information. A secure web application framework is discussed here which incorporates solutions to major security loopholes that attackers may use for stealing information or compromising systems. The security framework proposed here ensures an encrypted data transfer making the data safe and server-side vulnerability detection and avoidance for major attacks like SQL injection (SQLi) and Cross Site Scripting (XSS). The client side of the framework is responsible for validations, encryption, and session management through a JavaScript module. The server side of the framework is responsible for decryption and validation, data management, and URL management. The framework deployed with PHP showed a good outcome when tested with the Arachni web application security scanner. The framework will be further studied for performance with huge workloads. Further, the work will be extended to cover other attacks.

Keywords: SQL injection; Secure Sockets Layer; Advanced Encryption Standard; PHP; web application framework

1. Introduction

Nowadays, web applications are used in almost all fields like education, banking, finance, advertising, and many more. Since the usage of Internet applications has increased, the secure data that get communicated over the Internet have also increased. This has attracted a large set of hackers for financial benefit. The Internet has also become a place where a lot of web services and IoT devices communicate with each other with networking capability. Two main types of attacks mostly happen in web applications and services: injecting malicious code and logic vulnerabilities, as stated in [1]. Injection attacks happen by adding or injecting malicious code as a part of a query, script, or a command. This allows hackers to steal information or even change the behavior of the application. Logic attacks happen when there is a loophole that allows the session to be taken or information to be retrieved without proper authentication management, or when there is open data in the HTTP request that can affect the entire logic of the system. Even though there are a lot of measures taken for securing the web, most of the developers and deployer are taking things lightly. A lot of sites are even running without an SSL certificate. This helps hackers to easily steal information from web applications. There are a lot of contributions from different people in this. This work first discusses the major contributions towards securing...
web application, then the recent major attacks happening on web applications and the proposed secure web framework for application development.

2. Literature Review

Ref. [2] developed micro level dedicated web security services to handle vulnerabilities in the cloud platform. This work mainly focused on general data protection regulations that are being put forward by different countries. Getting proper agreement for data processing, privacy of collected data, and safe handling of information during communication are some of the goals of these services. These micro services extract the keywords from the request and identify the security level that needs to be applied to the services. Governance over the security is done based on the security requirements of the users.

Ref. [3] worked on XSS attacks on web applications. These are malicious codes that are injected into web applications. The study focused on XSS vulnerabilities in PHP source codes. This followed static and dynamic algorithms to identify XSS attacks. This work suggests an automated approach to use static as well as dynamic approaches like genetic, particle swarm, and ant colony algorithms to find the vulnerabilities in the code.

Ref. [4] proposed a method to evaluate the security and privacy in top cloud players. This considered major security parameters like confidentiality, availability, privacy, and accountability for evaluating security weightage. Weightage values were assigned from lowest to highest under seven categories. Based on the weightage values, overall security values were calculated and ranked. This uses service level agreements (SLA) for defining and identifying complex security requirements.

Ref. [5] created a benchmarking system for service providers to check and make informed decisions on the security loopholes in the system. This not only checks the security levels of the system, but also compares them with similar web services and provides a detailed view of the other factors that may affect the security of the system. This works in two phases; in the first phase, the framework checks whether the target service is well qualified for security deployment. In the second phase, the unsecure behavior of the service is analyzed, and the cause of such vulnerability is detected. The framework was tested with Denial-of-Service Attacks in different frameworks and found to be good.

Ref. [6] designed a prototype called Det-Logic for identifying logical loopholes in web applications. This has three phases. In the first phase, the specification is extracted; in the second phase, an attack vector is generated based on the information from the first phase and in the third phase, the responses are compared for any vulnerabilities and reporting is made accordingly. Three logical attacks are addressed here. The first is parameter manipulation, where the attacker changes some parameters of the website which may change the whole logic of the system. The second is access control, in which the attackers may access some resources due to a less robust authentication control mechanism. The third is workflow bypass, where a hacker can bypass some of the operations and move to the next operations inserting malicious code in between. Det-Logic design can identify logical vulnerabilities in the application.

Ref. [7] presented a security auditing approach for common injection attacks like XSS, SQL, XML, and XPath injection. The method extracts a small slice of code which is more appropriate to do security audits. The tool was developed for Java-based application, but it can be further applied to other languages.

Ref. [8] developed a framework that automatically verifies the security policies between web services communication. This creates a set of web service communication processes based on integration standards. These multi-party interactions are developed based on component models where integration and verifications are done at the early stages. This ensured an end-to-end secure information flow.

