# DEVELOPMENT OF VULNERABILITY FREE WEB APPLICATION WITH SECURED DATA

CH.Madhurya[1*], A. Ajay Kumar[2]
[1*]PG Scholar, Dept. of Computer Science and Engineering, GVP College of Engineering (A), Visakhapatnam, India
[2] Assistant Professor, Dept. of Computer Science and Engineering, GVP College of Engineering (A), Visakhapatnam, India

***Abstract***: the proposed system keeps track of the dashboard which in turn tracks various requests by the client. Along with that, it also considers security of the data by enforcing various security mechanisms such as SHA-256, salt and pepper, for password storage and data storage. Using merkle tree, hash the data to make data immutable which in turn provides more security for the data. The dashboard is protected from cross-site scripting (XSS) , Sensitive data exposure and cross-site request forgery (CSRF) which are seventh and third of top ten web application vulnerabilities on Open Web Application Security Project (OWASP) according to recent rankings of OWASP 2017. The main advantage of the proposed system is to hide the data by using merkle tree.

***Keywords***— OWASP, XSS, CSRF, Merkle Tree, Immutable

## INTRODUCTION

A dashboard is a user interface or a software based control panel which is used to acquire consolidate data across any organization which provides in depth analysis, which helps an organization efficiently develop their goals and strategies. Dashboards are mostly web based and accessed via web browsers. The web based portal is linked to a database at back-end which allows the data in tables and reports to be updated constantly. The developed system is a web-based application with centralized database. The technologies used is XAMPP (Cross Platform(X), Apache(A), Maria DB(M), PHP(P), Perl(P)). Any web-based applications or dashboards are vulnerable to different types of attacks such as Cross-site Scripting (XSS), Sensitive data exposure and Cross-site Request Forgery (CSRF) and many more. The proposed system mainly concentrates on security of data. XSS is a security bug that can affect web application

XSS is a type of injection which occurs when (a) Attacker sends malicious code in the form of script to different end user. On successful execution of the malicious code, the behaviour of the system or application will be completely changed. It also steals user private data and accomplishes attacker's objectives like cookie stealing, session token theft or to launch others attacks.

(b) The data included in the dynamic content, sent to the web user without validating or encoding the malicious content. If it is successful, then he can gain access to victim's account. Attacker sends it by using email, web message board and waits for the user to click on it. XSS are categorized into three types: Persistent or stored, Reflected or non-persistent and DOM-based attacks. Persistent attacks-where the injected script will be stored permanently on the target servers such as in comment field, database, etc.,

Persistent attacks are sometimes referred as Type-1 XSS. Non-persistent attacks-where injected script is reflected off the web server. Examples such as error message, search result or any input sent to the server as a part of the request. Non-persistent attacks sometimes referred as Type-2 XSS. DOM (Document Object Model) based attacks-subset of client XSS. It appears in DOM instead of part of the HTML. In this attack the response of the attack and HTML source code will exactly be the same and can be observed only on run-time.

Sensitive data exposure occurs when an application does not adequately protect sensitive information. The data can vary anything from passwords, session tokens and more. It deserves extra protection such as encryption and special precautions when exchanged with the browser. CSRF attacks are also known as one-click attack or session riding. It forces the end user to execute unwanted actions when they are currently authenticated.

Generally, CSRF targets state changing requests, because the attacker has no way to see the response to the forged requests. Immutability is obtained through merkle tree. Also called as hash tree. In merkle tree, each leaf node is a hash of transactional data and each non-leaf node is a hash of its previous hashes. It is a binary tree which requires even number of nodes. Merkle tree provides a mean to prove integrity and validity of data and it requires little memory space.

