XSS attack Prevention using Ramp Secret Sharing

Mrs.Tejaswini H. Aware[1], Prof. D. S. Kulkarni[2]
Department of Computer Engineering
[1, 2] D. Y. Patil College of Engineering Ambi Pune.

Abstract-*The web applications are developed and accessed by millions of users for various services. These applications are developed using various technologies like HTML, JavaScript, AJAX, XML etc. But the vulnerabilities at the design level in these technologies lead to security breach, resulting in theft of the users credentials. Thus, the security of these applications is becoming an important concern to ensure the users authentication and privacy. Cross site scripting attack (XSS) is also an exploitation of these vulnerabilities existing in the web applications. XSS still remains a big problem for web applications, despite the bulk of solutions provided so far. Content Security Policy (CSP) is also an approach to prevent this code injection. This paper studies the browser compatibility issues in deploying CSP to mitigate XSS vulnerabilities and also discusses how to resolve this incompatibility. A content security policy (CSP) can help Web application developers and server administrator's better control website content and avoid vulnerabilities to cross site scripting (XSS). In experiments with a prototype website, the authors CSP implementation successfully mitigated all XSS attack types in four popular browsers. Among the many attacks on Web applications, cross site scripting (XSS) is one of the most common. An XSS attack involves injecting malicious script into a trusted website that executes on a visitors browser without the visitors knowledge and thereby enables the attacker to access sensitive user data, such as session tokens and cookies stored on the browser. With this data, attackers can execute several malicious acts, including identity theft, key logging, phishing, user impersonation, and webcam activation. Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks.*

Keywords-Cross site scripting attack, Content Security Policy (CSP), Vulnerabilities at the design level, Data injection attack

I. INTRODUCTION

Avoid the web application attacks, the web browser security model is built on the same origin policy that isolates one origin from the other thus providing the developers a safe sandbox environment to build these applications in which the code from one origin (http://self.com) has access to only http://self.com data and the code from other origin (http://other.com) is not permitted to access http://self.com data. But the attackers by-pass this policy by exploiting cross site scripting vulnerabilities in the web applications. He injects his own script into the web applications and later this injected script will get embedded along with the actual intended response from the website whenever any user visits that particular web page. The victims browser executes all of the code that shows up on a page as being legitimately part of that pages security origin since the browser is not able to differentiate between the injected and the intended code. Thus, Cross-Site Scripting attack (XSS) is a code injection attack performed to exploit the vulnerabilities existing in the web applications by injecting html tag / JavaScript functions into the web page so that it gets executed on the victims browser when one visits the web page and successfully accesses to any sensitive victims browser resource associated to the web application (e.g. cookies, session IDs, etc.).

Successful cross site scripting can result in serious security violations for both the web site and the user. Web Applications have become one of the most important ways to provide a broad range of services to users. In the recent years, web-based attacks have caused harm to the users of web applications. Most of these attacks occur through the exploitation of security vulnerabilities in the web-based programs. So, the mitigation of these attacks is very crucial to reduce its harmful consequence. The main issue is that if malicious content can be introduced into a dynamic web page, neither the web site nor the client is capable of recognizing that anything like this happened and prevent it.

