International Journal of Security and Its Applications
Vol. 9, No. 3 (2015), pp. 139-152
http://dx.doi.org/10.14257/ijsia.2015.9.3.14

A Survey on Detection and Prevention of Cross-Site Scripting Attack

V. Nithya[1], S. Lakshmana Pandian[2] and C. Malarvizhi[3]

[1,3]*University College of Engineering, Thirukkuvalai, Anna University, India*
[2]*Pondicherry Engineering College, Puducherry, India*
[1]*nithyahimalini@ gmail.com,* [2]*lpandian72@pec.edu,* [3]*malar89.pdy@gmail.com*

Abstract

In present-day time, securing the web application against hacking is a big challenge. One of the common types of hacking technique to attack the web application is Cross-Site Scripting (XSS). Cross-Site Scripting (XSS) vulnerabilities are being exploited by the attackers to steal web browser's resources such as cookies, credentials etc. by injecting the malicious JavaScript code on the victim's web applications. Since Web browsers support the execution of commands embedded in Web pages to enable dynamic Web pages attackers can make use of this feature to enforce the execution of malicious code in a user's Web browser. The analysis of detection and prevention of Cross-Site Scripting (XSS) help to avoid this type of attack. We describe a technique to detect and prevent this kind of manipulation and hence eliminate Cross-Site Scripting attack.

Keywords: *Cross-Site Scripting attack, prevention, detection, Web Application*

1. Introduction

The World Wide Web (WWW), refers to as the web platform, has evolved into a large-scale system composed by millions of applications and services. In the beginning, there were only static web pages which provide static information expressed in text and graphics. As the Internet is growing, the web sites become more professional and dynamic. Now web applications are used to generate dynamic web pages and become the dominant method for implementing and providing access to on-line services and becoming truly pervasive in all kinds of business models and organizations. Today, most systems such as Social Networks, health care, blogs, banking, or even emergency response, are relying on these applications. Users can use web applications for communicating with other users via instant messaging, for reading email and for managing their photographs and other files, for editing and viewing video or even creating spreadsheets, presentations and text documents. Therefore, Internet is becoming an integral part our daily life. So there must include, an addition mechanisms to ensure security for internet users. Therefore providing a beneficial and safe networking environment is necessary. If vulnerability occurs in famous websites, a lot of visitors will be attacked and the result cannot be imagined. Cross-site scripting (XSS) attack is one of the most common vulnerabilities in web applications.

Cross-site scripting (XSS) attack considered as one of the top 10 web application vulnerabilities of 2013 by the Open Web Application Security Project (OWASP) [43]. According Cenzic Application Vulnerability Trends Report (2013) Cross Site Scripting represents 26% of the total population respectively [41] and considers as top most first attack. Two recent incidents highlighted the severity of xss vulnerability are Apple Developer Site (July 18, 2013) and Ubuntu Forums (July 14 and July 20, 2013) [44]. Cross Site Scripting attack carried out using HTML, JavaScript, VBScript, ActiveX, Flash, and other client-side

ISSN: 1738-9976 IJSIA

languages. A weak input validation on the web application leads Cross Site Scripting attacks to gather data from account hijacking, changing of user settings, cookie theft. Detection or prevention of XSS is a topic of active research in the industry and academia. To achieve those purposes, automatic tools and security system have been implemented, but none of them are complete or accurate enough to guarantee an absolute level of security on web application. One of the important reasons of this shortcoming is that there is lack of common and complete methodology for the evaluation either in terms of performance or needed source code modification which in an overhead for an existing system. A mechanism which will easily deployable and provide a good performance to detect and prevent the Cross-site scripting (XSS) attack is essential one.

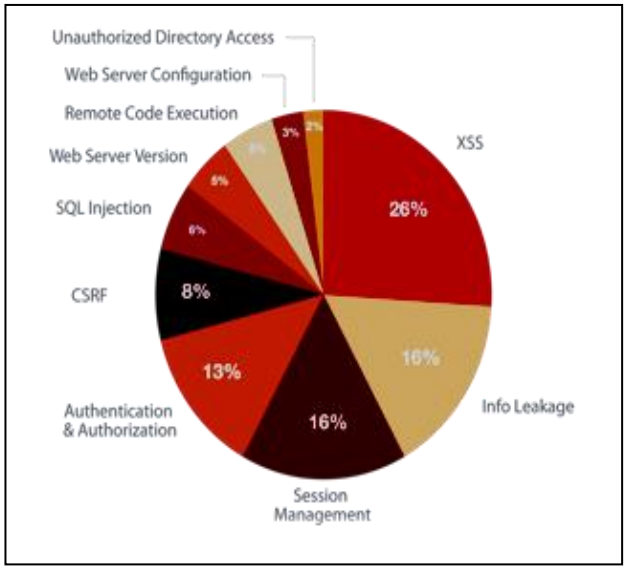

Figure 1.1. Web Application Security Vulnerability Population (2013)

2. Overview of Cross-Site Scripting Attack

In general, cross-site scripting refers to that hacking technique that leverages vulnerabilities in the code of a web application to allow an attacker to send malicious content from an end-user and collect some type of data from the victim. XSS can be defined as a security exploit in which an attacker can embed malicious script (JavaScript, VBScript, ActiveX, HTML, or Flash) into a vulnerable dynamic page, by executing the script on his machine in order to gather data. The use of XSS might compromise private information, manipulate or steal cookies, create requests that can be mistaken for those of a valid user, or execute malicious code on the end-user systems. The data is usually formatted as a hyperlink containing malicious content and which is distributed over any possible means on the internet. We study in this section three main types of XSS attacks: persistent, non- persistent XSS attacks and DOM based attack.

