Web Application Top 10 OWASP Attacks and Defence Mechanism

Madhuri N. Gedam
Research Scholar
Deptt. of Computer Engineering
Veermata Jijabai Technological Institute (VJTI),
Mumbai, India

Bandu B. Meshram
Professor
Deptt. of Computer Engineering
Veermata Jijabai Technological Institute (VJTI),
Mumbai, India

ABSTRACT
Enterprise Security API (ESAPI) is a security framework developed by the Open Web Application Security Project (OWASP) to help developers to build secure applications. ESAPI can generally help in securing web applications against various types of vulnerabilities. By incorporating ESAPI into web application development, developers can leverage its secure coding practices, libraries, and APIs to address various vulnerabilities that may be part of the OWASP Top 10 2023 attacks. The research explores the development of web based application with vulnerabilities and then OWASP Top 10 Attacks are made on it. The same web application is recoded by embedding ESAPI and the Top 10 attacks are made on this application. It is found that due to security provided into the web applications, attacks can not be made on the web application. However, it's important to stay updated with the latest security guidelines and recommendations from OWASP to ensure maximum protection against emerging threats.

Keywords
Software Development Life Cycle, OWASP Enterprise Security API, SQL injection, Cross-Site Scripting.

1. INTRODUCTION
Web application security is a major concern in today's digital era. Web applications are a popular target for attackers. It necessitates protection of a website from intrusion to avoid loss of business data and reputation of an organization. The non-profit organization Open Web Application Security Project (OWASP) Top 10 coverage implementation is necessary to fight against various kinds of attacks [2]. It has developed Enterprise Security API (ESAPI) framework which contains a set of security controls and utilities designed to help developers protect web applications from common vulnerabilities. The framework benefits from the collective knowledge and experience of the OWASP community, ensuring that it incorporates best practices and undergoes rigorous testing and scrutiny. It takes care of mitigation of common vulnerabilities, integration with other security tools and it is platform as well as language independent framework.

The OWASP ESAPI framework provides a set of security controls, functions, and guidelines that help to protect the application against various vulnerabilities identified by OWASP. By integrating and utilizing ESAPI within the application's architecture, developers can implement measures to mitigate these vulnerabilities and enhance the overall security of the application. ESAPI is designed to work well with other security tools and libraries. It can be integrated with vulnerability scanners, static analysis tools, and security frameworks, enhancing the overall security posture of the application and enabling a more holistic security approach. ESAPI addresses common web application security vulnerabilities such as SQL Injection (SQLI), Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Session Hijacking and many more. By utilizing the framework's features and controls, developers can significantly reduce the risk of these vulnerabilities and strengthen the security posture of their applications.

The research work involves designing a web application without web security in the first phase. Then it will be exposed to ethical hacking of the web site by attacking a system. Next work will be to protect the proposed system using ESAPI framework. Finally, security testing for all Top 10 attacks is carried out to check for secure application.

The paper is organized as follows. Section 2 describes detailed literature survey of ESAPI framework, OWASP Top Ten attacks. Section 3 presents proposed framework for securing web application. Section 4 concludes the paper and gives direction to the future work.

2. LITERATURE SURVEY
The detailed literature survey on OWASP Enterprise Security API (ESAPI) framework has been carried out and elaborated below.

2.1 ESAPI
The ESAPI framework is a set of APIs containing security controls and utilities to protect web applications from common vulnerabilities. Additionally, it will limit the use of new classes of vulnerabilities and stop zero day attacks that use known exploits [14].

2.2 Software Architecture for Application Design Using ESAPI
ESAPI's integration into application design is demonstrated in the Figure 1.

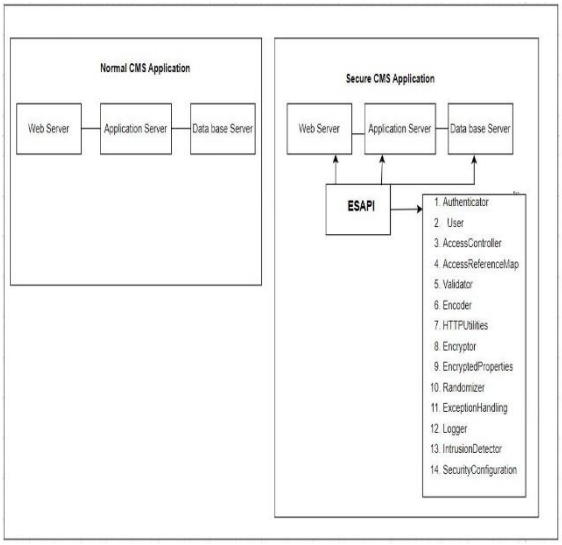

Fig 1: Application Design with ESAPI

The community support provided by OWASP and ESAPI is quite beneficial. In addition to Java Logging, JCE, and Adobe Commons FileUpload, the ESAPI class library also builds on other superior security libraries. It takes concepts from other security packages, including ACEGI, Apache Commons Validator, Microsoft's AntiXSS library, and many more. For enterprise developers, this library offers a single, easy interface to security functions [14].