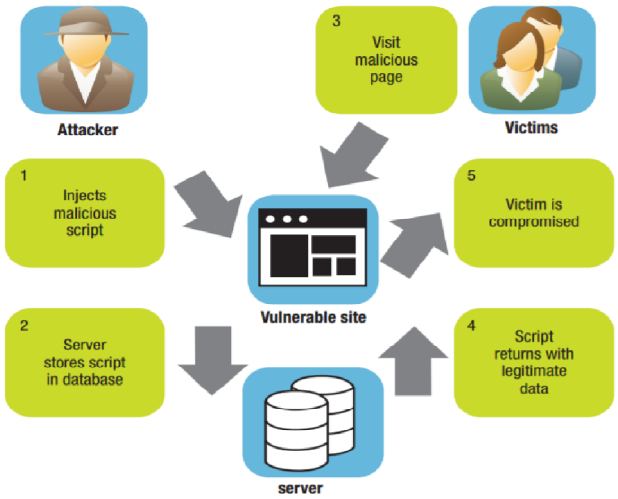


Fig.1: Block diagram of cross site scripting attack

A) BASIC CONCEPTS

The security of internet functions is fitting a foremost difficulty to make sure the persons authentication and privacy. Cross site scripting attack (XSS) can be an exploitation of these vulnerabilities existing within the net purposes. XSS nonetheless remains a giant quandary for net functions, regardless of the bulk of options furnished to this point. Content security coverage (CSP) can be a method to hinder this code injection. This paper reviews the browser compatibility issues in deploying CSP to mitigate XSS vulnerabilities and in addition discusses how to resolve this incompatibility.

B) ADVANTAGES OF PROPOSED SYSTEM

1. Since the requests with waiting time D are all assigned to temporary servers, it is apparent that all service requests can guarantee their deadline and are charged based on the workload according to the SLA. Hence, the revenue of the service provider increases.

2. Increase in the quality of service requests and maximize the profit of service providers.

3. This scheme combines short-term renting with long-term renting, which can reduce the resource waste greatly and adapt to the dynamical demand of computing capacity

C) SYSTEM METHODOLOGY

A combination of client side and server side solution which detects and prevents cross site scripting attacks based on the OWASP prevention guidelines. For this XSS checker function is added on both client and server. If an attack is detected at client side only it will not be forwarded to server thus saving runtime overhead which was not possible with server side solution and attacks occurring when requestis forwarded from client to server will also be detected and prevented which was not possible with client side solution.

II. REVIEW OF LITERATURE

The main motive for performing XSS attack is to execute malicious JavaScript in the victims browser to steal victims authentication details. Various prevention measures have been suggested to counter XSS vulnerabilities. It includes encoding, sanitization, blacklisting, white listing approaches etc. [4]. A survey has been done on the detection and prevention techniques proposed by various researchers to mitigate XSS risks. XSS vulnerabilities can be detected by performing static and dynamic analysis on the web applications. Many researchers have carried out their study in this domain and some are working on the server side solutions and some are working in client side solutions. Some researchers also carried out their study in the CSP domain [5] [6]. Z. Mao et. al. [7] introduced a technique known as BEAP (Browser-Enforced Authenticity Protection) that enabled the web browsers to limit the users credentials (cookies, session Id, authentication tokens etc.) to get transmitted with the requests on the basis of the policy that is interpreted on the basis of details surrounding the action that caused the HTTP request and it denies the transmission of users credentials when they arent deemed needed by BEAP, but the requests are still transmitted in a safer way. M. T. Louw et. al. [8] introduced a server side prevention technique against XSS attacks. This technique known as BEEP (browser enforced embedded policies) where a web site can embed a policy in its pages to specify which scripts are allowed to run. The browser, which knows exactly when it will run a script, can enforce this policy perfectly. O. Hallaraker and G. Vigna [9] proposed a mechanism for detecting malicious JavaScript. The system consists of browser embedded script auditing component and IDS to process the audit logs and compare them to signature of already known malicious behavior or attacks. H. J. Wang et. al. [10] introduced an operating systemmechanism, known as Mashup OS which improved the same origin policy by implementing granularity that does not exist in the same origin policy. This approach enables a site to specify a policy for an entire page that is then worked into the page regardless of the content injected by proposing a trio of new HTML tags that help a site express its relationship to other sites it may want to use as content libraries. C. Reis et. al. [11] suggested an approach to draw boundaries around programs, unintended code, programs in the browser, and other pieces of web sites and also discussed the reasons for non-applicability of uniform security policies to the entire web page. J. Burke et. al. [12] developed a policy to allow a web page to specify the URLs from where scripts are allowed to be loaded and where they are not allowed. This approach modifies the way that XML Http Requests (AJAX) can behave much in a similar way to how we address all resources, not just scripts. S. Shalini and S. Usha [13] provided a client-side solution to mitigate XSS attack that employs a three step approach to protect cross site scripting. This technique found to be platform independent and it blocks suspected attacks by preventing the injected script from being passed to the JavaScript engine rather than performing risky transformations on the HTML. Engine Kirda et. al. [14] presented Noxes, a client-side solution to mitigate cross-site scripting attacks. Noxes acts as a web proxy and uses both manual and automatically generated rules to mitigate possible cross-site scripting attempts. Stefano Di Paola and Giorgio. F [15] described a universal XSS attack against the Acrobat PDF