Ref. [9] developed a framework that can detect attacks on web servers based on the knowledge base and behavior of the attacker using an artificially intelligent system. This has a training phase where the system learns from the users about the possible attacks and stores this information in a knowledge base. Later, a comparison with the similarity
index of the knowledge base is performed, and access to the systems is provided. Similarly, ref. [10] developed a framework called AppMine, which also works on the knowledge base of the user’s requests. This uses a machine learning framework with different machine learning algorithms to train the knowledge base.

3. The Common Attacks on the Web Applications

Ref. [11] Open Web Application Security Project is a non-profit organization working on the security of applications and how well they can be secured. It publishes current threads, tools, and mechanisms to make applications secure. As per the recent reports of OWASP, injection, broken authentication, sensitive data exposure, XXE, broken access control, security misconfiguration, cross site scripting (XSS), insecure deserialization, using components with known vulnerabilities, and insufficient logging and monitoring are top in the list. This work mainly focuses on injection, broken authentication, Broken Access Control, and XSS. The other attacks will be studied in the future.

3.1. Thread Modelling

Application Overview:

The application is intended to provide secure data communication and processing from the client end to the server end. It is developed for maintaining sensitive information at an institutional level. The business logic and data access logic lie on the server end. Users can use the service through any web browser. The actors in the application are regular users consuming the services and administrators who access the services through the web platform.

The entry point to the application is restricted through 443 (SSL). All other ports are disabled for security reasons. All access is done through valid user credentials, which are validated from client and server side. The exit point involves killing the sessions and cookies through logout or a timer based on inactivity.

Threads:

This framework takes care of the following attacks:

- SQL injection attacks—exposure to executing commands in the database, thereby getting access or modifying or deleting data;
- Broken authentication attacks—attackers may gain access by compromising user-name/password, session keys, or user identity;
- Broken access control—allows unauthorized users to access restricted file contents or database content;
- Cross site scripting—injects script codes in the system, allowing the attacker to track or gain access to the entire system.

3.2. Injection Attacks

This is a kind of attack where the attacker can inject a piece of code that may change the usual behavior of the application or even steal information from the application. The most common of these is the SQL injection attack, where injecting a piece of SQL code into the inputs of the application can break the SQL server and fetch some information. One example of such an attack is the following:

```sql
select * from user_details where user_id='some user' and password='password'
```

An input to the application’s username “’ or 1=1;” may break this SQL query by taking all records from the table due to the 1=1 condition in the input field, as per the query below:

```sql
select * from user_details where user_id=’’ or 1=1;
```

This usually happens when data is provided in the SQL statement itself from the applications. The most common solution to this is separating data from the application logic and putting forward restrictions on data. SQL injection attacks are prevented by eliminating the character that can break the PHP code/SQL code. This is an additional
task along with common techniques, by using prepared statements, constructed standard procedures, and input validations.

3.3. Broken Authentication

This happens when an attacker gains control over the system partially or fully. These are very common in web applications due to improper session management or credentials in the applications. Usually, the session ID is available in a cookie file in the browser, or can be attached to the URL. In both cases, if a session is left open and if someone gains access to the system, they can hijack the session and use it.

A case study based on this is session ID hijacking. In this case, the hackers may steal the user’s valid session ID through the URL/browser when the user is inactive. The hacker may then copy the session ID and use it as if they were the same user as the original one.

3.4. Broken Access Control

There must be a proper access control mechanism to access any resource from the server. There are many attacks happening nowadays due to broken access control. Some of these provide access to documents even if the session is logged out. This happens due to improper session management. Some resources may be available without any authentication mechanism. This helps the hackers to access the data in the system or even sometimes inject malicious code that can change the behavior of the system.

One example of this attack is when a hacker passes some additional parameters to the URL or HTTP parameters that allow the user to access the restricted content on the page with restricted user login. They may even get access to a file system.

3.5. Cross Site Scripting (XSS)

This is also a kind of injection attack where the attacker adds scripts to the browser. When this information is sent to the server, this can steal the server information or provides hacker fed information for processing. This can even do more on the client side, where it can take any information from the DOM or from the session variables.

Ref. [12], a web application security company, provides a brief insight of the current web security issues in the Spring 2021 web vulnerability report. They conducted a survey over the world with different sets of organizations from January 2020 to December 2020. The findings of the research conclude that there are 27% high-severity vulnerabilities, 63% have medium-severity vulnerabilities, 25% are XSS vulnerable, and 26% have WordPress vulnerabilities. Also compared with the 2019 statistics, there is an increasing trend found in Remote Code Execution (RCE) and Vulnerable JavaScript libraries.