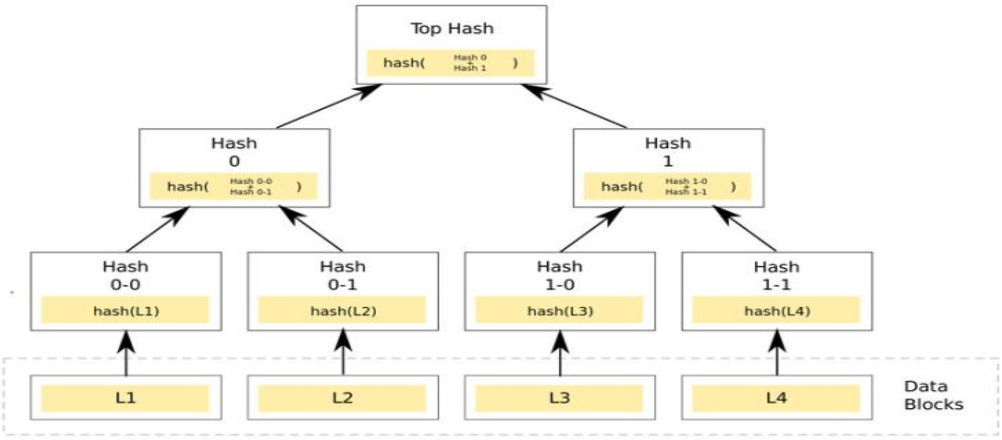


Fig.1 A simple merkle tree

## II. LITERATURE SURVEY

Several research and studies has done in the last few years for preventing XSS related issues in web applications. Even though many solutions are produced by the researchers, XSS vulnerability still exists in the applications. Similarly for CSRF and sensitive data exposure.

"BRB dashboard : A web-based statistical dashboard" by Siddharth Mahajan, Mitesh Parekh, Hardik Patel, Sharvari Patil focuses on testing the dashboard i.e., whether it is secured from security vulnerabilities and also uses SHA-256 for storing the passwords [1].

"Improved Cross-site scripting filter for input validation against in web services" by Elangovan Uma, Arputharaj Kannan focuses on detecting the cross-site scripting attacks [2].

"A report on CSRF security challenge and prevention techniques" by P Yadav and C D Parekh focused on recent challenges faced in CSRF attacks [3].

"XSS vulnerability assessment and prevention in web application" by A Shrivastava, S choudhary and A Kumari focused on detecting and preventing different types of XSS attacks [4].

## III. METHODOLOGY

In this section, we are going to discuss about methodology of the work which is specified earlier. The procedure for storing and securing data will be discussed in Algorithm 1 and preventing from CSRF attacks will be discussed in Algorithm 2. Immutability of data is discussed in Algorithm 3.

**Algorithm secure (data):**

Algorithm for storing the data securely

To protect data from security attacks, there is a need to store the data securely without getting effected from any attacks.

The algorithm is as follows:

Input: Data

Output: Secured Hash value

1. Read the given input

Data ← input();

2. Generate random salt value

Salt ← random();

3. Generate a random pepper value

Pepper ← random();

4. Append both salt and pepper values and generate a hash value
Hash ← hash(salt + pepper);

5. Append both generated hash and data and then hash it.
Hashed data ← hash(data + Hash)

6. Store the hashed data and repeat the steps 1 to 5 for remaining data.

In above algorithm, we specified that a hash value should be generated. The hash value is generated by using SHA-256 which is secured one. The second phase includes how to protect the application from XSS attack. At first we should not include any untrusted data into any HTML document.

Escape the script, div, p tags because including untrusted data inside the script is very dangerous, because it is easy to switch into execution. Along with it try to escape the special characters like <, >, &, “, / etc as follows:

< →&lt

>→ &gt

& → &amp

“ → &quot

/ → &#x2F

Even though when we include <script>, <p>,  tags in any form data, it should be removed before storing the data and it should not affect the database. The third phase includes protecting the application from CSRF attack.

**Algorithm token_gen(data):**

1. Generate a random token id and generate a hash for each user session.
token_get ← hash(random());

2. Add generated hash to form
<a href=”# ?key={$token_get}”>

3. On server side, get the token value by using request.get( ) method.
Token ← request.get(‘key’);

4. Check whether the token is valid or not
Hash(Token) ← request.get(‘key);

5. Repeat steps 1 to 4 for each users session

The fourth phase includes preventing sensitive data exposure.

Identify the sensitive data from the given data and use algorithm 1 to prevent the data from exposure.