2.1. Threats of XSS

Cross-site scripting poses severe application risks that include, but are not limited to, the following:

1. **Session hijacking:** such as adding JavaScript that forwards cookies to an attacker.
2. **Misinformation:** such as adding "For more info call 1-800-A-BADGUY"to a page".

3. **Defacing web site:** such as adding "This Company is terrible" to a page.
4. **Inserting hostile content:** such as adding malicious ActiveX controls to a page.
5. **Phishing attacks:** such as adding login FORM posts to third party sites.
6. **Takeover of the user's browser:** such as adding JavaScript code to redirect the user.
7. **Pop-Up-Flooding:** Malicious scripts can make your website inaccessible also can make browsers crash or become inoperable.
8. Scripts can spy on what you do such as History of sites visited and Track information you posted to a web site and Access to personal data such as (Credit card, Bank Account)
9. **Access to business data:** such as (Bid details, construction details)

2.2. Vulnerabilities Associated With XSS for Web Applications

Cross-Site Scripting poses several application Vulnerabilities that include, but are not limited to, the following:

1. Users unknowingly execute malicious scripts when viewing dynamic ally generated pages based on content provided by an attacker.
2. Attacker takes over the user session before the user's session cookie expires.
3. Attacker makes the users to connect to a malicious server to his/ her choice.
4. An attacker convinces a user to access a URL supplied, which could cause script or HTML of the attacker's choice to be executed in the user's browser. Using this technique, attacker takes actions with the privileges of the user who accessed the URL, such as issuing queries on the under lying SQL databases and viewing theresults and to exploit the known faulty implementations on the target system.
5. Secure Socket Layer Encrypted connections may be exposed: The malicious script tags are introduced before encrypted connection is established between the client and the legitimate server. Secure Socket Layer encrypts data sent over the connection, including the malicious code, which is passed in both directions which assures that the client and server are communicating without snooping; SSL does not attempt to validate the legitimacy of data transmitted. Because there is a legitimate dialog between the client and the server, SSL reports no problems. The malicious code attempts to connect to a non-SSL URL that may generate warning messages about the insecure connection, but the attacker can circumvent this warning simply by running an SSL-capable web server.
6. For making the attacks to be persistent through the poisoned Cookies, the code from authentic web sites once found to have malicious code cookies will be modified. For the dynamic generation of pages if any field from the cookie is used by the vulnerable website then the attacker can include the malicious code in specific cookie by modifying it.
7. An attacker is able to execute script code on the client machine that exposes data from a vulnerable server inside the client's intranet by constructing a malicious URL. If the compromised client has cached authentication for the targeted server, the attacker may gain unauthorized web access to an intranet web server.
8. Instead of acting as any particular system an attacker only needs to identify a vulnerable intranet server and convince the user to visit an innocent looking page to expose potentially sensitive data on the intranet server.
9. Even if user's browser is restricting execution of scripting languages from some hosts or domains, attackers are able to violate this policy. This can be done by including malicious script tags in a request sent to a server that is allowed to execute scripts.

10. If there is no character set is specified in the page returned by the web server, browsers interpret the information they receive according to the character set chosen by the end user.
11. Some web sites fail to specify the character set which will lead to choosing alternate character set by the user at risk.
12. The behavior of forms can also be modified by the attackers under certain conditions.

2.3. Types of Cross-Site Scripting Attack

There are three distinct types of XSS attacks: non persistent, persistent, and DOM-based.

2.3.1. Persistent XSS Attack: In the persistent XSS an attacker can inject the malicious code into the page persistently and that means the code will be stored in the target servers as an html text, such as in a database, in a comment field, messages posted on forums, *etc.*, and this code will be stored in the page which will show to the user victim later on. If the user victim goes to the page which is embedded with XSS attacking code, the code will execute on the user victim's browser, which, in turn sends the user's sensitive information from his site to the attacker's site. The persistent XSS attack is also known as stored XSS attack. Compared with "REFLECTED XSS", this type of XSS does more serious harm. If the "STORED XSS" vulnerability is successfully exploited by hackers, it will persistently attack the users until administrator remove this vulnerability. The following Figure 2. 1 shows architecture of exploiting the persistent XSS attack by a malicious attacker.