plug-in. When the client clicks the link and the data is processed by the page (typically by a client side HTML embedded script such as JavaScript), the malicious JavaScriptpayload gets embedded into the page at runtime. Kailas Patilet. al. [17] had done measurement study of CSP on real world applications to understand the difficulties from site developers point of view to adopt CSP policy. They also implemented User CSP tool as a firefox extension that uses dynamic analysis to automatically infer CSP policies to enforce client-side policies on websites. They found 27 websites including Twitter implementing CSP in their respective web pages.

III. EXISTING SYSTEM AND PROPOSED SYSTEM

A) EXISTING SYSTEM

An XSS attack involves injecting malicious script into a trusted website that executes on a visitors browser without the visitors knowledge and thereby enables the attacker to access sensitive user data, such as session tokens and cookies stored on the browser.1 With this data, attackers can execute several malicious acts, including identity theft, key logging, phishing, user impersonation, and webcam activation. Content Security Policy(CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP is designed to be fully backward compatible; browsers that don't support it still work with servers that implement it, and vice-versa. Browsers that don't support CSP simply ignore it, functioning as usual, defaultingto the standard same-origin policy for web content. If the site doesn't offer the CSP header, browsers likewise use the standard same-origin policy.

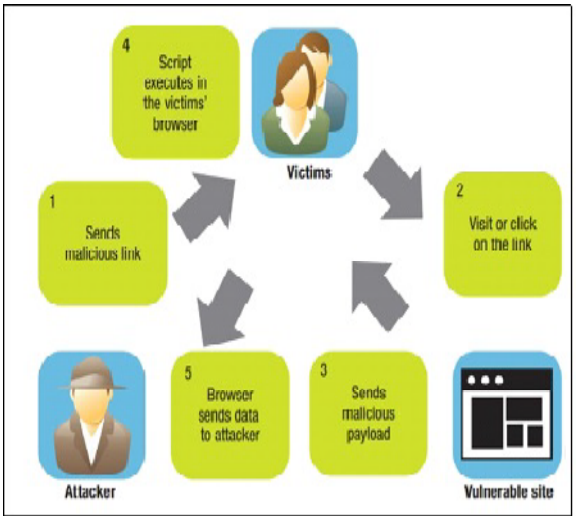


Fig.2: Existing System

B) PROPOSED SYSTEM

A client-side tool that acts as a Web proxy, disallows requests that do not belong to the website and thus thwarts stored XSS attacks. Browser-enforced embedded policies (BEEPs) let the Web application developer embed a policy in the website by specifying which scripts are allowed to run. With a BEEP, the developer can put genuine source scripts in a white list and disable source scripts in certain website regions. Document Structure Integrity (DSI) is a client-server architecture that restricts the interpretation of entrusted content. DSI uses parser-level isolation to isolate inline entrusted data and separates dynamic content from static content. However, this approach requires both servers and clients to cooperatively upgrade to enable protection.

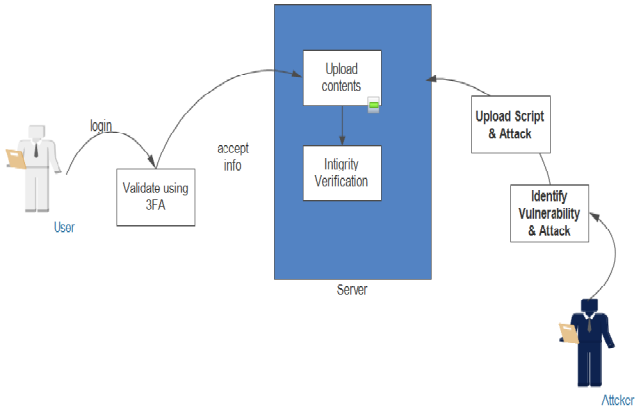


Fig.3: Block Diagram proposed System

C) SCOPE OF PROPOSEDWORK

1) Persistent XSS

A persistent XSS attack does not need a malicious link for successful exploitation; simply visiting the webpage will compromise the user. Persistent XSS is often difficult to detect and is considered more harmful than the other two attack types. Because the malicious script is rendered automatically, there is no need to target individual victims or lure them to a third party website. Consequently, attackers can easily hide their activity; for example, in a blog, they could embed the script in a seemingly innocuous comment. All visitors to that site would then unknowingly put their brow strand the sensitive data stored on it at risk.