**Algorithm immute(data):**

Input: Data

Output: Root hash

1. Read the data from each field
Data ← input();

2. Generate a hash for each field data and make them as chid nodes.
Child_hash ← hash(data)

3. Check whether there are even number of nodes
4. If not then make the last hash duplicate to create even number of nodes.
5. Repeat step 2 till selected data in the record completes.
6. Merge the chid hash and hash it to get the parent node
Parent_hash←
hash(hash(child_hash1),hash(child_hash2));

7. Repeat the process of hashing parent_hash until we get the root node.

## IV. RESULTS

### 4.1 Result

Fig. 4.1 shows the data which is converted into hashes after applying secure (data) algorithm. The fields receiver and transaction id are the sensitive data which should not be in plain text

| receiver | transactionid | transferred_at | an |
|---|---|---|---|
| 34f2ed424473c7de0ee237a812ca279b | 0c52d419a421fb13bb58357e67b7fb4b | 2018-06-04 12:17:23 | |
| 34f2ed424473c7de0ee237a812ca279b | 0ea6f098a59fcf2462afc50d130ff034 | 2018-06-04 12:18:45 | |
| 34f2ed424473c7de0ee237a812ca279b | 93963474edfd08f1f1e7244f663b4708 | 2018-06-04 12:24:29 | |
| 34f2ed424473c7de0ee237a812ca279b | 28c4661da88dffe967089b43666e4844 | 2018-06-04 12:24:40 | |
| 9ba205f93383853fd59da60d55808976 | 4b880d619bbbcbbea22b13bfa30a1ace | 2018-06-04 12:25:53 | |
| 9ba205f93383853fd59da60d55808976 | 7b01caa07b92e8424d45487ee923bdd4 | 2018-06-04 12:27:20 | |
| 9ba205f93383853fd59da60d55808976 | 1dba3025b159cd9354da65e2d0436a31 | 2018-06-04 12:27:29 | |
| 130d16df066d5e40f306c1058316f13c | 48fd3e91841d8619c84400e661895675 | 2018-06-04 12:28:33 | |
| 9ba205f93383853fd59da60d55808976 | 6dbbe6abe5f14af882ff977fc3f35501 | 2018-06-04 12:56:50 | |
| 9ba205f93383853fd59da60d55808976 | 92699ee8e81849b1817a5d73d3bf8e02 | 2018-06-04 13:55:09 | |
| 9ba205f93383853fd59da60d55808976 | 7c2c48a32443ad8f805e48520f3b26a4 | 2018-06-05 10:55:31 | |
| 34f2ed424473c7de0ee237a812ca279b | 2c8acfd9373aef9b1caa21e451877fe1 | 2018-06-05 16:10:44 | |
| c46dc19f9852710f3af9e31de2e0318d | 38cca1363531ea990168f56b051baa79 | 2018-06-08 10:58:52 | |
| d5e5164d850271ad387220d0936373a5 | 712711c4792aae089713c1858fbe2f6e | 2018-11-05 10:14:23 | |

Fig.4.1 Secured data after applying algorithm secure (data)

### 4.2 XSS Results

(a) Fig. 4.2.1 shows how the tags like script, image are going to affect the data in the database and it also shows onmouse tags which also affects the application data.

| | | | |
|---|---|---|---|
| Edit Copy Delete | 23 | 462844198 | <script> |
| Edit Copy Delete | 24 | 450883812 | <script>alert(document.cookie) |
| Edit Copy Delete | 25 | 238399668 | <script>alert("hacked")</scrip |
| Edit Copy Delete | 26 | 891393693 | IMGSRC=java& |
| Edit Copy Delete | 27 | 750664534 | IMG SRC="jav	ascript:aler |
| Edit Copy Delete | 28 | 557697557 | IMG SRC="jav	ascript:aler |
| Edit Copy Delete | 29 | 773402502 | <a onmouseover=alert(document. |
| Edit Copy Delete | 30 | 495857559 | <IMG SRC=javascript:alert(Stri |

Fig.4.2.1 List of some XSS attacks before applying XSS prevention algorithm.