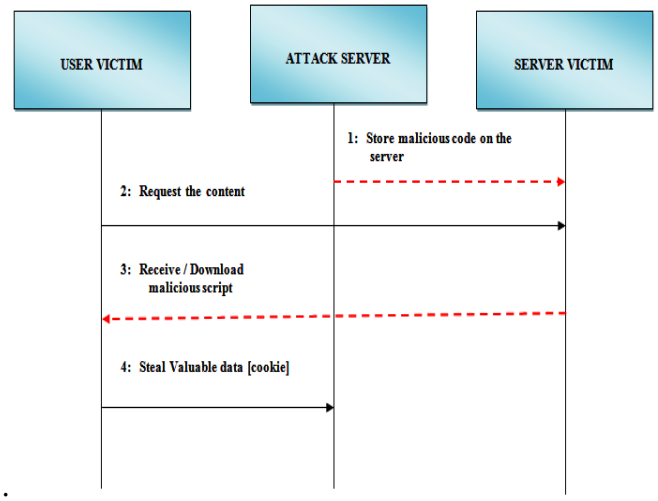

Figure 2.1. Architecture of Exploiting the Persistent XSS Attack

2.3.2. Non-persistent XSS Attack: Non-persistent cross-site scripting vulnerability is the common type of XSS attacks. The attack code is not persistently stored, but, instead, it is immediately reflected back to the user. It is also known as reflected XSS attack. In this type the injected code is sent back to the user victim off the server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request. To do this, the attacker sends a link to the user victim (*e.g.*, by email). If the user victim clicks on the link, the vulnerable web application displays the requested web page with the information passed to it in this link. This information contains the malicious code which is now part of the web page that is sent back to the web browser of the user,

where it is executed. The following Figure 2.2 is an architecture which shows the sequence of steps of exploiting the reflected XSS vulnerability by a malicious attacker.

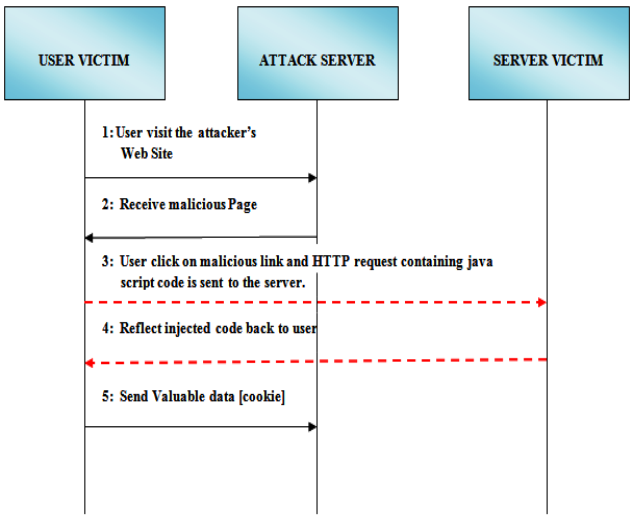

Figure 2.2. Architecture of Exploiting the Non-persistent XSS Attack

2.3.3. DOM-based XSS: DOM-based cross-site scripting attacks are performed by modifying the DOM "environment" in the client side instead of sending any malicious code to server. DOM is short for Document Object Model and it is a platform and language neutral interface. DOM allows the scripting to change the HTML or XML document, the HTML or XML document can be modified by a hacker's scripting or program. Therefore, DOM-based XSS uses DOMs vulnerability to make the XSS come true. This type of XSS vulnerability is totally different from the reflected or stored XSS attack and it does not inject malicious code into a page. So, it is the problem of the insecure DOM object which can be controlled by the client side in the web page or application. For this reason, hackers can let the attack payload execute in the DOM environment to attack the Victim side. The following Figure 2.3 is an architecture which shows the sequence of steps of exploiting the reflected XSS vulnerability by a malicious attacker.

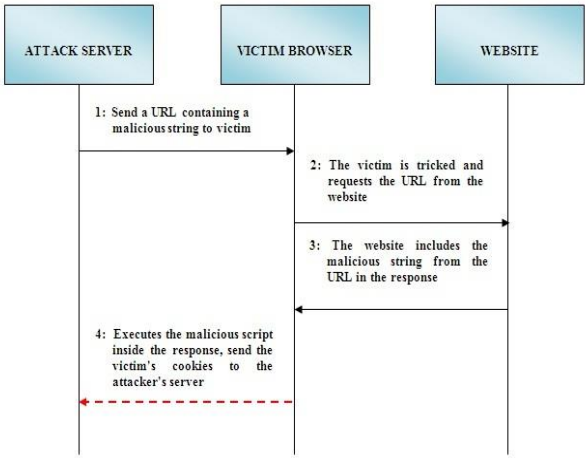

Figure 2.3. Architecture of Exploiting the DOM-based XSS Attack

3. Related Work

There are several works that have been proposed and implemented in the past by various researchers to secure web application from cross site scripting attacks. Some of the most important works are cited below.

3.1. Cross-site Scripting (XSS) Attack Detection

In order to detect Cross-site scripting (XSS) attack, filtering and other detection methods are being researched. This section explains the related work.

3.1.1. Static Analysis Approach:

Analysis of String: A.S. Christensen, A. Møller, and M.I. Schwartzbach suggested the study of static string analysis for imperative languages. They have shown usefulness of string analysis for analyzing reflective code in Java programs and checking for errors in dynamically generated SQL queries [1]. They used finite state automata as a target language representation to analyze Java. They chose FSAs because FSAs are closed under the standard language operations. Methods from computational linguistics were also applied to generate good Finite State Automata approximation of CFGs [2]. This approach is not so efficient than the other string analyzes because the source of data was not tracked and it must also determine Finite State Automata between each operations, so not a practical approach to find XSS vulnerabilities. The same approach has been followed by Y. Minamide to design a string analysis for PHP which is not approximating CFGs to Finite State Automata. This technique checks the presence of "<script>" tag in the whole document [3]. As web applications more often have their own scripts and also there are several other ways to invoke a JavaScript interpreter the approach is not at all practical to find XSS vulnerabilities.