The ESAPI provides various interfaces to be used for security of the web application as listed in the Table 1.

Table 1. ESAPI Interfaces

Sr. No	Interface names	Purpose
1	Authenticator control	Authentication
2	AccessController	Role based access control
3	HTTPUtilities	To provide HTTP specific handler
4	Encoder	HTML/XML encoding
5	Encryptor	Encryption of the data
6	Executor	Protection of OS commands
7	IntrusionDetector	Detection of security bypass activities
8	Randomizer	To generate random numbers or strings using cryptography.
9	Validator	Data validation in the application

2.3 OWASP Top Ten Coverage

ESAPI has sufficient features to offer security against the majority of the OWASP Top Ten attacks when used appropriately. The insecure communications category is typically not under the developer's control and it is the only significant exception [15]. The mapping of various ESAPI interfaces with OWASP Top Ten attacks is done as shown in Figure 2.

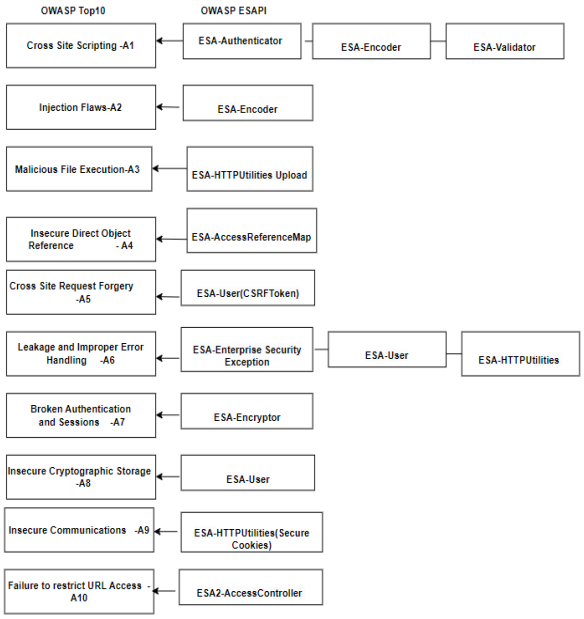

Fig 2: ESAPI mapping with Top Ten vulnerabilities

ESAPI Security control interfaces addresses Top Ten vulnerability as detailed below.

1. **AccessController**

public interface AccessController

This interface is a collection of methods and used to provide centralized access control in various application layers. The roles or permissions assigned to current User object (from Authenticator.getCurrentUser()) will be determined by this object. The assertAuthorizedForFunction() rule is recommended to be implemented using hasPrivilege() or isUserInRole() methods. Access control uses assertAuthorized() to make it simple to use and verify.

Here is a code of ESAPI implementation for access control check.

```
try
{
ESAPI.accessController().assertAuthorized("businessFunction
", runtimeData);
    // execution of the business function
 }
catch (AccessControlException ace)
{
 //  attack performed
}
```

The rendering of particular controls in the user interface layer can be controlled using access control checks. Those are not considered as attacks because they are expected to fail if an unauthorized user signs in. Both the data and business logic layers must implement access control checks [21].

```
<%if(ESAPI.accessController().isAuthorized(
"businessFunction", runtimeData ) ) { %>
<a href="/doAdminFunction">ADMIN</a>
```

<%} else {%>

NORMAL

<% } %>

Methods

(a) isAuthorized boolean isAuthorized(Object key, Object runtimeParameter)	It is used to run the AccessControlRule, which is listed in the resources/ESAPI-AccessControlPolicy.xml file and is identified by key. It also controls execution flow. Input: Key: key maps to <AccessControlPolicy><AccessControlRules><AccessControlRule name="key"
(b) isAuthorizedForData boolean isAuthorizedForData(String action, Object data)	It is used to determine if the current user has permission to view the data that is referenced and is represented as an Object. Input: action – It is used to determine whether an access control option like a role, or an action like Read, Write, etc. is being performed on the object. data - The actual object being accessed, an object identifier, or a reference to the object being accessed. Output: if the data has been allowed, it is true

2. AccessReferenceMap

public interface AccessReferenceMap<K>

extends Serializable

The AccessReferenceMap interface maps a set of internal direct object references to a set of secure indirect references that can be exposed to the public. Filenames, database keys, and other direct object references need to be protected. Developers should generally avoid exposing their direct object references to prevent attacks. Indirect references are handled as strings to make it easier to utilize in HTML. Indirect references may be produced by implementations that use simple integers or more complex random character strings. Implementations would most likely provide a constructor that accepts a set of direct references.

It becomes difficult for an intruder to guess random strings as indirect object references, compared to simple integers. This interface will be helpful to prevent Cross-Site Request Forgery (CSRF) attacks [11].