(b) Fig 4.2.2 shows how the tags are stored or removed after applying XSS preventive measure

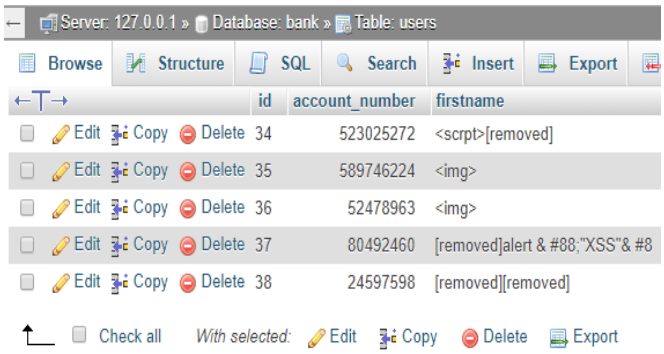


Fig. 4.2.2 Result after applying the algorithm.

4.3 Immutablity Results

(a) Fig. 4.3.1 shows Transaction ID, From, To values which are in hash form. Here Transaction ID is Merkel root and From and To are the child nodes.

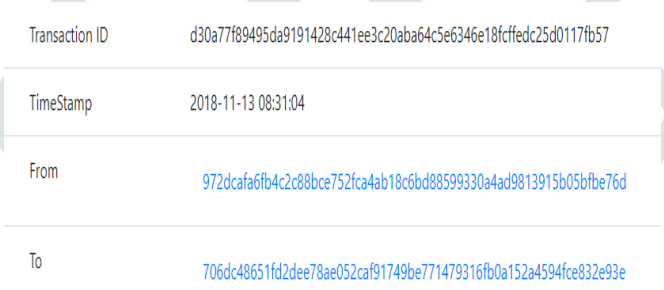


Fig.4.3.1 Generation of parent hash nodes from Algorithm immute(data)

## V. CONCLUSION AND FUTURE SCOPE

By encrypting password with SHA-256 with salt and pepper, there is an improvement in the authentication process. So that trusted user can access the information. As the industry is depending more and more on information technology, dashboards are becoming more common and critical for decision making.
In order to keep this information safe and secure the application should be prevented from web attacks.
Although we are protecting the application against different web attacks there may be a possibility of breaching it. So, in future the algorithms should be generated in such a way that it can store and secure the data properly without getting affected from any web attacks.

## REFERENCES

[1] S. Mahajan, M. Parekh, H. Patel and S. Patil, "BRB dashboard: A web-based statistical dashboard," *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, Coimbatore, 2017, pp. 1-6.

[2] Elangovan Uma, Arputharaj Kannan, "Improved Cross-Site Scripting Filters for input validation against attacks in web services," Kuwait Journal of Science, Vol.41, no. 2, pp.175-203,2014.

[3] P. Yadav and C. D. Parekh, "A report on CSRF security challenges & prevention techniques," *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, Coimbatore, 2017, pp. 1-4.

[4] A. Shrivastava, S. Choudhary and A. Kumar, "XSS vulnerability assessment and prevention in web application," *2016 2nd International Conference on Next Generation Computing Technologies (NGCT)*, Dehradun, 2016, pp. 850-853.

[5] 5. B. A. Scriber, "A Framework for Determining Blockchain Applicability," in *IEEE Software*, vol. 35, no. 4, pp. 70-77, July/August 2018.

[6] P. Chaudhary, B. B. Gupta and S. Yamaguchi, "XSS detection with automatic view isolation on online social network," *2016 IEEE 5th Global Conference on Consumer Electronics*, Kyoto, 2016, pp. 1-5.

[7] M. E. Masri and N. Vlajic, "Current state of client-side extensions aimed at protecting against CSRF-like attacks," *2017 IEEE Conference on Communications and Network Security* (CNS), Las Vegas, NV, 2017, pp. 390-391

[8] A. Sudhodanan, R. Carbone, L. Compagna, N. Dolgin, A. Armando and U. Morelli, "Large-Scale Analysis & Detection of Authentication Cross-Site Request Forgeries," 2017 IEEE European Symposium on Security and Privacy (EuroS&P), Paris, 2017, pp. 350-365.