Bounded Model Checking: Y.W Huang, F. Yu, C. Hang, C. H. Tsai (2004) use counterexample traces to minimize the number of sanitization routines inserted and to identify the reason of errors that enhance the precision of both code instrumentation and error reports [4]. Variables representing current trust level were assigned states which further were used in verifying the legal information flow in a web application. Now in order to verify the correctness of all safety states of program Abstract Interpretation, Bounded Model Checking technique was used. Some of the major problems in their approach were to leave out alias analysis or include file resolution issues.

Software Testing Approaches: Y. Huang, S. Huang, Lin, and Tsai use number of software-testing techniques such as black-box testing, fault injection, and behavior monitoring to web application in order to deduce the presence of vulnerabilities [5]. This approach combines user experience modeling as black-box testing and user-behavior simulation [4]. There are many other projects where a similar kind of approach has been followed like WebInspect [6], APPScan [7] and ScanDo [8]. Since, these approaches are applied to identify errors in development cycle, so these may unable to provide instant Web application protection [9] and they cannot guarantee the detection of all flaws as well [10].

Taint Propagation Approach: This kind of analysis is being used by many static and dynamic approaches, they use data flow analysis to track movement of information flow from source to sink. [11, 9, 12, 12, 14] There are some assumptions in this approach: The application is secure if a sanitization operation is performed from source to sink in the in all

the paths. [15]. It is not considered a good idea to have faith on user's filter and not to check the sanitization function because there are some XSS vectors which can easily bypass many filters considered to be strong. Thus it doesn't provide strong security mechanism here [16].

Using Untrusted Scripts: Using a list of untrusted scripts to detect harmful script from user given data is used to detect harmful scripts. Wassermann and Su's recent approach in [17] is shadow of this technique. The method followed here was building policies and generating regular expressions of untrusted tags and then checking if there is an intersection between CFG, generated from String taint static analysis and regular expression generated. If the result was positive then further actions had to be taken. It is believed that using untrusted script is easy and but poor idea. The same has been stated in the document of OWASP [43]. It is clearly mentioned in the document not to use "blacklist" validation to detect XSS in input or to encode output. Search and replacement of a few characters ("<", ">") which are considered as bad is weak and has been attacked successfully. There are a number of different kinds of XSS which can easily bypass blacklist validation.

3.1.2. Dynamic Analysis Approach:

Browser-Enforced Embedded Policies Approach: A white list of all benign scripts is given by the web application to browser to protect from malicious code [18]. This was a good idea which allows only the scripts in the provided list to run; however, since there is a difference in the parsing mechanism of different browsers a successful filtering system of one browser may not be successful for other. Hence, although the technique explained in this paper is quite successful against these kinds of situations but a modification is required to be done in all the browsers to enforce the policy. So, it suffers for scalability problem from the web application's point of view [19]. Every client needs to have this modified version of browser on their system.

Syntactical Structure Approach: Su and Wassermann in [20] suggested an approach which states that when there is a successful injection attack there is a change in the syntactical structure of the exploited entity. They present an approach to check the syntactic structure of output string to detect malicious payload. Augment the user input with metadata to track this sub-string from source to sinks. This metadata help the modified parser to check the syntactical structure of the dynamically generated string by indicating end and start position of the user given data. Moreover the process was blocked if there was a sign of any abnormality. This approach was found to be quite successful while it detects any injection vulnerabilities other than XSS only checking the syntactic structure is not sufficient to prevent this sort of workflow vulnerabilities that are caused by the interaction of multiple modules [21].

Proxy-based Approach: Noxes, a web proxy protects against transferring of sensitive information from victim's site to third party's site [22]. This is an application-level firewall to block and detect malware. There is a fine grained control provided to users on every connection coming or leaving the local machine. If any connection is mismatched with the firewall's rules then firewall prompts the user to decide whether the connection needs to be blocked or allowed. Similar kind of approaches has been followed in [23, 46, 24]. It is not sufficient to Blacklist a link to prevent cross-site Scripting attacks. Huang et al. suggested, proxy-based approach does not provide a method to find errors, moreover it requires a watchful configuration. This approach could definitely increase false positive as they protect the unpredictable link with no examination of the fault.

Interpreter-based Approaches: This approach has been suggested by Pietraszek, and Berghe in which there is use of instrumenting interpreter to track untrusted data at the character level and for identifying vulnerabilities that use context-sensitive string evaluation at each susceptible sink [25]. This approach is sound and can detect vulnerabilities as they add security assurance by modifying the interpreter. But approach of modifying interpreter is not easily applicable to some other web programming languages, such as Java, Jsp, Servlets [26].