Set fileSet = new HashSet();

fileSet.addAll(...); // add direct references (e.g. File objects)

AccessReferenceMap map = new AccessReferenceMap(fileSet);

// store the map somewhere safe - like the session!

String indRef = map.getIndirectReference(file1);

String href = "http://www.aspectsecurity.com/esapi?file=" + indRef);

...

// if the indirect reference doesn't exist, it's likely an attack

// getDirectReference throws an AccessControlException

// you should handle as appropriate

String indref = request.getParameter("file");

File file = (File)map.getDirectReference(indref);

Methods

(a) getIndirectReference <T> K getIndirectReference(T directReference)	It is used to fetch a safe indirect reference to critical direct object reference. This call should be used by developers for creating URLs, form fields, hidden fields, etc. Input: directReference - direct reference Output: indirect reference

3. **Authenticator**

public interface Authenticator

The Authenticator interface specifies different methods for producing and maintaining account credentials and session IDs in order to safeguard credentials. A thread local variable is used to store current user's identity. The application calls setCurrentUser() whenever HTTP request is received. The value of getCurrentUser() is used at many places in this API. Hence user object need not required to be passed to methods throughout the library. It simplifies authentication process with the current request and parameter names like username and password. The password needs to be verified before creating a session and setting the current user [12].

public void doPost(ServletRequest request, ServletResponse response) {

try

{

User user = ESAPI.authenticator().login(request, response);

// continue with authenticated user

}

catch (AuthenticationException e)

{

// handle failed authentication (it's already been logged)

}

Methods

(a) verifyPassword

boolean verifyPassword(User user, String password)

It is essential to store passwords in a hash format for security purpose. The password verification is done through this method. When performing the most sensitive operations, such as transactions, altering an email address, and modifying other account information, "reauthentication" is sometimes required.

This class's hashPassword(password, accountName) method is used for this purpose.

Input:

user - the user who needs to be authenticated, password - the hashed password of user-supplied value

Output:

It returns true in case the password is correct for the specified user

(b) createUser

User createUser(String accountName, String password1, String password2)

throws AuthenticationException

This method is used to create a new user with supplied information variables in the proper format and strength with verifyAccountNameStrength(String), verifyPasswordStrength(String, String). The entry of password is required to be done twice through user interface and need to be checked for similarity.

Input:

accountName - the account name of the new user

password1 - the first time entered password in the user creation form.

password2 - the second time entered password of the new user in the user creation form to check for similarity.

Output:

New User gets created

4. **Encoder**

public interface Encoder

This interface is used to perform input decoding and output encoding using various methods. Double encoding needs to be avoided by using canonicalization to prevent encoded attacks. A "whitelist" or "positive" security model must be applied to every method.

Encoding and decoding through this interface is done through the functions that depend on a group of codecs contained in the org.owasp.esapi.codecs package [13].

Methods

a) encodeForHTML String encodeForHTML (String input) Utilise HTML entity encoding to encode data for usage in HTML.	Input: the text to encode for HTML Output: input encoded for HTML
b) encodeForSQL String encodeForSQL(Codec codec, String input) For use in a SQL query, encode input. The recommended method is to utilize the PreparedStatement interface.	codec - a Codec that declares name of the database being used like MySQL, Oracle, etc. Input: text to be encoded for SQL Output: encoded output for use in SQL
(c) encodeForURL String encodeForURL(String input) throws EncodingException This method is used to perform URL encoding of the full string.	Input: input - the text to be encoded for use in a URL Output: input encoded for use in a URL Throws: EncodingException – on failure of encoding

5. **Encryptor**

public interface Encryptor

Common hashing, random number generation, and encryption operations can be carried out using a variety of methods supported by the Encryptor interface. An efficient cryptographic implementation, such JCE or BouncyCastle, should be used in implementations. ESAPI.Encryptor, a property in ESAPI.properties is the main property determining which implementation class is used. These properties are allowed to select the encryption algorithms, the preferred JCE provider, etc. Encryptor.MasterKey and Encryptor.MasterSalt must be set before using ESAPI encryption and 'setMasterKey.sh' help to set these two properties[15].

Methods

(a) Hash String hash(String plaintext, String salt) throws EncryptionException	Input: plaintext – the string to encrypt in a plain text format. salt - the salt to be added in the string in a plain text format before hashing Output: The plaintext is saved as the encrypted hash in a string format. Throws: EncryptionException whenever desired hash algorithm is not available or some other problem is faced in hashing.
(b) Encrypt CipherTextencrypt(PlainText plaintext) throws EncryptionException	It applies the cypher transformation defined by the property Encryptor on the provided plaintext bytes to encrypt them.
(c) Decrypt PlainText decrypt(CipherText ciphertext) throws EncryptionException	Input: The cipher text input is supplied Output: the plain text is provided as the output by decrypting provided cipher text. Throws:EncryptionException

6. **Executor**

public interface Executor

With very little risk, an OS command can be run via this interface. Implementations should consider precautions to

minimise the chance of injection into the command or the parameters. Specific time-out period should be implemented for the prevention of DoS attacks [16].

