Prevention, Detection and Mitigation of Cross Site Scripting Attacks in Web Applications

[1]Sanket Patel, [2]Priyanka Vishwasrao, [3]Prof. Sakina Shaikh

Master of Computer Application
Sardar Patel Institute of Technology
Mumbai, India

***Abstract*— With the increasing dependencies on the web applications for day-to-day activities, users are exposed to an all time high risk of being affected by cyber attacks. One of such attack is the Cross Site Scripting (XSS) attacks that seems to be passive in nature but directly impacts the experience of the user session. Open Web Application Security Project (OWASP) and Common Vulnerabilities and Exposures (CVE) reported Cross-Site Scripting (XSS) as one of the most serious vulnerabilities in the web applications. Though many vulnerability detection approaches have been proposed in the past, existing detection approaches have the limitations in terms of false positive and false negative results. This paper proposed an automated browser mediating approach to authenticate the scripts on the page and thus mitigating the XSS vulnerabilities in the web applications.**

***Keywords*— Cross Site Scripting (XSS), Web Application Security, Pattern Matching, Browser Mediating Approach**

I. INTRODUCTION

The loss of privacy is a rising issue among web rights activists as more and more users sign up to share their lives with the world. If a major exploit is ever found that allowed anyone to see anything posted or otherwise told on any media site, it would have huge repercussions for both the users and the network's good status. Measures must be taken by social media developers for their users' sakes to stop the misuse of private information, take action against the posting of media containing non-consenting individuals and their own carelessness in handling user data.

Attacks on institutions like bank would cause heavy loss to the public as well to the entities holding stake of those bank, it's obvious that attackers get attracted to mine of money which could be a synonym to bank.

Not just big e-commerce websites but also small e-commerce sites are often the target of attacks, with hackers taking advantage of companies without the dedicated security staff and expertise of a company that's in the top half of the Fortune 500. And while breaches at smaller companies may not make the headlines (if they are detected at all), the number of small ecommerce sites – the long tail – provides a tempting volume of sites to attack.

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Inherently, XSS is different from other types of online attacks wherein there is a direct attempt to harm the user or the application service. Whereas, XSS is majorly used to modify the experience of the end user thereby furnishing an altered version of the web application instead of the version actually requested by the user.

Understanding the request-response mechanism

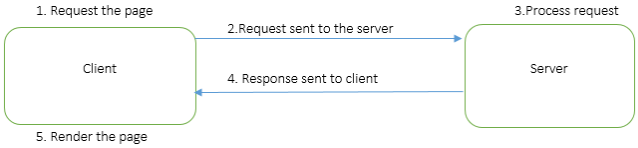

Fig. 1 : Basic workflow of a web application

1. Web pages are written in HTML, Hypertext Markup Language. A markup language is a computer language that describes the layout, format and content of a page. The Web browser renders the page according to the HTML code. Here goes the client side script for validations.
2. Web Page is requested through HTTP protocol from Server where the webpage originally resides. The server receives the request for a page sent by client. The browser connects to the server through an IP Address; the IP address is obtained by translating the domain name.
3. Web servers respond to a browser's request for a web page and deliver it through the internet.
4. This response is then sent to client in form web page through HTTP response mechanism the response is sent in html page hence it can be rendered on client.
5. The script present on client side helps to validate the content sent to client from server also renders it onto the browser.

On analyzing the above operations in the traditional web applications, we can highlight a few of the possible channels that an attacker can exploit. On analyzing the existing workflow of a web application, one can clearly say that there are two distinct entities located at a physically separate location but connected via a communication link which in our case is the internet. One of the most common way that an exploiter can opt for is the man-in-the-middle attack. The attacker can intercept the request from the user, modify the request and forward the same to the server. The server receives and processes the request and sends the response to the user. Again, the attacker may intercept the response, alter the original message with the forged message and then finally forwards it

the intended user. Now the user views the forged message and thus is a victim of an online attack.

XSS can be called as an adaptation of the traditional man-in-the-middle attack except for the fact that the middleman resides on the client system itself.

II. CONCEPT

Cross-site scripting (XSS) is a code injection attack that allows an attacker to execute malicious JavaScript in another user's browser.

The attacker does not directly target his victim. Instead, he exploits a vulnerability in a website that the victim visits, in order to get the website to deliver the malicious JavaScript for him. To the victim's browser, the malicious JavaScript appears to be a legitimate part of the website, and the website has thus acted as an unintentional accomplice to the attacker.

The only way for the attacker to run his malicious JavaScript in the victim's browser is to inject it into one of the pages that the victim downloads from the website. This can happen if the website directly includes user input in its pages, because the attacker can then insert a string that will be treated as code by the victim's browser.

In the example, a simple server-side script is used to display the latest comment on a website. The script assumes that a comment consists only of text. However, since the user input is included directly, an attacker could submit this comment: "<script>...</script>". Any user visiting the page would now receive the response as When the user's browser loads the page, it will execute whatever JavaScript code is contained inside the <script> tags. The attacker has now succeeded with his attack.

The ability to execute JavaScript in the victim's browser might not seem particularly malicious. After all, JavaScript runs in a very restricted environment that has extremely limited access to the user's files and operating system. In fact, you could open your browser's JavaScript console right now and execute any JavaScript you want, and you would be very unlikely to cause any damage to your computer [5].

However, the possibility of JavaScript being malicious becomes clearer when you consider the following facts:

- JavaScript has access to some of the user's sensitive information, such as cookies.
- JavaScript can send HTTP requests with arbitrary content to arbitrary destinations by using XMLHttpRequest and other mechanisms.
- JavaScript can make arbitrary modifications to the HTML of the current page by using DOM manipulation methods.