3.1.3. Static and Dynamic Analysis Approach

Lattice-based Approach: There is a tool called WebSSARI which combines static and runtime features and find security vulnerabilities by applying static taint propagation analysis. WebSSARI follows type state and lattice model and uses flow sensitive, intra-procedural approach to determine vulnerability. When this tool knows that tainted data has reached sensitive function, it automatically puts runtime guards which are also called as sanitization routines [27]. There is a big drawback with this technique that it gives a large number of false negative and positive because of its intra-procedural type-based analysis [28]. Furthermore this approach also takes results from users' designed filters as safe. Hence, the real vulnerabilities might be missed, as it is quite possible that malicious payload may not be detected by designated filtering function.

3.2. Cross-site Scripting (XSS) Attack Prevention

There are main criteria that can be used for distinguishing between XSS prevention techniques is at point of deployment server-side or client-side and both.

3.2.1. Server Side Solution: The cross site scripting vulnerabilities in Web applications, a number of testing tools has been proposed earlier. There are two types of testing tool such as, Black-box [45] Web application testing tools as well as white-box [29] vulnerability scanners have been suggested in previous research, and are successfully used in real time practice. This kind of tools can generally help in identifying cross site scripting vulnerabilities, it is likely that some remain undetected, which clearly recommends additional safeguards for web application. In [30], an application-level firewall is suggested, which is located on a security gateway between server and client, and which applies all security relevant checks and transformations. By separating the security relevant part of the code from the rest of the application, as well as providing a specialized Security Policy Description Language to design it, the system helps Web developers to apply measures against XSS in a less error prone fashion. Martin Johns et al (2008) proposed a passive detection system to identify successful XSS attacks. In their work, they have compiled a data set of 500000 individual HTTP request/response pairs from 95 popular web applications in combination with both real word and manually crafted XSS exploits. Moreover, their detection approach results in a total of zero false negatives for all tests, while maintaining an excellent false positive rate for more than 80% of the examined web applications.

Wurzinger, *et al.,* (2009) [31] introduced secure web application proxy for server side solution to detect and prevent cross site scripting attacks. Their system comprises a reverse proxy that intercepts all HTML responses, as well as a modified Web browser which is utilized to detect script content. In another work, Shahriar and Zulkernine (2009) applied the idea of mutation based testing technique to generate adequate test data sets for testing XSS vulnerabilities. They addressed XSS vulnerabilities related to web applications that use Hypertext Preprocessor (PHP) and JavaScript code to generate dynamic HTML contents and

their mutants. Their approach also includes a robust mechanism for identifying scripts at the server side and removes any script in the output that is not intended by the web application.

The defense mechanisms for cross site scripting attacks was proposed by Prithvi Bisht and Venkatakrishanan (2008) based on input validation that can effectively prevent XSS attacks on the server side. They also discussed several new XSS attacks and analyzed the reasons for the failure of filtering mechanisms in defending these attacks. They also proposed a new framework called XSS GUARD that has been designed to prevent XSS attacks on the server side. XSS GUARD works by dynamically learning the set of scripts that a web application intends to create for any HTML request.

Prithvi Bisht, *et al.,* (2010) proposed approach to automatically detect potential server side vulnerabilities in web applications through black box analysis. Their approach has been used to test and discover several previously unknown vulnerabilities in a number of open source web applications and live web sites. Apart from this, the researchers developed the WebSSARI (Web Security via Static Analysis and Runtime Inspection) tool [33], which performs type-based static analysis to identify potentially vulnerable code sections and instrument them with runtime guards. Scott and Sharp [30] describe a web proxy that is located between the users and the web application, and that makes sure that a web application adheres to pre written security policies. The main categories of such policy based approaches are that the creation and management of security policies is a tedious and error-prone task. Similar to [30], there exists a commercial product called AppShield, which is a web application firewall proxy that apparently does not need security policies. Furthermore [30], reports that AppShield is a plug and play application that can only do simple checks and thus, can only provide limited protection because of the lack of any security policies. A well-known, dynamic server side prevention mechanism is Perl's taint mode [33]. In this method, the flow of tainted values is tracked within the Perl interpreter. The input from untrusted sources is marked as being potentially malicious, and propagated through the program. Any attempt to use tainted data directly or indirectly in a critical command that invokes a subshell, modifies files, directories, or processes will be aborted with an error. The developer is given means to test the taint status of data, as well as the ability to sanitize the data where this seems appropriate.

3.2.2. Client-side Solution: Complementary to mitigating XSS on the server-side, there are several client-side solutions. In [34], a strictly client-side mechanism for detecting malicious Java Scripts is proposed. The system consists of a browser-embedded script auditing component, and IDS that processes the audit logs and compares them to signatures of known malicious behavior or attacks. With this system, it is possible to detect various kinds of malicious scripts, not only XSS attacks. However, for each type of attack a signature must be crafted, meaning that the system is defeated by original attacks not anticipated by the signature authors. Client side cross site scripting protection (Noxes Tool) [35] is a client-side Web-proxy that relays all Web traffic and serves as an application-level firewall. The approach works without attack-specific signatures. However, as opposed to SWAP [36], Noxes requires user-specific configuration (firewall rules), as well as user interaction when a suspicious event occurs. Huang describe the use of a number of software-testing techniques and suggest mechanisms for applying these techniques to web applications. The main aim is to cover and fix web vulnerabilities such as XSS. Philipp Vogt, *et al.,* (2007) proposed a solution that prevents XSS attacks on the client side by tracking the flow of sensitive information inside the web browser. Web Static Approximation [20] uses a static string analysis technique to approximate possible string output for variables in a web application and checks if the approximated string output is disjoint with unsafe strings defined in a