Methods

(a) executeSystemCommand ExecuteResult executeSystemCommand(File executable, List params) throws ExecutorException	Input: executable - command to be executed params - the parameters to be passed to the command for execution Throws: ExecutorException

7. **HTTPUtilities**

public interface HTTPUtilities

Additional security for HTTP requests, responses, sessions, cookies, headers, and logging is provided through the methods in this interface [17].

Methods

(a) assertSecureRequest void assertSecureRequest() throws AccessControlException	Calls assertSecureRequest with the current request.
(b) encryptHiddenField String encryptHiddenField(String value) throws EncryptionException Encrypts a hidden field value for use in HTML.	Input: value - the plain text value of the hidden field Output: encrypted value of the hidden field

8. **IntrusionDetector**

public interface IntrusionDetector

It is used to monitor security-related events and detect attack patterns supporting both custom events and exceptions [18].

Methods

(a) addEvent Void addEvent(String eventName, String logMessage) throws IntrusionException	Input: eventName - name of the event to add logMessage - logging the message with the event Throws: IntrusionException - the intrusion exception

The event should be logged to IntrusionDetector for logging purposes. The event should be saved at some place to check the security exception threshold limit. The User object is the suggested place to store the security event for the current user. The required security action can be executed and logged if the User successfully meets a security threshold.

9. **Logger**

public interface Logger

The security related events can be logged with this interface. According to the severity of the events, it allows for different levels of logging, like fatal being the highest value, error, warning, info, debug, and trace being the lowest value. Implementors should make use of a well-known logging library because building a high-performance logger is quite difficult. The always() method logs a message regardless of the log level on each iteration.

Methods

(a) setLevel void setLevel(int level)	Input: level - to define the log level.
(b) Always void always(Logger.EventType type, String message) Log an event regardless of what logging level is enabled.	Input: type - the type of the event message - the message to log

10. **Randomizer**

public interface Randomizer

This interface is used to generate cryptographically random numbers and string. Weak sources of randomization can compromise the effectiveness of numerous security solutions.The JCE or BouncyCastle are two examples of robust cryptographic implementations that implementers should employ. In ESAPI.properties, the specific algorithm utilized can be customized [20].

Methods

(a) getRandomString String getRandomString (int length, char[] characterSet)	Returns: the random string

A cryptographically strong pseudo-random number generation is provided by java.security SecureRandom.

Input:

length - the length of the string

characterSet - the set of characters to include in the created random string

Output:

random string

11. **SecurityConfiguration**

public interface SecurityConfiguration

extends EsapiPropertyLoader

All configuration data necessary to regulate how the ESAPI implementation operates is contained in the SecurityConfiguration interface. Use the operating system's access restrictions to limit access to the location or locations where the configuration data is stored [21].

Methods

(a) getApplicationName @Deprecated String getApplicationName()	Output: the name of the current application

12. **User**

A user account can be created, disabled, expired, or unlocked using this interface. Administrators enable the account, which can be locked due to failed logins or expire after a specified date. Enabled, unlocked, and unlocked accounts pass authentication. For authentication to succeed, the User must be unlocked, enabled, and not expired. [22].

Methods

(a) getLastHostAddress String getLastHostAddress()	Output: to get last host address of the user
(b) incrementFailedLoginCount void incrementFailedLoginCount()	Output: Increment failed login count.
(c) isSessionAbsoluteTimeout boolean isSessionAbsoluteTimeout()	Output: true, on exceeding session time out of the user

13. **Validator**

public interface Validator

This interface contains many methods to validate untrusted input and gives boolean output. Implementation should use a "whitelist" validation technique that matches specific patterns to stop bypasses through encoding or other means [23].

Methods

boolean validateInput(String context, String userInput, String inputType, int maxInputLength, boolean allowEmpty)

throws IntrusionException

(a) isValidInput	boolean isValidInput(String context, String input, String type, int maxLength, boolean allowNull) throws IntrusionException Calls isValidInput and returns true if no exceptions are thrown.
(b) isValidCreditCard	boolean isValidCreditCard(String context, String input, boolean allowNull) throws IntrusionException Calls getValidCreditCard and returns true if no exceptions are thrown.

3. PROPOSED FRAMEWORK FOR SECURING WEB APPLICATION

The proposed work on web security involves two main parts (i) Web application development without security and attacks on it (ii) Development of same web applications using Enterprise Security API. The OWASP Top 10 attacks are made on secure applications and found that OWASP Top 10 attacks can not be made on the secure web application. The aim of the proposed work is to implement secure web applications using ESAPI. The objectives of the work are as follows

1. To implement the web application using web technology without web security.
2. To do OWASP Top 10 attack on web Application.
3. To provide the security to the web application using OWASP ESAPI framework
4. To test the secure web application by performing OWASP Top 10 attack.