III. CLASSIFICATION OF XSS ATTACKS

Cross-site Scripting can be classified into three major categories [4] —

- Stored XSS
- Reflected XSS
- DOM-based XSS

1. Stored XSS

Stored XSS attacks involves an attacker injecting a script (referred to as the payload) that is permanently stored (persisted) on the target application (for instance within a database). The classic example of stored XSS is a malicious script inserted by an attacker in a comment field on a blog or in a forum post.

2. Reflected XSS

In Reflected XSS, the attacker's payload script has to be part of the request which is sent to the web server and reflected back in such a way that the HTTP response includes the payload from the HTTP request.Using Phishing emails and other social engineering techniques, the attacker lures the victim to inadvertently make a request to the server which contains the XSS payload and ends-up executing the script that gets reflected and executed inside the browser.

3. DOM- based XSS

DOM-based XSS is an advanced type of XSS attack which is made possible when the web application's client side scripts write user provided data to the Document Object Model (DOM). The data is subsequently read from the DOM by the web application and outputted to the browser.

IV. PROPOSED APPROACH

In order to prevent and minimize the impact of an XSS attack, we propose to safeguard both the server as well as the client. A persistent XSS is carried on with the intention to harm the server and indirectly all the clients that the server provides service to. Whereas a non-persistent XSS attack is directly intended at the end-user or the client. Following are the proposed approach that can be used to prevent both the types of XSS attacks.

A. Persistent XSS Attacks

A major part of data submitted by the client that is persisted by the server is textual data or multimedia data. To put it in other words, except for few exceptional cases, there are not many situations where a user submits an executable content. Keeping this proposition in mind, we can validate the request at the browser level itself. Following steps needs to be taken :

1. A client enters the data into a form that will be submitted to the server.
2. Assuming that the client is infected by an XSS script, the client request will be tampered by the attacking script without the knowledge of the client.
3. The request will be prevented by the browser even before it is submitted to the server. The browser will do so based on the content of the request message.
    a. If the request message contains any executable content like a javascript, the request will be cancelled.
    b. Else, the request will be treated as a normal request and will be forward to the server for processing.

For exceptional scenario, where there is a need to submit executable content to the server, following steps may be taken:

a. The server specifies its intent to accept executable data in its headers.
b. The client enters the data that may contain executable script.
c. The browser in this case will skip the validation of request message and directly forward the same to the server.

B. Non-Persistent XSS Attacks

There are situations in which XSS script changes the content of the server's response. In such situations, the client is made to interact with additional content injected by the XSS script. Many a time such additional content is the javascript code. These types of attacks are not easy to be detected as the server is completely unaware of the the injection of a third-party code into its response. To prevent such attacks following steps can be taken:

1. On the server-side a descriptor file needs to be maintained that may be in an XML format. The file will contain all the list of all the scripts blocks that may be received in the server response.
2. The descriptor file is sent to the client with the initial request and then stored in the browser's cache for the session.
3. When the browser starts to render the response on the screen, following steps takes place:
    a. Whenever a script block is encountered, the browser checks for its authenticity in the descriptor file.
    b. If the block is found in the descriptor, it is rendered on the screen.
    c. Else, the block is removed from the response.

This above approach is feasible for cases where the XSS attack adds additional scripts to the server response which will be eventually be ignored by the client browser.

Alternatively, another approach may be adopted wherein the templatization of the server response needs to be done. Following are the steps for the same:

1. For every response generated for the request, minimized version of the DOM structure will be sent along the response that can only be interpreted by the browser.
2. As the DOM structure is pre-defined, any kind of injection or modification in the DOM structure done by the XSS attack will be identified and eventually be nullified.

V. LIMITATIONS

The solutions proposed in this paper to prevent XSS attack are most suitable for web applications where the structure is static in the long-run. There is also an additional overhead of maintaining an descriptor file as well as the response template on the server. Due to the above limitation, the proposed approach works best with but not limited to web applications that have high security needs like online banking.

VI. CONCLUSION AND FUTURE WORK

With the increasing dependencies on the web applications for day-to-day activities, users are exposed to an all time high risk of being affected by cyber attacks. This paper proposed an automated browser mediating approach to authenticate the scripts on the page and thus mitigating the XSS vulnerabilities in the web applications. The paper proposed two approaches - descriptor file based approach for preventing persistent XSS attack and template based approach for preventing non-persistent XSS attacks.

The proposed solution has a few overheads in maintaining additional descriptor and template file. This leads to an increase in bandwidth requirements for every request. As a further study in this field, there is a need to find a solution to minimize the file size and also optimize caching at the browser level. Also, web applications that have a highly dynamic structure, i.e. applications that have a fluctuating DOM structure cannot be prevented from XSS attacks using the proposed approach. A study needs to be conducted to handle this exception.

REFERENCES

[1] Mukesh Kumar Gupta, Mahesh Chandra Govind, Girdhari Singh, Priya Sharma, "XSSDM: Towards Detection and Mitigation of Cross-Site Scripting Vulnerabilities in Web Applications", 2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)

[2] Yu Sun, Dake He, "Model Checking for the Defense against Cross-site Scripting Attacks", 2012 IEEE DOI 10.1109/CSSS.2012.537

[3] Mukesh Kumar Gupta, Mahesh Chandra Govind, Girdhari Singh, "Predicting Cross-Site Scripting (XSS) Security Vulnerabilities in Web Applications", 2015 12th International Joint Conference on Computer Science and Software Engineering (JCSSE)

[4] "Types of XSS", Acunetix - http://www.acunetix.com/websitesecurity/xss/

[5] "Excess XSS :A Comprehensive tutorial on cross site scripting", Excess XSS - http://excess-xss.com/