specification file. If the approximate string output is disjoint with the unsafe strings, Web Static Approximation reports that the application is not vulnerable. Stephen Chong et al (2007, 2009a) developed a compiler that uses policies to automatically partition the program into JavaScript code running in the browser, and Java code running on the server. Their system protects the code and data that are placed on the client side. Moreover, this code and data can be replicated across the client and server to provide both security and performance. Moreover, they also proposed a method to build secure web application via automatic portioning. Mike Ter Louw and Venkatakrishnan (2009) presented a new XSS defense strategy that is widely deployed in existing web browsers for providing effective security. This approach relaxes the trust placed on browsers for interpreting un-trusted content. They have evaluated blueprints against a barrage of stress tests that demonstrates strong resistance to attacks and provides excellent compatibility with web browsers with reasonable performance overheads. Collin Jackson, *et al.,* (2009) proposed and implemented a solution for protecting browsers from Domain Name System (DNS) rebinding attacks. Their primary focus is in the design of strong defenses against DNS rebinding attacks that protect modern browsers. Nuo Li, *et al.,* (2010) proposed a new approach to generate test inputs for User Input Vectors (UIV) based on the analysis of client side information. They used input field information to generate valid inputs and then perturb valid inputs to generate invalid test inputs. They conducted an empirical study which results in comparison to existing vulnerability scanners. Their approach is more effective than the existing vulnerability scanners in finding semantics related vulnerabilities of UIV in web applications.

3.2.3. Hybrid Mitigation Approaches: Some solutions apply hybrid approaches, which also involve the Web browser. The server annotates the delivered content and provides information on the legitimacy or level of privileges of scripts. The Web browser is then responsible for checking and enforcing these annotations. BEEP (Browser-Enforced Embedded Policies) [37] proposes to use a modified browser that hooks all script execution attempts, and checks them against a policy, which must be provided by the server. Two kinds of policies are suggested. First, using a white list of the hashes allowed the scripts, which the browser can check against. Second, labeling those nodes in the HTML source, which are supposed to contain user-provided content, so the browser can determine whether a scripts position in the DOM tree is within user-provided content. The modified browser verifies each script with respect to the policy and prohibits scripts from execution that do not comply. Wes Masri and Andy Podgurski have stated [38] that information flow based work will increase the false positives and it is not an indicative strength if the information flow is high. There are validation mechanisms [39] and scanners proposed to prevent XSS vulnerabilities [40]. Some software engineering approaches are also proposed such as WAVES for security assessment. However none of the solutions are not built for the latest developments and would fail if tags are permitted in the web applications. Jayamsakthi Shanmugam. [40] provided solutions based on financial and non financial applications but this does not cater for the XSS attacks emerge from various interfaces. In Nonce spaces [41], the authors propose to use randomized XML namespaces in order to partition the content into different trust classes. The client is responsible for interpreting the namespaces and restricting the content's rights according to a policy that is provided alongside the Web site. The owner of the site can assign the desired trust levels via XPath expressions, and thus, disallow JavaScript code in HTML sub trees that are supposed to contain user contributed content. The mentioned hybrid mitigation techniques offer the most powerful features and the best ratio between parameterization costs and level of protection. However, they share the same drawback as the strictly client-based solutions: The requirement to being deployed on user's machines.

4. Conclusion

The use of the web paradigm is becoming an emerging strategy for application software companies. It allows the design of pervasive applications which can be potentially used by thousands of customers from simple web clients. Moreover, the existence of new technologies for the improvement of web features allows software engineers the conception of new tools which are not longer restricted to specific operating systems. Existing approaches to secure traditional applications are not always sufficient when addressing the web paradigm and often leave end users responsible for the protection of key aspects of a service. This situation must be prevented since, if not well managed, it could allow improper uses of a web application and lead to a violation of its security requirements. We focus in this paper on the types of Cross-Site Scripting attacks against the security of web applications. This attack relays on the injection of a malicious code into a web application, in order to compromise the trust relationship between a user and the web application's site. If the vulnerability is successfully exploited, then it allow attackers to execute script in the victims browser, which can hijack user sessions, deface web sites, insert hostile content, and conduct phishing attacks.. We survey in this paper the XSS attacks that can actually affect current web applications, and we discuss existing approaches for its prevention, and detection of XSS attack. We discuss these approaches and their limitations, as well as their deployment and applicability.