2) Non-persistent XSS

A non-persistent, or reflected, XSS attack, which occurs when a website or Web application passes invalid user inputs. Usually, an attacker hides malicious script in the URL, disguising it as user input, and lures victims by sending emails that prompt users to click on the crafted URL. When they do,

the harmful script executes in the browser, allowing the attacker to steal authenticated cookies or data. In the figure, we assume that victims have authenticated themselves at the vulnerable site.

3) Non-persistent XSS

A webpage is composed of various elements, such as forms, paragraphs, and tables, which are represented in an object hierarchy. To update the structure and style of webpage content dynamically, all Web applications and websites interact with the DOM, a virtual map that enables access to these webpage elements. Compromising a DOM will cause the client-side code to execute in an unexpected manner. ADOM-based, or Type-0, XSS attack executes in the same manner as a non-persistent XSS attack except for step 3.In a DOM-based attack, rather than having the server carry the malicious payload in its HTTP response, the attacker encodes a malicious value in a URL and sends it to the victim. The attack occurs when the victim's browser executes the malicious code from the modified DOM. On the client side, the HTTP response does not change but the script executes maliciously. This exploit works only if the browser does not modify the URL characters. A DOM-based XSS attack is the most advanced type and is not well known. Indeed, much of the vulnerability to this attack type stems from the inability of Web application developers to fully understand how it works.

IV. DETAILED DESIGN DOCUMENT

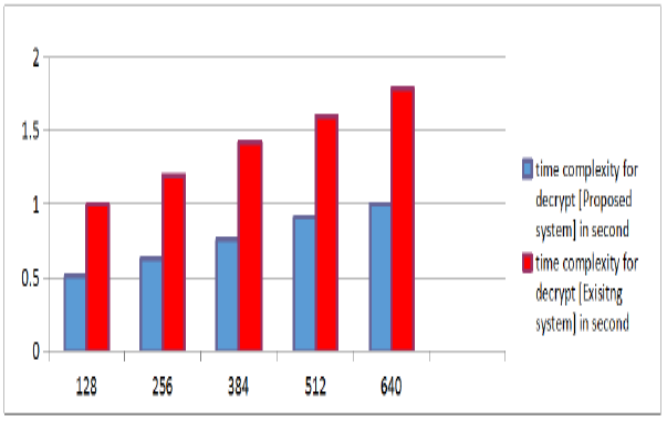
To avoid the web application attacks, the web browser security model is built on the same origin policy that isolates one origin from the other thus providing the developers a safe sandbox environment to build these applications in which thecode from one origin (http://self.com) has access to only http://self.com data and the code from other origin (http://other.com) is not permitted to access http://self.com data. But the attackers by-pass this policy by exploiting cross site scripting vulnerabilities in the web applications. He injects his own script into the web applications and later this injected script will get embedded along with the actual intended response from the website whenever any user visits that particular web page. The victims browser executes all of the code that shows up on a page as being legitimately part of that pages security origin since the browser is not able to differentiate between the injected and the intended code. Thus, Cross-Site Scripting attack (XSS) is a code injection attack performed to exploit the vulnerabilities existing in the web applications by injecting html tag / JavaScript functions into the web page so that it gets executed on the victims browser when one visits the web page and successfully accesses to any sensitive victims browser resource associated to the web application (e.g. cookies, session IDs, etc.). Successful cross site scripting can result in serious security violations for both the web site and the user. Web Applications have become one of the most important ways to provide a broad range of services to users. In the recent years, web-based attacks have caused harm to the users of web applications. Most of these attacks occur through the exploitation of security vulnerabilities in the web-based programs. So, the mitigation of these attacks is very crucial toreduce its harmful consequence. The main issue is that if malicious content can be introduced into a dynamic web page, neither the web site nor the client is capable of recognizing that anything like this happened and prevent it.

