The Code Sanitizer: Regular Expression Based Prevention of Content Injection Attacks

Sandeep D Sukhdeve[1], Prof.(Mrs) Hemlata Channe[2]
[1]*M.E.student* ,[2]*Asst.Professor, Pune Institute of Computer Technology, Pune(India).*

***Abstract*—*We are increasingly relying on web, and performing important transactions online through it. The impact and quantity of security vulnerabilities in such applications has increased in recent years. Regular expression has become a common practice to ensure execution of trusted application code. However, its effectiveness in protecting client-side web application code has not yet been established. In this paper, we seek to study the efficacy of regular expression based approach for preventing script injection attacks. The paper proposes an efficient use of regular expressions to identify malicious payload contents. This paper analyzes important aspects in content injection attacks. The goals of this research work are two-fold: i) propose an efficient way to identify content injection attacks (XSS and SQL injection) using regular expressions, and ii) We present a Nondeterministic Finite Automata (NFA) based approach to detect content injection attacks. Our evaluation on Alexas top 500 sites and phpBB popular PHP application shows that the proposed approach effective on preventing content injection attacks on the input fields available on those websites. The proposed approach incurs an average performance overhead of 1.02%.*

Keywords—*Regular expression, Content Injection, Cross-site scripting, SQL injection, injection attacks.*

I. Introduction

NOWADAYS various transactions are performed by dynamic web applications. For example, users pay their utility bills, book air tickets by using dynamic websites to save time and money. It is crucial that user data should be kept secret. That is, confidentiality and integrity of user data must be provided by developers of the web application but unfortunately there is no such guarantee for preserving the underlying web application from various content injection attacks. Content injection attacks can breach the confidentiality property and integrity property of information in the vulnerable web application.

A. *Content Injection and Types*

Content injection attack refers to inserting malicious content into a legitimate site. There are two types of content injection attacks namely SQL injection and Cross-site scripting.

- *SQL Injection (SQLI)*: It is a prevalent attack technique that allows attackers to gain direct access to the databse of the web application and extract sensitive information from the victim web application's database [30].
- *Cross-site Scripting (XSS)*: It is a web application vulnerability that allows attackers to inject client-side script into web pages that can be viewed by other users. Attackers can use cross-site scripting vulnerability as a tool to bypass access control mechanisms such as the same-origin policy (SOP) [2], [29]. Some examples of real-world cross-site scripting attacks can be found at [5], [15], [24], [32], [33].

Script injection attacks are the most prevalent threat on the web. According to the symantec security threat report of 2015 [27], cross-site scripting is listed on number two out of top ten web application vulnerabilities. Web sites vulnerable to content injection attacks are from various category such as blogging, hosting, entertainment, shopping, sports, etc. According to the Open Web Application Security Project (OWASP) 2013 top ten web application vulnerability report [22], *injection* attack is a number one web application vulnerability and cross-site scripting listed at number three among the top ten web application vulnerabilites.

In 2011, the National Institute of Standards and Technologys National Vulnerability Database [17] reported total 289 SQL injection vulnerabilities in websites (7% of all vulnerabilities). In December 2011, security experts from SANS Institute reported a major SQL injection attack(SQLIA) that affected approximately 160,000 websites using ASP.NET, Microsofts Internet Information Services (IIS), and SQL Server frameworks [9].

Regular expressions have become a common practice to ensure the execution of trusted applications. Mainstream OSes such as Windows and Linux check the signature of application binaries against the public whitelist using regular expression before installing them [34]. Browsers enable code signature verification on its extensions using regular expressions to make sure that its code is signed by a trusted party (e.g., Mozillas AMO [14]). Although regular expressions have been successfully implemented in several application platforms, its effectiveness in protecting web applications has not yet been established. Lack of validation and incorrect sanitization of user inputs make websites vulnerable to content injection attacks. To address this problem, researchers have proposed various mechanisms ranging from simple static analysis to complex dynamic analysis. In this paper, we seek to study the efficacy of regular expression-based whitelisting for defending web applications against script injection attacks.