References

[1] A. S. Christensen, A. Møller and M. I. Schwartzbach, "Precise analysis of string expression", In proceedings of the 10th international static analysis symposium, LNCS, Springer-Verlag, vol. 2694, pp. 1-18.
[2] M. Mohri and M. Nederhof, "Regular approximation of context-free grammars through transformation", Robustness in Language and Speech Technology, **(2001)**, pp. 153–163.
[3] Y. Minamide, "Static Approximation of Dynamically Generated Web Pages", In WWW'05: Proceedings of the 14th International Conference on the World Wide Web, **(2005)**, pp. 432–441.
[4] Y. W Huang, F. Yu, C. Hang, C. H. Tsai, D. Lee and S. Y. Kuo, "Verifying Web Application using Bounded Model Checking," In Proceedings of the International Conference on Dependable Systems and Networks, **(2004)**.
[5] Y.-W. Huang, S.-K. Huang, T.-P. Lin and C.-H. Tsai, "Web application security assessment by fault injection and Behavior Monitoring," In Proceeding of the 12th international conference on World Wide Web, ACM, New York, NY, USA, **(2003)**, pp.148-159.
[6] "Web Application Security Assessment," SPI Dynamics Whitepaper, SPI Dynamics, **(2003)**.
[7] "Web Application Security Testing – AppScan 3.5", Sanctum Inc., http://www.sanctuminc.com.
[8] "InterDo Version 3.0", Kavado Whitepaper, Kavado Inc., **(2003)**.
[9] Y.-W. Huang, F. Yu, C. Hang, C. H. Tsai, D. Lee and S. Y. Kuo, "Securing web application code by static analysis and runtime protection," In Proceedings of the 13 th International World Wide Web Conference, **(2004)**.
[10] D. Scott and R. Sharp, "Abstracting Application-Level Web Security," In Proceeding 11thinternational World Wide Web Conference, Honolulu, Hawaii, **(2002)**, pp. 396-407.
[11] Y. Xie and A. Aiken, "Static detection of security vulnerabilities in scripting languages," In Proceeding of the 15thUSENIX Security Symposium, **(2006)** July, pp. 179-192.
[12] V.B. Livshits and M. S. Lam, "Finding security errors in Java programs with static analysis," In proceedings of the 14th Usenix security symposium, **(2005)** August, pp. 271-286.
[13] "JavaScript Security: Same origin," Mozilla Foundation, **(2006)** February, http://www.mozilla.org/projects/security/components/same-origin.html.
[14] N. Jovanovic, C. Kruegel and E. Kirda, "Precise alias analysis for syntactic detection of web application vulnerabilities," In ACM SIGPLAN Workshop on Programming Languages and Analysis for security, Ottowa, Canada, **(2006)** June.
[15] D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel and G. Vigna, "Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications," In IEEE symposium on Security and Privacy, **(2008)**.

[16] Z. Su and G. Wassermann, "The essence of command Injection Attacks in Web Applications," In Proceeding of the 33rd Annual Symposium on Principles of Programming Languages, USA: ACM, **(2006)** January, pp. 372-382.
[17] G. Wassermann and Z. Su, "Static detection of cross-site Scripting vulnerabilities," In Proceeding of the 30th International Conference on Software Engineering, **(2008)** May.
[18] T. Jim, N. Swamy and M. Hicks, "BEEP: Browser-Enforced Embedded Policies," In Proceedings of the 16th International World Wide Web Conference, ACM, **(2007)**, pp. 601-610.
[19] P. Bisht and V. N. Venkatakrishnan, "XSS-GUARD: Precise dynamic prevention of Cross-Site Scripting Attacks," In Proceeding of 5th Conference on Detection of Intrusions and Malware & Vulnerability Assessment, LNCS, vol. 5137, **(2008)**, pp. 23-43.
[20] Z. Su and G. Wassermann, "The essence of command Injection Attacks in Web Applications," In Proceeding of the 33rd Annual Symposium on Principles of Programming Languages, USA: ACM, **(2006)** January, pp. 372-382.
[21] D. Balzarotti, M. Cova, V. V. Felmetsger and G. Vigna, "Multi-Module Vulnerability Analysis of Web-based Applications," In proceeding of 14th ACM Conference on Computer and Communications Security,Alexandria, Virginia, USA, **(2007)** October.
[22] E. Kirda, C. Kruegel, G. Vigna and N. Jovanovic, "Noxes: A client-side solution for mitigating cross site scripting attacks," In Proceedings of the 21st ACM symposium on Applied computing, ACM, **(2006)**, pp. 330-337.
[23] N. Jovanovic, C. Kruegel and E. Kirda, "Pixy: A static analysis tool for detecting web application vulnerabilities (short paper)," In 2006 IEEE Symposium on Security and Privacy, Oakland, CA, **(2006)** May.
[24] D. Scott and R. Sharp, "Abstracting Application-Level Web Security," In Proceeding 11thinternational World Wide Web Conference, Honolulu, Hawaii, **(2002)**, pp. 396-407.
[25] T. Pietraszek and C. V. Berghe, "Defending against Injection Attacks through Context-Sensitive String Evaluation", In Proceeding of the 8th International Symposium on Recent Advance in Intrusion Detection (RAID), **(2005)** September.
[26] Z. Su and G. Wassermann, "The essence of command Injection Attacks in Web Applications," In Proceeding of the 33rd Annual Symposium on Principles of Programming Languages, USA: ACM, **(2006)** January, pp. 372-382.
[27] D. Balzarotti, M. Cova, V. V. Felmetsger and G. Vigna, "Multi-Module Vulnerability Analysis of Web-based Applications," In proceeding of 14th ACM Conference on Computer and Communications Security,Alexandria, Virginia, USA, **(2007)** October.
[28] Y. Xie and A. Aiken, "Static detection of security vulnerabilities in scripting languages," In Proceeding of the 15thUSENIX Security Symposium, **(2006)** July, pp. 179-192.
[29] V. Felmetsger, N. Jovanovic, D. Balzarotti, M. Cova, E. Kirda and C. Kruegel, "Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications," In IEEE Security and Privacy Symposium, **(2008)**.
[30] R. Sharp and D. Scott," Abstracting Application Level Web Security," In Proceedings of the 11th ACM International World Wide Web Conference (WWW 2002), **(2002)** May 7-11.
[31] P. Wurzinger, C. Platzer, C. Ludl, E. Kirda and C. Kruegel, "SWAP: Mitigating XSS Attacks using a Reverse Proxy", ICSE Workshop on Software Engineering for Secure Systems, IEEE, **(2009)**, pp. 33- 39.
[32] W. Halfond, A. Orso and P. Manolios, "WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation", IEEE Trans. Software Eng., **(2008)** January, pp. 65-81.
[33] J. Allen, "Perl Version 5.8.8 Documentation Perlsec", **(2006)**, http://perldoc.perl.org/perlsec.pdf.
[34] N. Ikemiya and N. Hanakawa, "A New Web Browser Including A Transferable Function to Ajax Codes", In Proceedings of 21st IEEE/ACM International Conference on Automated Software Engineering (ASE '06), Tokyo, Japan, **(2006)** September, pp. 351-352.
[35] E. Kirda, N. Jovanovic, C. Kruegel and G. Vigna, "Client-Side Cross-Site Scripting Protection," ScienceDirect Trans.computer and security, **(2009)**, pp. 184-197.
[36] P. Wurzinger, C. Platzer, C. Ludl and C. Kruegel,"SWAP:Mitigating XSS Attacks using a Reverse Proxy," In proceedings of the 2009 ICSE Workshop on Software Engineering for secure systems, **(2009)**, pp. 33-39.
[37] T. Jim, N. Swamy and M. Hicks, "BEEP: Browser- Enforced Embedded Policies," In 16th International World Wide Web Conference (WWW2007), Banff, **(2007)**.
[38] S.-Y. Kuo, Y.-W. Huang, C.-H. Tsai and D. T. Lee, "Non Detrimental Web Application Security Scanning", In Proceedings of 15th International Symposium on Software Reliability Engineering (ISSRE'04), France, **(2004)** November, pp. 219-230.
[39] Chung-Hung, T. Y.-W. Huang, S.-K. Huang and T.-P. Lin, "Web Application Security Assessment By Fault Injection and Behavior Monitoring", In Proceedings of the 12th international conference on World Wide Web, Budapest, Hungary, **(2003)** May, pp. 148–159.