Cross Site Scripting allows an attacker to embed malicious scripts into a dynamic web page which can be vulnerable and can result in hijacking of user sessions, defacing web sites, or redirecting the user to malicious sites. A high level view of typical XSS attack is Depending on the ways HTML pages reference user inputs, XSS attacks can be classified as reflected, stored, or DOM-based. Content Security Policy (CSP) is a browser security mechanism to allow developers and server administrators to white list the locations from which applications can load the resources. By default, CSP is disabled in the browsers. To enable CSP in web applications, site developer must define an HTTP header representing the trusted locations or sites from which various sources can be downloaded. Any page served with this header will have its own security policy enforced by the browser loading it, provided that the browser supports CSP.

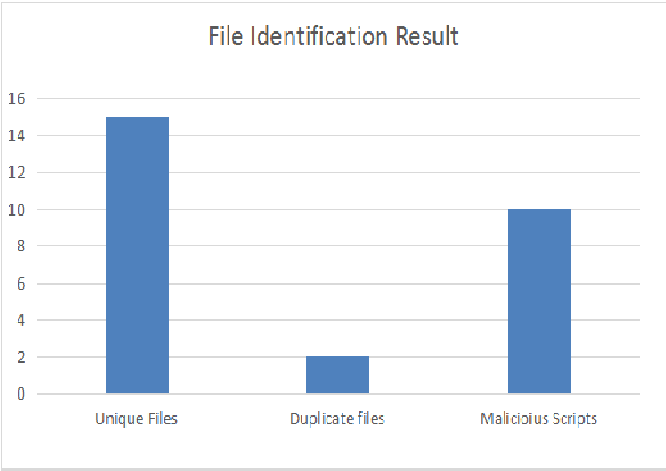
Cross-site scripting (XSS) is a computer security vulnerability that found in web applications that insert input from the user into the dynamic pages sent back to the user's browser without filtering special characters largely used in software programming. In such cases, malicious code called scripts can be inserted into the dynamic page of a targeted site. The malicious script runs in the user's browser as if the script came from the trusted Web site. As a result, the attacker gains access to the same information and privileges as the user on the targeted site. For this attack to occur, the attacker targets a Web site and the people who visit it. Most of attackers initiate the attack by enticing a web user to click on a hyperlink. The hyperlink contains a request for the targeted Web site and the malicious script. The targeted Web site sends the users browser a dynamic page in response to the request that includes the malicious script. The malicious script can read, alter, and send any sensitive information accessible to the user's browser. The attacker gains access to private user information. XSS attack exploits the trust a user has in a particular Web site. E-commerce sites are candidates for XSS attacks.

V. RESULT

Differentiate output results of enc-dec (Base 64, Hexadecimal) results are given in Fig. for Existing and Proposed System, Fig. shows the results at base 64 encoding while It gives the results of hexadecimal base encoding. We can notice that there is significant difference at both system.



The primary execution measurements used to Files in the proposed systems are shown the files found in malicious content, unique cases and duplicate cases while measures the content of file. It gives the calculation time at the server and the customer, still on the grounds that the time required for exchange of last and transitional results in the middle of user and server.



V. CONCLUSION

Through performing the above experiments, it has been concluded that CSP can mitigate XSS assaults. However the difficulty in imposing CSP within the internet sites is observed to be browser compatibility disorders as there is no one single typical CSP header that may be outlined for all of the browsers. Also, the website developer does now not comprehend about the browser where customer is going to open the net web page of his internet site. The consumer can use chrome, IE, safari, firefox or any different browsers. If the developer use header (content material-safety-coverage:default-src self), then XSS attack can be mitigated in case of Chrome, Opera and Firefox; but code injection can also be carried out in Safari, IE and other browsers. In a similar fashion, if the developer use header(X-content material-protection-policy: sandbox ..) , then XSS attack may also be mitigated in case of most effective IE however code injection can also be applied in different browsers. As a result, to resolve this incompatibility, a code is written in personal home page that's to be deployed in commencing of the webpage where CSP is to be implemented.