B. *Observations*

To evaluate effectivness of the proposed approach against dynamics of different websites, we conduct a longitudinal study on the Alexas top 500 websites (1500 web pages) and phpBB popular PHP blog web application, testing manually with depth limit of 3. We use XSS cheatsheets and SQL injection strings to test effectivness of the proposed approach.

We observed that *the aim of content injection attack to gain illegal access to user data*. The Structural Query Language Injection (SQLI) attack happens when an attacker is allowed to change the logic or syntax of a SQL query by injecting new SQL keywords. *SQL Injection attack is a class of content injection attacks that occurs when there is no input validation mechanism deployed by web devleopers in the web application.*

In cross-site scripting attack, the attackers hides malicious content into the content being delivered from the compromised site. When the resulting combined content process by the browser at the client-side, browser is unable to differentiate between inject content and website's benign content, and thus operates under the permissions granted to that system. By finding ways of injecting malicious content into web pages, an attacker can gain access to sensitive page content, and a variety of other information maintained by the browser on behalf of the user [29]. *The successful XSS attack is a result of the absence of the input validation in the web application by the developers.*

The existing techniques proposed by researchers are either not publicly available or are difficult to adopt. Lightweight and easy to adopt are desirable properties for a solution to combat content injection attacks. Developers unawareness of security mechanisms and content injection sanitization can result in data loss or corruption, lack of accountability, or denial of access. Injection can sometimes lead to complete host takeover. Therefore, it is important to provide a solution that protect web applications from SQLI and XSS attacks.

C. *Main Idea*

Use regular expression to combat against SQLI and XSS attacks. Recently, regular expressions have become more pouplar than static patterns to scan payloads. For example, the recent ruleset of Snort IDS includes about 2,000 static patterns and more than 500 regular expressions. A theoretical worst case study shows that a single regular expression of length n an be expressed as a DFA of up to $O(\sum^n)$ states whereas NFA representation would require only $O(n)$ states [10]. Therefore, *the propose solution in this paper uses NFA based regular expression technique for efficiently detection of content injection attacks.*

D. *Contributions*

In summary, this paper makes the following contributions:

1) We conduct a 2-month study on effectivness of the proposed approach on the Alexa Top 500 sites and phpBB popular PHP blog web application via regular expressions.
2) We propose an novel approach, a multi-layered sanitization approach based on regular expressions to prevent script injection attacks. The proposed approach employs a novel regular expression scheme, structural regular expression signatures which rely on

source code structure, and are secure & robust against content injection attacks.

3) Classifies state-of-the-art research performed on detection and prevention of content injection attacks.
4) Provides analysis of important aspects in content injection attacks.
5) Presented a Nondeterministic Finite Automata (NFA) based implementation to support more complex regular expressions than the previous approaches.

The rest of this paper is organized as follows: Section II explains content injection attack vectors (SQL injection and Cross-site Scripting attack) with the help of example. Section III discusses the literature survey. Section IV describes our motivation. Section V describes architecture of the proposed scheme. Section VI presents mathematical model of the system. Section VII describes implementation details and evaluation results of the proposed approach and we conclude the paper in Section VIII.

II. Background

An Example of SQL Injection Attack

Typically users are requested to provide some input data on web pages (for example, username and passwords) and web applications make a SQL query to the database based on the information received from the user. Malicious user can send crafted input to change the SQL statement structure and execute arbitrary SQL commands on the vulnerable system.

Lets consider an example of web application that accepts username and password for users in order to allow authenticate users to login to the web site. When user input is received at server side by the web application, following SQL query is created and executed by the web application to verify credentials provided by the user:

```
SELECT * FROM usertable WHERE
userID = 'Sandeep' and password =
'abc123'
```

Assume malicious users provided following crafted input in the password input box:

```
' or 1=1 --
```

The SQL query in the web application will become:

```
SELECT * FROM usertable WHERE
userID = Sandeep and password = ''
or 1=1 --'
```

The "**or 1=1**" will make the query TRUE and results in returning all the records in the "usertable" to the malicious user. The "**–**" comments out the last **'** character appended by the web application.