3.1 Secure Environmental Settings

The web application is made available in the public domain. The network defenses like firewall, intrusion prevention system is used as shown in Figure 3.

Fig 3: Firewall and IPS

Zero-day vulnerabilities are those that are discovered in the proprietary code of specific web applications but are not yet known to security defense systems. These vulnerabilities are easily exploitable by an experienced attacker.

Hardware Requirements: The minimum hardware specifications which can be used for implementing the system are: Processor: Intel Core i7, RAM: 16 GB, Hard disk: 1 TB

Software Requirements: Operating System: Windows 10 (64 bit), Database: MySQL, Technologies Used: Java (JDK 1.7), HTML, JSP and Java Servlet for web application.

Other Tools: Apache Tomcat, Net Beans IDE, Any Web Browser (Google Chrome/MozillaFirefox, etc.), Enterprise Security API.

ESAPI: It is a free, open source framework containing security control libraries helping to write lower-risk existing as well as new applications.

3.2 Proposed Defense Mechanism Implementation

In this section, authors have done experimentation for attacks and proposed the security mechanism to show how security

shall be implemented against these attacks on the web application with illustrative screenshots.

3.2.1 A1- SQL INJECTION

In SQL injection attack, a SQL query is injected into the program through the client's input field [3]. This attack is able to extract, modify sensitive data from the database, execute catastrophic operations such as shutdown the database, recover the content of the database file system and can execute commands to the operating system.

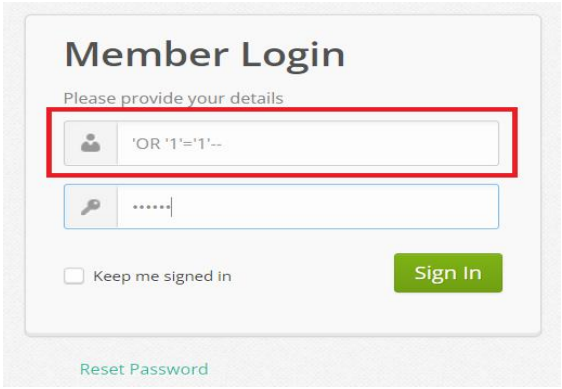

Fig 4a: SQL Injection Attack

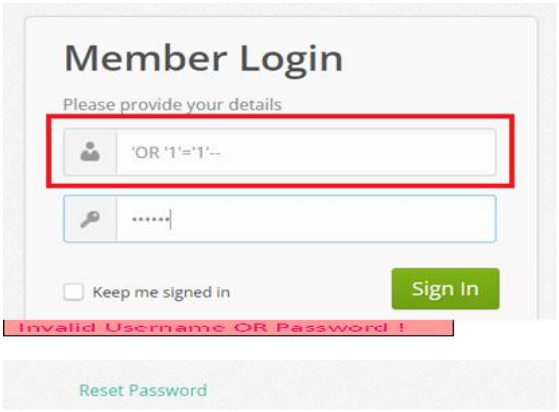

Fig 4b: SQL Injection Prevention

As shown in Figure 4a, the vulnerable input field is used as an entry point to exploit using WHERE condition which will always return true and expose the critical data [15]. The proposed SQL injection prevention implementation is as shown below:

```
Codec ORACLE_CODEC = new OracleCodec();

String query="select name from users where id="+
ESAPI.encoder().encodeForSQL(ORACLE_CODEC,
validateUserId)) and date_created>='"+
ESAPI.encoder().encodeForSQL(ORACLE_CODEC,
validatedStartDate)+"'";
```

For prevention of SQL Injection (Figure 4b), ESAPI's encoder for SQL can be used. It will transform any injection query to a statement which is safe to be executed on SQL database preventing SQL injection attack. With the use of ESAPI encoder for SQL, the Injection attack was repelled by the application.

3.2.2 A2- Cross-Site Scripting (XSS) Attack

It occurs when a malicious script is sent or executed from the victim's browser without proper validation in web applications [24]. The result of XSS attack is the user may be sent to malicious websites, user sessions may be stolen, or websites may be defaced.

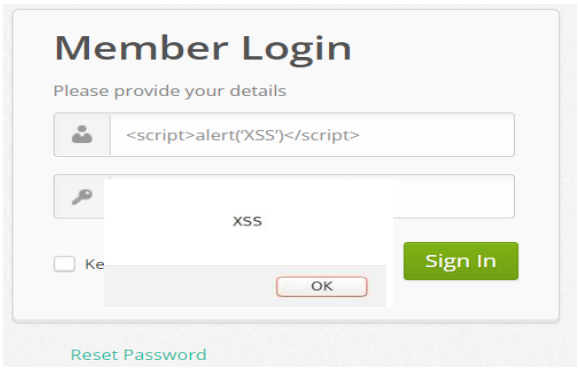

Fig 5a: Cross-Site Scripting Attack

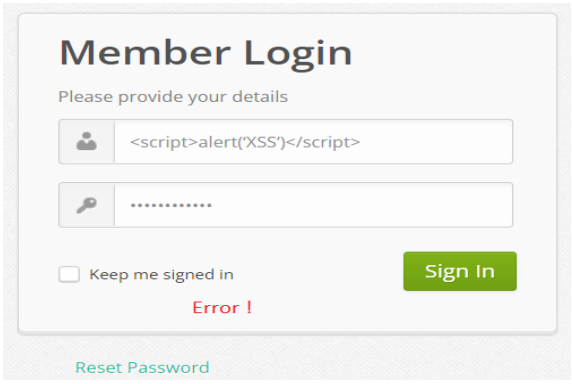

Fig 5b: Cross-Site Scripting Prevention

The javascript used to reveal the cookie of an user is as shown in Figure 5b.