ACKNOWLEDGMENT

At the outset, I thank the Lord Almighty for the grace, strength and hope to make my endeavour a success. I also express my gratitude to Prof. D. S. Kulkarni , Head of the Department and my project Guide for providing me with adequate facilities, ways and means by which I was able to complete this Paper. I express my sincere gratitude to him for his constant support and valuable suggestions without which the successful completion of this project would not have been possible. I thank Prof. Anupkumar Bhongale, of PG Coordinator for his boundless cooperation and helps extended for this Paper. I express my immense pleasure and thankfulness to all the teachers and staff of the Department of Computer Engineering for their cooperation and support. Last but not the least, I thank to IJSART convener and especially my family members who in one way or another helped me in the successful completion of this work. I wish you the best of success. Thank You

REFERENCES

[1] www.html5rocks.com/en/tutorials/Security/Content-Security- Policy/.
[2] Content-Security-Policy.com
[3] Sid Stamm et. al., Reining in the Web with Content Security Policy, Proceedings of the 19th international conference on WWW , ACM, USA :921-930.
[4] Haneet kour and LalitSen Sharma, Tracing out cross site scripting vulnerabilities in modern scripts, IJANA Vol7 Issue5,2016: 2862-2867.
[5] Isatou Hydara and et al.Current state of research on cross-site scripting (XSS) A systematic literature review, ELSEVIER Information and Software Technology 58 (170186) 2015.
[6] Amit Singh and S Sathappan, A Survey on XSS web-attack and Defense Mech- anisms IJARCSSE, Vol 3 issue 4, March 2014 .

[7] Z. Mao, N. Li, and I. Molloy, Defeating cross-site request forgery attacks with browser-enforced authenticity protection, In Financial Cryptography and Data Security: 13th International Conference, FC 2009, Accra Beach, Barbados, February 23-26, 2009. Revised Selected Papers, Springer-Verlag, Berlin, Heidelberg, 2009.238-255.

[8] M. T. Louw and V N. Venkatakrishnan, Blueprint: Robust Prevention of Cross- Site Scripting Attacks for existtng browser. Proc. 30th IEEE Symp Security and Privacy (SP 09), IEEE CS, 2009: 331-346.

[9] O.Hallaraker and G.Vigna, Detecting Malicious JavaScript Code in Mozilla, In Proceedings of the IEEE International Conference on Engineering of Complex Com- puter Systems, 2005.

[10] YH. J. Wang, X. Fan, J. Howell, and C. Jackson, Protection and communication abstractions for web browsers in mashupos, In SOSP 07: Proceedings of twenty-first ACM SIGOPS symposium on Operatingsystems principles, New York, NY, USA, 2007:1-16.

[11] C. Reis, S. D. Gribble, and H. M. Levy, Architectural principles for safe web programs, In Sixth Workshop on Hot Topics in Networks (HotNets) 2007, Atlanta, Georgia, November 2007.

[12] CJ. Burke. Jsonrequest, part 2 (cross domain policy for all). Blog,(http://tagneto.blogspot.com/2 March 2006.

[13] S.Shalini and S.Usha, Prevention of XSS attacks on web applications in the client side, IJCSI, Vol. 8, Issue 4, No1, July 2011.

[14] Engin Kirdaa, Nenad Jovanovicb, Christopher Kruegelc, Giovanni Vignac,Client- side cross-site scripting protection, ELSEVIER, Computers security 28, 2009: 592-604.

[15] SubvertingAjax StefanoDiPaola, GiorgioFedon http://events.ccc.de/congress/2006/Fahrplan/att 1158 - Subverting Ajax.pdf.

[16] Jake Meredith, Content security policy best practices,https://www.isecparteners.com, July 14, 2013.

[17] Kailas Patil and Braun Frederik, A Measurement Study of the Content Security Policy on Real-World Applications, International Journal of Network Security, Vol.18, No.2, Mar.2016: 383-392.