An Example of Cross-site Scripting Attack

Attacker can inject scripts in a web page by stored. For example, when asked for a text inputs by web page, malicious user can provide script code rather than providing text inputs. Therefore, if the developers of website are not carefully sanitizing the user inputs, an occurrence of the text $< script >$ ... $< /script >$ in the input text can be stored and later executed in the visitor's browser when it renders the script of malicious user in the page.

III. Literature Survey

A. SQL Injection Detection and Prevention Solutions

Several solutions that mitigate the risk posed by SQL Injection attacks have already been proposed [1], [6]–[8], [12]. All of these solutions have been successful in mitigating SQL Injection attacks. However, none of these solutions address the actual SQL injection attack that exist in the source code. A common way to remove SQL injection vulnerability is to separate the SQL structure from the SQL input by using prepared statements. Stephen et.al. [28] proposed a prepared statement replacement algorithm and a corresponding tool for automated fix generation.

Cristian [21] et.al. presented a hybrid approach based on the Adaptive Intelligent Intrusion Detector Agent (AIIDA-SQL) for the detection of SQL injection attacks. The AIIDA-SQL agent incorporates a Case-Based Reasoning (CBR) engine which is equipped with learning and adaptation capabilities for the classification of SQL queries and detection of malicious user requests. To carry out the tasks of attack classification and detection, the agent incorporates advanced algorithms in the reasoning cycle stages. Concretely, an innovative classification model based on a mixture of an Artificial Neuronal Network together with a Support Vector Machine is applied in the reuse stage of the CBR cycle. This allowed clasification of SQL queries.

B. Cross-site Scripting Detection and Prevention Solutions

Mozilla has a feature called signed scripts [25]. Scripts are signed when they require additional privileges, such as writing to local files, and the absence of a signature does not constrain scripts. Server-side techniques to protect against script injection attacks have been reported extensively in the literature. A systematic approach to filtering injected attacks involves partitioning trusted and untrusted content into separate channels and subjecting all untrusted content to application defined sanitization checks [20]. Su and Wassermann [26] develop a formal model for command injection attacks and apply a syntactic criterion to filter out malicious dynamic content. Applications of taint checking to server programs that generate content to ensure that untrustworthy input does not flow to vulnerable application components have also been explored [16], [31]. UserCSP [19] is a Mozilla tool that allows security savvy users to specify and enforce content security policy to protect themselves from cross-site scripting attacks. The tool automatically infers content security policies for the websites user visits and enforce them to protect users from XSS attacks. Other solutions [3], [4] need browser modifications to identify untrusted or malicious scripts from trusted scripts. JCShadow [18] proposed an approach to isolate untrusted scripts included within origin from diferent sources. It is effective and robust against malicious scripts mistakenly included by web publishers in their web applications. Whereas the proposed approach in this paper focuses on the sanitization of the content injected by malicious users through the input text fields available on websites for various purposes such as login username/passwords, form fields, etc.

BrowserShield [23] propose to defeat JavaScript-based attacks by rewriting scripts according to a security policy prior to executing them in the browser. In BrowserShield, the rewriting process inserts trusted JavaScript functions to mediate access to the document tree by untrusted scripts.

Jackson et al. [11] describe several unexpected repositories of private information in the browsers cache that could be stolen by XSS attacks. They advocate applying a refinement of the same-origin policy [13] to cover aspects of browser state that extend beyond cookies. By allowing the server to explicitly specify the scripts that it intentionally includes in the document, our approach can also be thought of as an extension of the same-origin policy.

IV. Need and Motivation

Traditionally, content injection was limited to personal computing environments. However, content injection vulnerabilities could spread much faster due to the popularity of mobile computing and cloud computing technologies. Many research efforts have been taken to address problems related to content injection (XSS and SQLI) since the discovery of those attacks. Still, XSS vulnerabilities are still prevalent in web application source codes and attacks are still taking place victimizing site owners and innocent users. Inadequate validation and sanitization of user inputs is root cause of sucessful content injection attacks, however, exisitng solutions failed to address the root cause correctly to mitigate the content injection attacks such as XSS and SQLI.