XSS cheat sheets are used for cross-site scripting attacks. A filter based on cheat sheets, which contains the most common attack scripts is used. These scripts in the code will be stopped by the security framework designed in the preprocessing stage.

4. The Proposed Framework for Web Security

This section describes the methods adopted in the framework deployed in PHP for overcoming the above-mentioned attacks. This uses JavaScript and PHP as its development environment. This logic can be extended to other languages too.

4.1. JS Module

This module takes care of data security between the client and the server. Even though there are mechanisms like SSL, there are possibilities that the SSL can be spoofed. This is not a kind of attack, but this is a man in the middle who can retrieve the information we communicate with the help of spoofing tools like SSLStrip, CarbonCopy, or similar tools. Since there are a lot of open SSL options available, attacks on this are also common. So, to secure the data, the data when sent from the client itself are encrypted by the module introduced. Further, to kill the unclosed session data after a period, this invokes a method to kill sessions and cookies by monitoring idle activity. The validation control helps in
checking the input data and basic malicious codes. Figure 1 represents the proposed framework for secure web applications. This describes the process involved in JS and PHP modules.

4.2. PHP Module

The major issues with web applications are processing SQLi attacks. To overcome this issue, all the inputs are sanitized with predefined sets of possible knowledge bases. Further, this uses a PDO model to take care of these attacks. The PDO by itself cannot provide full security. So, all the data that need to be processed are used under utf8 encoding format alone as a first step, the special characters that can break the code are blacklisted from execution. This can possibly reduce SQLi attacks.

This framework blocks all inputs from clients other than the intended data fields. Further, XSS cheat sheets [13] are implemented with the framework to block XSS attacks.

The session data that is maintained in the system is not always static data. This works with session information generated with a unique random number, which serves as the session information of the user. So, session hijacking is not possible. This framework also checks for spoofing attacks and authentication on each resource request.

As per recent studies, Remote Code Execution (RCE) is one of the top attacks that happen in web applications. Through RCE, a hacker can gain access to a system to run random code. Usually, a firewall in the system is configured to block incoming connections, allowing specific ports like 80 and 443. But the outbound connections are usually open in the systems. Taking this as an advantage, through RCE, attackers use reverse shell to run programs in the target machine. Once a RCE is done, a reverse shell, which listens to a port will be added to the target machine. A reverse shell is a shell session established on a compromised connection that is initiated from a remote computer or server and not from the native host. Attackers with a successful connection exploit a remote command execution, and use a reverse shell to get access to interactive shell session on the target machine and continue their attack. A reverse shell also can be the sole mechanism to gain remote shell access across a NAT or firewall. If a web application like PHP passes a parameter sent to the server through a GET request to the PHP include() function without any kind of validation, then the attacker can execute some miscellaneous code to get access of the system or data of the system.

One example of such an attack uses the following:

```
ncat -l -p port_no
```

which listens to the outbound port number specified in a Linux system. Then, the attacker can communicate through this port and do remote command executions.

Some of the solutions to the above-said issues can be done as follows.
Usually, websites communicate through HTTP which is purely clear text format which can be read easily by others. To keep it safe from others, it is necessary to add SSL or TLS to the application. As an additional layer of security, HTTP Strict Transport Security (HSTS) can be added which always forces us to use HTTPS in the browser.

Further, it is a good practice to use only the same source for all web content. For example, a header with header (‘Access-Control-Allow-Origin: *’) will allow all links to communicate on the website, which opens the possibility for a click-jacking attack, providing a way for hackers to inject a document or a script loaded to the site which can interact with a resource from another origin. To overcome this, same origin policy must be enforced and wherever external sites are used, that must be added to the access control. The possible blocks where such attacks may happen are a java script with <script src="..."></script>, a CSS script added with <link rel="stylesheet" href="...">, displaying images with <img> sources, playing media files through <video> and <audio> tags, and any external objects embedded with <object> or <embed>.

Another problem in web applications is the referrer policy. Once a referrer is available for a web application, there are possibilities to send data to the referrer. For example, if our application displays an advertisement on a page run by a malicious user, if we get into the site through the malicious site, the malicious site becomes the referrer. If, at any point, we use some sensitive information on the site, there are chances that this may be retrieved by the malicious site. This can be prevented by restricting the amount of information that flows through the referrer by setting a proper referrer policy.

The policies followed for the framework is listed in Listing 1.

Listing 1. Policy enforced under the framework.