Cross-Site Scripting Prevention
<script>alert(document.cookie)</script>

XSS attack can be prevented by using ESAPI's validator. The validator can be used to validate any user input such as user name, email_id , SSN, etc.

```
Validator.Email=^[A-Za-z0-9._%-]+@[A-Za-z0-9.-]+\\.[a-
zA-Z]{2,4}$1
Validator.IPAddress=^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-
9]?)\\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$
Validator.URL=^(ht|f)tp(s?)\\:\\/\\/[0-9a-zA-Z]([-.\\w]*[0-9a-
zA-Z])*(:(0-9)*)*(\\/?)([a-zA-Z0-9\\-
\\.\\?\\,\\:\\'\\/\\\\\\+=&amp;%\\&amp;language\\$#_]*)?$
```

If the properties available in validator.properties aren't sufficient for the application, then developers can define their own validators by writing custom registry expressions. The XSS attack is prevented using ESAPI's validator.

3.2.3 A3- Broken Authentication and Session Management

Due to wrong implementation of authentication and session management, the web application is prone to allow attackers to compromise user session tokens and user passwords [6].

This is an example of a vulnerable code which leads to broken authentication. The user is not properly authenticated before log in. This kind of programming is unfortunately surprisingly common in systems nowadays.

Fig 6a: Session Attack

Fig 6b: Broken Authentication Prevention

Session Attack is an example of bad session management as shown in Figure 6a. As shown in Figure 6b, the session id is included in the URL. If a user shares this URL with someone, then their session id will also be shared and other users will get this session id and other details such as credit card details.

session.setAttribute("csrfToken", ESAPI.randomizer().getRandomString(20, DefaultEncoder));

The above mentioned code is an example of a good authentication method, which uses ESAPI encoder to prevent injection attacks and authenticates user if and only if username and password combination is correct. The session id is stored in HTTP session which is stored at the server end. So, URLs will not consist of session variables. Broken Authentication Prevention is shown in Figure 6b. It shows how URL is displayed in address bar of a browser with proper session management. It is seen that there are no session variables showing up in the URL.

3.2.4 A4- Insecure Direct Object References

Due to wrong implementation of security configurations, database object such as directory, file or database key is exposed to the attackers. It can result into access of unauthorized data due to lack of an access control check or other security measure.

Fig 7a: Insecure Direct Object References Attack

Fig 7b: Insecure Direct Object References Prevention UI

Figure 7b is an example of direct object reference. On the holdings page, the information is shown of the user id provided in the URL. If the id is changed to any other id, then the details of that particular user (if exists) will be displayed.

```
protected static String getIndirectReference(String userID)
{
AccessReferenceMap map new RandomAccesaReferenceMap
();
String indirectReference map.addDirectReference
(userID).toString();
 return indirectReference;
}
```

To prevent direct object reference, ESAPI reference maps can be used. It is used by binding the user ids that are stored in the database with a random string. So that an attacker cannot guess other users' ids.

3.2.5 A5- Cross Site Request Forgery (CSRF)

A victim of a CSRF attack is forced to send a fake HTTP request to a vulnerable website while logged in, replete with the victim's session cookie and any other automatically included authentication credentials. As a result, the attacker can direct the victim's browser to transmit commands that the vulnerable program will see as originating from the user in a legitimate manner.

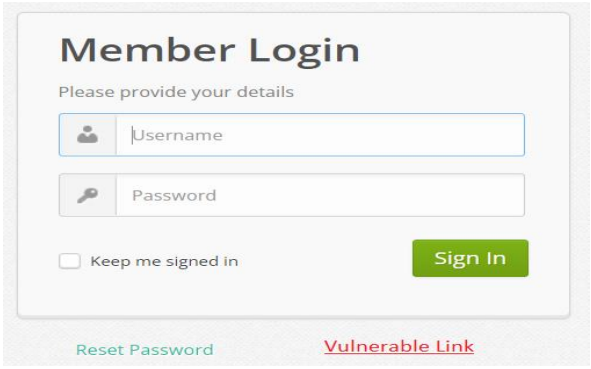

Fig 8a: Cross-Site Request Forgery Attack

If a user clicks on any vulnerable link (Figure 8a), automatic request will send to server results into execution of code.

session.setAttribute("csrfToken", ESAPI.randomizer().getRandomString(20, DefaultEncoder));

As shown in the above code, CSRF attack can be prevented by using a token which can be made required for every secure transaction. This token can be generated by using ESAPI token generators.