V. Architecture of the Proposed Solution

Figure 1 shows architecture of the proposed solution to detect and prevent content injection attacks. It contains *user input detector* which extracts user input and send it to detection module for further processing. The *ASCII decoder* unit is used to decode user input data in ASCII format for consistency in data format and void encoding techniques used by attackers to conceal the data. The *Regular Expression Matcher* takes user input from ASCII decoder and also takes *regular expression set* to perform matching of regular expression on user data. *Dynamic pattern matcher* allows updation of rule sets to detect various content injection attacks. The *XSS Sanitizer* module removes XSS code from the input and *SQLI Sanitizer* removes SQL code from the input to sanitize the content.

VI. Mathematical Modelling

`Standard Theorem`: Language (L) is regular if and only if it has a regular expression [10].

`Proposed Theorem`: SQL injection language and XSS injection language is regular if there exists a regular expression for it. To prove the above proposed theorem, we show how to construct for any given regular expression an equivalent NFA.

First, we list the charachteristic pattern for XSS and SQL injection attacks. The SQL injection attack string charactersitics are given below:

1) contains zero or more alphanumeric or underscore characters.

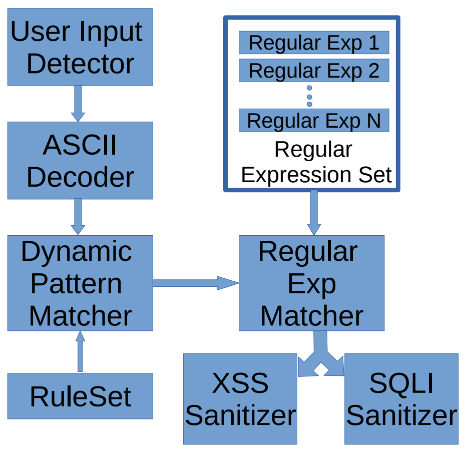

Fig. 1. Architecture of the Proposed Solution

2) presence of ubiquitous single quote or double quote followed by above sequence.
3) the word or with various combinations of its uppercase/lowercase.
4) mathematical equation to make the query true.

The figure 2 shows NFA model for a regular expression to detect SQL injection attack. NFA is a tuple that contains (Q, F, $\sum$, δ, q_0).
Q = $\{q_0, q_1, q_2, - - - - - q_{13}\}$, where Q is set of states.
F = $\{q_9, q_{11}, q_{12}, q_{13}\}$, where F is set of rejection states.
$\sum = \{0, 1, 2, 3, - - -9, a, b, c, d, - - - - - -, x, y, z, <, -, >, ', ", =, \#\}$, where $\sum$ is set of inputs.
δ is transition function.
q_0 is initial state.

The cross-site scripting (XSS) attack string characteristics are given below:

1) contains zero or more alphanumeric or underscore characters.
2) opening $< script >$ tag.
3) contains zero or more alphanumeric or underscore characters.
4) closing $< /script >$ tag.

The figure 3 shows NFA model for a regular expression to detect XSS attack.
Q = $\{q_0, q_1, q_2, - - - - - q_{17}\}$, where Q is set of states.
F = $\{q_{17}\}$, where F is set of rejection states.
$\sum = \{0, 1, 2, 3, - - -9, a, b, c, d, - - - - - -, x, y, z, <, /, >\}$, where $\sum$ is set of inputs.
q_0 is initial state.