Enable SSL/TLS for the site
Ensure HTTPS access alone
Add HSTS for additional enforcement of security layer
Allow content from same domain alone by ensuring
  Access-Control-Allow-Origin: https://mydomain
Allow only the required methods on the site. Most recommended with POST
  Access-Control-Allow-Methods: POST
Configure the referrer policy
  Referrer-Policy: no-referrer
Configure the embedded objects and iframes to access same domain/deny access
  X-Frame-Options: DENY (or) SAMEORIGIN
Configure
  directory browsing restrictions,
  proper redirects for HTTP error pages
Configure URL Management
Data Sanitization

5. Results and Discussion

The basic setup for web security systems was analyzed by setting up this environment on specific sites with and without SSL encryption; the data was tested with different malicious codes from different sets of users. All the policies enforced in the application were working fine with the expected outcome.

Data from the sites were found to be encrypted, since we used standard encryption between the client and server through AES. The sites were found to automatically delete the session information and cookie information after the set interval of time. SQL and XSS injections were tried on the sites and we found that they were discarded by the system.

The Arachni web application security scanner [12] was used to check web application vulnerabilities based on the framework. Arachni is one of the best open source technologies to check almost all kinds of web application attacks. Refs. [14–16] have conducted studies on different web application security scanners, considering different attacks and parameters;
from these studies, it was found that Arachni is the one of the best tools that scans almost all kinds of security audits and provides accurate results.

The architecture of the web servers was made with the techniques deployed in [17,18]. This architecture provides good hardware utilization with its best performance for the users considering power optimization.

Figure 2 shows the issues by type, trust, and severity on partial implementation of the designed framework. The blue bar in Figure 2 represents the trusted connections, the light red bar represents the untrusted connections, and the orange line represents the severity in connections that get attacked. This shows that the severity level of attacks is very minimal in the platform.

![Figure 2](https://example.com/fig2)

**Figure 2.** Issues by type, trust, and severity on partial implementation of framework.

Figures 3 and 4 represent the severity possibilities and issues by type, respectively. The test machine was set up without SSL, resulting in a strict transport security header issue; further, the X-Frame options header was not initially enabled. The HTTP trace allowed for XSS attacks. In Figures 3 and 4, the informational issues occurred since the server responded with 200 (OK) and 404 (Not Found) status codes. Even though this is not a severe issue, this will give an insight of the web application. The non initialized X-Frame header resulted in Low security issues. The non initialized SSL resulted in medium scale security issues. By further refining the framework, most of the loopholes were overcome.

![Figure 3](https://example.com/fig3)

**Figure 3.** Severities based on possible impact.
Figure 4. Issues by type.

Figure 5 represents the further refinement of the framework. Informational security, SSL and header frames were added to this refinement. The results show that the risk level is very low compared to Figure 2. The red bar in Figure 5 disappeared since risk is minimal and the orange line indicating severity became very low. SSL addition made the application trustworthy.

As seen in the Table 1, all the tests done on the machine passed the vulnerability tests and were found to be a good framework for web application development which has high security concerns. The system is being further studied for other attacks on the web platform. The tick mark in the Table 1 shows that the test passed for the respective attack parameters at different weightage of users.
Table 1. Security pass test for the defined attacks.

| Users | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
|-------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Data Protection | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Broken Authentication | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Broken Access Control | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| SQL Injection | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| XSS | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

✓—represents the test passed for the attack parameter.

6. Conclusions

Nowadays, securing the web is very important since all kinds of communications and transactions are happening through this web platform. Even though individual solutions like [19–21] are available for providing security in web applications, many application developers take this lightly. This work proposed a frame for secure application development with PHP where SQL injection attacks, broken authentication, broken access control, and cross site scripting were studied, and safety mechanisms were implemented.

The framework will be further extended for the other possible web application attacks in the future. Also, this framework will be extended further to other scripting languages. In the future, this model will be documented for all developers, even for non-PHP developers; the scalability and performance on a large scale will be measured since there is a little preprocessing required for framework implementation.

Author Contributions: Both the authors S.S.N. and K.M. contributed equally to the work. Conceptualization—S.S.N. and K.M., Methodology—S.S.N. and K.M., Software—S.S.N. and K.M., Validation—S.S.N. and K.M., and Drafting—S.S.N. and K.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research did not receive any funding.


Informed Consent Statement: Not Applicable.

Data Availability Statement: Data analysis was done on real time inferences of the test machines. If further insight is required, the data can be accessed by sending an email to s.suprakash@gmail.com or karuppasamy.m1987@gmail.com.

Acknowledgments: Thanks to the Kalasalingam Academy of research and Education for providing support and facilities for completing the entire research work.

Conflicts of Interest: The authors declare no conflict of interest.

References

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.