3.2.6 A6- Security Misconfiguration

The security configurations need to be applied in application, frameworks, application server, web server, database server since beginning as default configurations are mainly known by attackers. This involves keeping the most current releases of all programs, including any code libraries utilized by the application. To prevent Security Misconfiguration -a) Identify all components and the versions, along with all dependencies, b) Monitor the these components security , c)create security guideline for component use, d) When necessary, take into account enclosing components in security wrappers to deactivate unnecessary functionality.

3.2.7 A7- Insecure Cryptographic Storage

Proper encryption or hashing is often not used by web applications to protect sensitive data, such as credit card numbers, SSNs, and authentication credentials.

ROLL_NO	STUD_NAME	PASSWD
1234	Abhishek Jain	Omkar@1234
1235	Komal Shah	komal_1989#shah
1236	Vikrant Saraf	Svik@1236

Fig 9a: Insecure Cryptographic Storage vulnerability

ROLL_NO	STUD_NAME	HASH_PASSWD
1234	Abhishek Jain	87cc4ce599fe39236f88af5fc79e7f26
1235	Komal Shah	016d7651e020d1de98b90166221633c4
1236	Vikrant Saraf	bbd6dae11df9087725f5dc9b0af5a5b9

Fig 9b: Insecure Cryptographic Storage Prevention

Figure 9a is an example of insecure cryptographic storage. As shown in image, the passwords are stored in plaintext. If the database is compromised then attacker will easily get all the users' passwords. Figure 9b is another example of insecure cryptographic storage. Here, the passwords are hashed without using a salt. So, same passwords will be hashed to the exact same string.

ROLL_NO	STUD_NAME	HASH_PASSWD	SALT_HASH_PWD
1234	Abhishek Jain	87cc4ce599fe39236f88af5fc79e7f26	73ccb5ac8e11db426d73439b245097b1
1235	Komal Shah	016d7651e020d1de98b90166221633c4	dfae95800237636918d2c65a94648f30
1236	Vikrant Saraf	bbd6dae11df9087725f5dc9b0af5a5b9	241d88324b97bcdeefd63a188df08a6d

Fig 9c: Insecure Cryptographic Storage Prevention

As shown in the above code, Insecure Cryptographic Storage Prevention is a good example of storing passwords. Here, the passwords are concatenated with a random string called as 'salt', and then hashed. This results into different hashes of the same passwords. In this case, attackers would require 3000 years to brute force the passwords.

3. CONCLUSION

ESAPI provides secure alternatives to prevent common injection attacks like SQL injection and Command injection. It offers validated input and output encoding methods to prevent malicious input from being executed as code. ESAPI includes output encoding techniques to sanitize user-supplied data, preventing it from being interpreted as malicious scripts by web browsers. It also helps in proper contextual output encoding to protect against DOM-based Cross-Site Scripting (XSS) using Authenticator, Encoder and Validaor interfaces. ESAPI provides Cross-Site Request Forgery (CSRF) token management to help prevent unauthorized actions by ensuring that requests originate from trusted sources. It validates the tokens in incoming requests to detect and prevent CSRF attacks. ESAPI offers secure session management functionalities like session timeout handling, secure cookie management, and protection against session fixation attacks. It also provides authentication modules and methods to ensure secure user authentication processes. Thus Authentication and Session Management vulnerability is patched using Authenticator control interface. ESAPI namely AccessController interface helps enforce proper access controls by providing methods for role-based access control (RBAC), authorization checks, and proper handling of permissions and privileges. Thus Access Control is secured from attacker. ESAPI assists in implementing secure access controls using AccessController interface to prevent unauthorized access to sensitive or private resources by ensuring that direct object references are properly protected. Thus Insecure Direct Object References vulnerability can also be mitigated. It also provides default security configurations such as Encryptor interface and best practices to help developers secure their applications. It includes settings related to encryption, logging, error handling, and more. However administrator should do proper configuration of web server, application server, operating systems and data base configuration. ESAPI offers secure file upload and download functionalities with built-in protection like HTTPUtilities upload interface against path traversal attacks and verification of file types and sizes. Thus Secure File Handling protects the files from download or upload or any modifications. ESAPI promotes secure error handling and logging practices using Logger interface to prevent information leakage that can be exploited by attackers. ESAPI provides a set of cryptographic functions and utilities such as Randomizer for securely handling sensitive data, such as encryption, hashing, and secure random number generation. Thus by incorporating ESAPI into web application development, developers can leverage its secure coding practices, libraries, and APIs to address various vulnerabilities that may be part of the OWASP Top 10 2023 attacks.