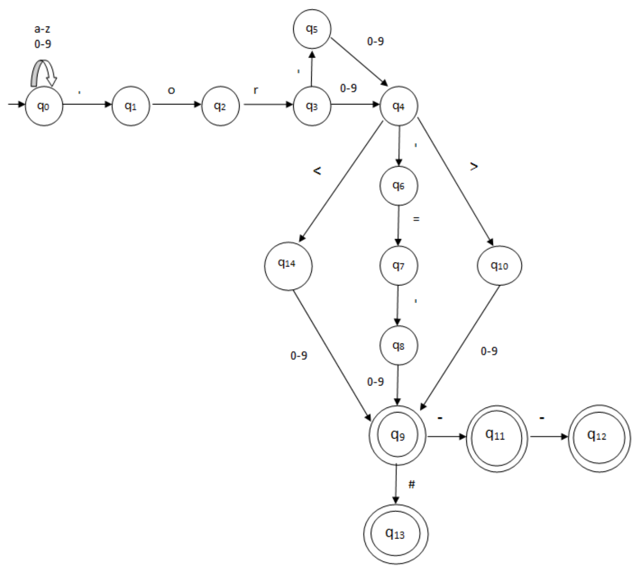

Fig. 2. NFA model of regular expression for SQL injection detection

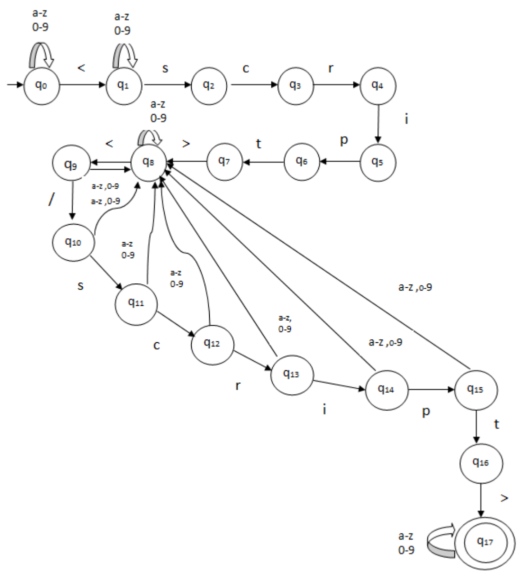

Fig. 3. NFA model of regular expression for XSS detector

VII. Implementation and Evaluation

In this section, we give an empirical analysis of our 2-month long measurement study (from 21st February 2016 to 18th April 2016) on Alexas top 500 sites and phpBB popular PHP blog web application with the following goals. First, we would like to determine whether regular expresions are practical for text input fields in Alexas Top 500 websites and popular PHP apps. Second, we show that our regular expression based scheme and policies defined in Section V are easy to implement and expressive enough to detect content injection on Alexas top websites. Lastly, we show that our approach incurs only a small performance overhead on the browser.

A. Implementation

We implemented the proposed approach by extending Mozilla Firefox version 44.0.2, an open source version of Mozilla Firefox. We implemented an extension which can be patched to the Mozillas Firefox by adding approximately 500 lines of JavaScript code and 100 lines of HTML code. We have released our extension to the Mozillas addon public repository (AMO).

Platform: All experiments were conducted in Mozilla Firefox v44 set up on a HP/Dell Desktop host, configured with Intel(R) Core(TM) i5-200U/i7-3687U 2.10GHz CPU and 4GB RAM running 64-bit Linux Mint 17/Windows 8.

B. Evaluation

The proposed approach is evaluated with a few examples to ensure that it is able to detect content injection. The runtime overhead of the proposed approach is also measured.

Testbed: To measure effectiveness of the proposed approach, various scenarios are configured and tested with the presence of proposed approach and content injection attacks. We created Ubuntu 9.04 virtual machine with phpBB web blog installed and configured in it. We played a role of attacker who injects malicious contents into the phpBB web forum/blog. Without the proposed proposed solution implemented to detect and prevent content injection attack, we sucessfully exploited XSS and SQL injection flaws in the phpBB forum/blog.

Effectiveness Results: The content injection attacks manually tried with the Ubuntu 9.04 virtual machine we created with phpBB web blog. We performed same content injection attacks on both vanilla Firefox web browser and Firefox web browser with the proposed solution prototype installed. The content injection attacks were successful when we used vanilla Firefox web browser, whereas they were prevented when we used the Firefox web browser installed with the extension prototype of the proposed approach.

Preformance Overhead: To measure performance overhead, we manually visited atmost three web pages for every domain in the Alexas top 500 websites and measure the time required to process each input by the proposed approach prototype in Mozilla Firefox web browser - average over five attempts. The average performance overhead of Alexas 500 web site is measured as 1.02%. Such performance overhead is acceptable for all websites and shows compatibility of the proposed approach.