[40] M. Ponnavaikko and J. Shanmugam, "A Solution to Block Cross Site Scripting Vulnerabilities Based on Service Oriented Architecture", In Proceedings of 6th IEEE international conference on computer and information science (ICIS 07) published by IEEE.
[41] H. Chen and M. V. Gundy, "Using randomization to enforce information flow tracking and thwart crosssite scripting attacks," In Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS), **(2009)**.
[42] http://info.cenzic.com/rs/cenzic/images/Cenzic-Application-Vulnerability-Trends-Report-2013.pdf.
[43] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.
[44] https://isis.poly.edu/~jcappos/papers/tr-cse-2013-02.pdf.
[45] "Acunetix Acunetix Web Vulnerability Scanner", **(2008)**, http://www.acunetix.com/.
[46] **(2005)**, "AppShield," Sanctum Inc. http://sanctuminc.com.

Authors

V. Nithya received her Bachelor of Engineering in 2010 from Mailam Engineering College, Tindivanam and Master of Technology in 2012 in Computer Science Engineering (CSE) from Pondicherry Engineering College, Pondicherry, India .Currently she is working as Teaching Fellow in the department of CSE in University College of Engineering, Thirukkuvalai, Anna University, India. She has more than 3 publications in International Journals and presented papers in International conferences. Her area of interest includes Web Application Security, Information Security and database management system.

S. Lakshmana Pandian received his Bachelor of Engineering in Electrical and Electronics Engineering, from Government College of Engineering, Tirunelveli in 1993, and Master of Engineering in 1998 in Computer Science and Engineering, from Government College of Engineering, Tirunelveli and Ph.D. degree in the Department of CSE from Anna University, Chennai in 2011. He is working as Associate Professors in the department of CSE in Pondicherry Engineering College, India. He has more than 7 publications in International Journals. He has presented more than 8 papers in International conferences. He has guided both UG and PG candidate's projects. His areas of interest include Language Technology (Natural language processing, Compilers, Automata theory and computations) and Embedded Systems.

C. Malarvizhi received her bachelor's degree in electronics and communication engineering from Pondicherry university ,Pondicherry in 2010, and her master's degree in wireless communication from Pondicherry engineering college, Pondicherry in 2012.Currently she is working as teaching fellow in the department of electronics and communication engineering, university college of engineering, Thirukkuvalai (A constituent college of Anna university, Chennai, India).Her research interests include wireless security, MIMO wireless communications, and Distributed antenna system(DAS).