There are various vulnerabilities in web technology components like HTML, Java script, JSP, JAVA and backend Oracle database that have been used for the development of web application. Hence apart from ESAPI, secure coding can be written by the developers. The author's research is marching towards this goal.

4. REFERENCES

[1] Elder, S. E., Zahan, N., Kozarev, V., Shu, R., Menzies, T., and Williams, L. 2021. Structuring a Comprehensive Software Security Course Around the OWASP Application Security Verification Standard. IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSESEET), 95-104.

[2] Marchand-Melsom, A., and Mai, D. B. N. 2020. Automatic repair of OWASP Top 10 security vulnerabilities: A survey. IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20), Seoul, 23-30.

[3] Spoto, F., Burato, E., Ernst, M. D., Ferrara, P., Lovato, A., Macedonio, D., Spiridon, C. 2019. Static Identification of Injection Attacks in Java. ACM Transactions on Programming Languages and Systems, Vol. 41, No. 3, 18-58.

[4] Gedam, M. N., and Meshram, B. B. 2022. Proposed Secure 3-Use Case Diagram. International Journal of Systems and Software Security and Protection, Volume 13, Issue 1, IGI Global.

[5] Gedam, M. N., and Meshram, B. B. 2019. Vulnerabilities & Attacks in SRS for Object-Oriented Software Development. Lecture Notes in Engineering and Computer Science: Proceedings of The World Congress on Engineering and Computer Science, 94-99.

[6] Lala, S. K., Kumar, A., Subbulakshmi, T. 2021. Secure Web development using OWASP Guidelines. International Conference on Intelligent Computing and Control Systems(ICICCS), 323-332.

[7] Brown, L. D., Hua, H., and Gao, C. 2003. A widget framework for augmented interaction in SCAPE.

[8] Ingle, D.R., and Meshram, B. B. 2012. Attacks on Web Based Software And Modelling Defence Mechanisms. International Journal of UbiComp.

[9] Chavan, S. B., and Meshram, B. B. 2013. Classification of web application vulnerabilities. International Journal of Engineering Science and Innovative Technology (IJESIT).

[10] Khochare, N., Chalurkar, S., and Meshram, B. B. 2012. Survey on Web Application Vulnerabilities Prevention Tools. International Journal of Managment, IT and Engineering.

[11] Available Online - https://www.javadoc.io/doc/org.owasp.esapi/esapi/2.0.1/org/owasp/esapi/AccessReferenceMap.html

[12] Available Online - https://www.javadoc.io/static/org.owasp.esapi/esapi/2.0.1/index.html?org/owasp/esapi/Authenticator.html

[13] Interface Encoder, Jeff Williams (2007) - https://www.javadoc.io/doc/org.owasp.esapi/esapi/2.0.1/org/owasp/esapi/Encoder.html

[14] OWASP Enterprise Security API - https://owasp.org/www-project-enterprise-security-api/

[15] Interface Encrypter, Jeff Williams (2007) - https://javadoc.io/doc/org.owasp.esapi/esapi/2.0.1/org/owasp/esapi/Encryptor.html

[16] Interface Executor, Jeff Williams (2007) - https://javadoc.io/doc/org.owasp.esapi/esapi/2.1.0/org/owasp/esapi/Executor.html

[17] Interface HTTPUtilities, Jeff Williams (2007) - https://www.javadoc.io/doc/org.owasp.esapi/esapi/2.0.1/org/owasp/esapi/HTTPUtilities.html

[18] Interface IntrusionDetector, Jeff Williams (2007) - https://javadoc.io/doc/org.owasp.esapi/esapi/2.0.1/org/owasp/esapi/IntrusionDetector.html

[19] Available Online - https://www.javadoc.io/doc/org.owasp.esapi/esapi/2.1.0/org/owasp/esapi/Logger.html

[20] Interface Randomizer, Jeff Williams (2007) - https://javadoc.io/doc/org.owasp.esapi/esapi/2.0.1/org/owasp/esapi/Randomizer.html

[21] Interface SecurityConfiguration, Jeff Williams (2007) - https://www.javadoc.io/doc/org.owasp.esapi/esapi/2.1.0.1/org/owasp/esapi/SecurityConfiguration.html

[22] Interface User, Jeff Williams (2007) - https://www.javadoc.io/doc/org.owasp.esapi/esapi/2.0.1/org/owasp/esapi/User.html

[23] Interface Validator, Jeff Williams (2007) - https://www.javadoc.io/doc/org.owasp.esapi/esapi/2.0.1/org/owasp/esapi/Validator.html

[24] Rodríguez, G. E., Torres, J. G., Flores, P., and Benavides, D. 2019. Cross-site scripting (XSS) attacks and mitigation: A survey. Elsevier.

[25] Gedam, M. N., and Meshram, B. B. 2019. Proposed Secure Content Modeling of Web Software Model. NCRIEST, Nashik.

[26] Available Online - https://javadoc.io/doc/org.owasp.esapi/esapi/2.0.1/org/owasp/esapi/AccessController.html