C. Testing Scenarios

Preposition1: Resilience to cross-site scripting(XSS) attacks.
Scenario: Consider an adversary X that injects script code into the form fields $< script > ... < /script >$ and clicks on the submit button to the send it the web server.
Resilience: The extension prototype of the proposed approach intercepts all text input fields and performance regular expression check on user inputs to check the presence of script tags in the input. If it finds script tags then it replaces $<$ symbol in the script tag with $\&lt;$ and $>$ symbol with $\&gt;$. This prevents script from execution on victim users browser.

Preposition2: Resilience to SQL injection attacks.
Scenario: Consider an adversary X that injects SQL injection code into the form fields $'or1 = 1--$ and clicks on the submit button to the send it the web server.
Resilience: The extension prototype of the proposed approach intercepts all text input fields and performance regular expression check on user inputs to check the presence of SQL code symbols in the input. If it finds SQL code symbols then it replaces them with the corresponding HTML code. This prevents web server from mistakenly interpretting user input as SQL code and SQL code execution on the web server.

VIII. Conclusion

This paper analyzed important aspects in content security systems. We proposed an efficient way to identify content injection attacks (XSS and SQL injection) using regular expressions. In addition, we presented a Nondeterministic Finite Automata (NFA) based implementation, which takes advantage of new basic building blocks to support more complex regular expressions than the previous approaches.

In this paper, we conduct a longitudinal manual study of input fields avaialble on websites to evaluate the efficacy of regular expression-based sanitizer for preventing script injection attacks. We then propose a system, which 1) employs a multi-layered whitelisting approach using a novel regular expression scheme, structural regular expression, that is robust against content injection attacks; and 2) comes with an proof-of-concept implementation to ensure its practicality in real-world settings. Our large-scale evaluation shows that the proposed approach can prevent content injection attacks with reasonable performance.

REFERENCES

[1] G. Buehrer, B.W. Weide, and P.A.G. Sivilotti. Using parse tree validation to prevent sql injection attacks. In *Proceedings of the 5th International Workshop on Software Engineering and Middleware*, 2005.

[2] CGIsecurity. The cross-site scripting (xss) faq. http://www.cgisecurity.com/xss-faq.html.

[3] Xinshu Dong, Kailas Patil, Xuhui Liu, Jian Mao, and Zhenkai Liang. An entensible security framework in web browsers. *Technical Report TR-SEC-2012-01, Systems Security Group, School of Computing, National University of Singapore*, 2012.

[4] Xinshu Dong, Kailas Patil, Jian Mao, and Zhenkai Liang. A comprehensive client-side behavior model for diagnosing attacks in ajax applications. In *Proceedings of the 18th International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2013.

[5] Dennis Fisher. Persistent xss bug on twitter exploited by worm. http://threatpost.com/en_us/blogs/persistent-xss-bug-twitter-being-exploited-092110.

[6] W.G.J. Halfond and A. Orso. Amnesia: analysis and monitoring for neutralizing sql-injection attacks. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, 2005.

[7] W.G.J. Halfond and A. Orso. Combining static analysis and runtime monitoring to counter sql-injection attacks. In *Proceedings of the Third International Workshop on Dynamic Analysis*, 2005.

[8] W.G.J. Halfond, A. Orso, and P. Manolios. Using positive tainting and syntax-aware evaluation to counter sql-injection attacks. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2006.

[9] Mark Hofman. Sql injection attack happening atm. isc.sans.org/diary/SQL+Injection+Attack+happening+ATM/12127.

[10] J. E. Hopcroft and J. D. Ullman. Introduction to automata theory, languages and computation. In *Reading, 2nd Ed., Addison-Wesley, 2001*, 2001.

[11] Collin Jackson, Andrew Bortz, Dan Boneh, and John C. Mitchell. Protecting browser state from web privacy attacks. In *Proceedings of the International Conference on World Wide Web (WWW)*, 2006.

[12] Kamlesh Kumar and Deen Bandhu. Prevention and detection techniques for sql injection attacks. In *Proceedings of the IJCTT vol-12, No-03*, 2014.

[13] Mozilla. Same origin policy for javascript. https://developer.mozilla.org/En/Same_origin_policy_for_JavaScript.

[14] Mozillia. Mozilla. signing a xpi. In *https://goo.gl/Ffls5r*.

[15] Nex. The clickjacking meets xss: a state of art. http://www.milw0rm.com/papers/265, 2008.

[16] Anh Nguyen-tuong, Salvatore Guarnieri, Doug Greene, Jeff Shirley, and David Evans. Automatically hardening web applications using precise tainting. In *Proceeding of the 20th IFIP International Information Security Conference*, 2005.

[17] National Institute of Standards and Technology. National vulnerability database (nvd). http://web.nvd.nist.gov/view/vuln/search.

[18] Kailas Patil, Xinshu Dong, Xiaolei Li, Zhenkai Liang, and Xuxian Jiang. Towards fine-grained access control in javascript contexts. In *31st International Conference on Distributed Computing Systems (ICDCS), 2011*, pages 720–729, June 2011.

[19] Kailas Patil, Tanvi Vyas, Fredrik Braun, Mark Goodwin, and Zhenkai Liang. Poster:usercsp-user specified content security policies. Symposium On Usable Privacy and Security (SOUPS) POSTER, 2013.

[20] Tadeusz Pietraszek, Chris V, and En Berghe. Defending against injection attacks through context-sensitive string evaluation. In *Proceeding of the Recent Advances in Intrusion Detection*, 2005.

[21] Cristian Pinzn, Javier Bajo Juan F. De Paz, lvaro Herrero, and Emilio Corchado. Aiida-sql: An adaptive intelligent intrusion detector agent for detecting sql injection attacks. In *Proceedings of the 10th International Conference on Hybrid Intelligent Systems*, 2010.

[22] OWASP-The Open Web Applicaiton Security Project. Owasp top ten project. https://www.owasp.org/index.php/Top10#OWASP_Top_10_for_2013.

[23] Charles Reis, John Dunagan, Helen J. Wang, Opher Dubrovsky, and Saher Esmeir. Browsershield: Vulnerability-driven filtering of dynamic html. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.

[24] RSnake. Xss(cross site scripting) cheat sheet esp: for filter evasion. http://ha.ckers.org/xss.html.

[25] Jesse Ruderman. Signed scripts in mozilla. http://www.mozilla.org/projects/security/ components/signed-scripts.html.

[26] Zhendong Su and Gary Wassermann. The essence of command injection attacks in web applications. In *Proceedings of the ACM Symposium on Principles of Programming Languages (POPL)*, 2006.

[27] Symantec. Internet security threat report volume 20. https://www4.symantec.com/mktginfo/whitepaper/ISTR/21347932_GA-internet-security-threat-report-volume-20-2015-social_v2.pdfg, April 2015.

[28] Stephen Thomas, Laurie Williams, and Tao Xie. On automated prepared statement generation to remove sql injection vulnerabilities. In *Proceedings of the Elsevier Journal on the Information and Software Technology*, 2009.

[29] Wikipedia. Cross-site scripting. http://en.wikipedia.org/wiki/Cross-site_scripting.

[30] Wikipedia. Sql injection. https://en.wikipedia.org/wiki/SQL_injection.

[31] Yichen Xie and Alex Aiken. Static detection of security vulnerabilities in scripting languages. In *Proceedings of the USENIX Security Symposium*, 2006.

[32] xssed.com. Myspace.com hit by a permanent xss. http://www.xssed.com/news/83/Myspace.com_hit_by_a_Permanent_XSS/.

[33] xssed.com. New orkut xss worm by brazilian web security group. http://www.xssed.com/news/77/New_Orkut_XSS_worm_by_Brazilian_web_security_group/.

[34] Z. Yan and S. Holtmanns. Trust modeling and management: from social trust to digital trust. In *IGI Global*, 2008.