Current Trends in Technology and Science
ISSN: 2279-0535, July-2020, Volume: XI, Issue: IV

Handling Susceptible Script Encryption in Web Environment

Amit Verma, Rahul Tiwari, Shital Gupta
CSE Department, SORT Peoples University
amitverma2196@gmail.com, ahultiwari3189@gmail.com, email4sgupta@gmail.com

Abstract: Today, web security is becoming more and more significant in our daily life. Remaining to that the fact that we cannot sentient without the Internet, providing a decent and security networking environment is suggestively necessary. However, cross site scripting (XSS) attacks threat millions of websites. XSS can be recycled to inject malevolent scripting code into applications, and then return the code back to the customer side. When operators use the web browser to visit the place where the spiteful scripting code has been injected, the code will execute directly to the customers computer. A common solution is distinguishing the key words of XSS in the browser JavaScript appliance or on the attendant part to filter the malicious code. Nonetheless, the attacker can concept different new types of malicious are scripting to avoid detecting so that it is difficult to collect all keywords in the detecting-list to avoid XSS attacking. Therefore, it is worth allowing more people pay consideration to XSS and discovery more solutions to avoid XSS occurrences.

Keywords: - JavaScript, XSS Attacking

1. Introduction

The most exceptional revolution in the history of mankind in the field of communication is the Internet. There are various services and resources, defined as Internet. As the Internet is growing, the web sites become more professional and dynamic. The term WWW (World Wide Web), which 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 aiming at providing information expressed in text and graphics. As the Internet is growing, the web sites become more professional and dynamic. In order to be able to change the design of the web page to meet today's taste and to provide personalized and current information to the users, the web sites no longer use static web pages. 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. Apart from advantages of Internet like, faster communication, information sharing, entertainment, social networking, online-services (education, banking, government enterprises etc.) and e-commerce, there are some disadvantages with it e.g. theft of personal information, spamming, virus threat, pornography, social disconnect etc. The benefits have to communicate, to share data from one to many and many more having some drawbacks, by which any third person can steal your data without your knowledge and permission, is termed as attack.

Web has also become complex, without boundaries and immediate in its nature. A single Web page today can be comprised of information from many simultaneous sources from around the world. It only takes one of these sources to be compromised in order for a new Web attack to be quickly propagated and delivered to many unsuspecting Web users. Web-based systems are a composition of infrastructure components, such as web servers and databases, and of application-specific code, such as HTML-embedded scripts and server-side applications.

2. RELATED WORK

By Shaukat Ali, Shah Khusro, AzharRauf [6] et.al, provide security framework that will use well-known cryptographic techniques to address the issues of data confidentiality, data integrity, and authentication as well as protection against the most common XSS and CSRF attacks in web mashups. This new approach to software development can pose many security challenges that bypass the domain of cross site referencing and issues in data integrity, user authentication, and data confidentiality emerge. Presents a security framework using well-known cryptographic techniques that can be used in Server-Side mashup model and will provide solutions to most common mashup security attacks suchas CSRF, XSS and other relevant security issues.

Ms. R. Priyadarshini, Ms. Jagadiswaree D, Ms. Fareedha A, Mr. Janarthanan M/ B. S. Abdur Rahman [7] et.al, Proposed solution to detect and prevent against the malicious attacks over the developer's Web Application written in programming languages like PHP, ASP.NET and JSP also created an API (Application Programming Interface) inactive language through which transactions and interactions are sent to IDS Server through Inter Server Communication Mechanism. System is not restricted to detect and prevent vulnerabilities in PHP based applications but also can be used for the web applications developed in .NET and JSP by which the intrusions are detected and prevented via IDS Server with the API developed in native language.

Jan-Min Chen/ Yu Da University Miaoli, Taiwan, Chia-Lun Wu/ Tatung University Taipei, Taiwan [8] et.al, implemented an automated vulnerability scanner that for the injection attacks. It is a system that automated scanned the injection attack vulnerabilities. Model scans Web application security were detecting vulnerability based on injection point, exactly obtain the information of injection point, and using black box testing to analysis what potential vulnerability, tackled vulnerable injection point.

Hossain Shahriar and Mohammad Zulkernine [9], et al. developed a server side JavaScript code injection detection approach, in which pre and post pend each legitimate JavaScript code block with comment statements that include identical random token. They identify the expected features of a JavaScript code block, save the features in policies, and embed the policy information into comments. The injection attack includes, (i) code without comment, (ii) code with correct and duplicate comment, and (iii) code with correct and non-duplicate comment; however, the actual code features are not matching with the intended features specified in a policy. They developed a prototype tool in Java to inject JavaScript comments and generate policies based on legitimate code features.

Blake Anderson and Daniel Quist [10] et al. demonstrate the application of a Gaussian process change-point model to the problem of identifying a code injection attack against the Internet Explorer browser. Using system call traces to condition the transition probabilities of Markov chains, and able to locate the change-points caused by these exploits.

Ryan Riley, Purdue University, Xuxian Jiang, George Mason University, DongyanXu, Purdue University [11], et al. present an architectural approach to prevent code injection attacks. Instead of maintaining the traditional single memory space containing both code and data, which is often exploited by code injection attacks, the approach creates a split memory that separates code and data into different memory spaces. Consequently, in a system protected by code injection attacks may result in the injection of attack code into the data space.

Hallaraker et al. [12] proposed a strictly client-side mechanism for detecting malicious JavaScript's. The system uses an auditing system in the Mozilla Firefox web browser that can perform both anomaly or misuse detection. This system monitors the execution of JavaScript and compares it to high level policies to detect malicious behavior. For each scenario specific, rules have to be implemented to enable detection. These rules allow specifying sequences of JavaScript methods, together with their corresponding, that are considered malicious, parameters. With this information, state driven rules can be implemented. The system performs most of the auditing in XPConnect, which is the layer that connects the JavaScript engine with the other components of Mozilla Firefox. Some additional auditing features are implemented in DOMClassInfo (interface flattening and behavior implementing), Live Connect (communication between JavaScript, Java applets and other plugins) and the Security Manager. Internal processing performed by the JavaScript program is not accessible to the rules.

3. Methodology

3.1 Approach

To move the analysis of systems security from an art to a science, a framework for a methodical security analysis and recommendation of solutions will develop. This framework includes a methodical process of creating attack and protection trees, development of metrics and rule sets to propagate the metrics throughout the trees, and tools for the analyst to interface with a decision-maker to select the appropriate protections for the system. In this approach, implement protections into a system which is repeatable and unambiguous.

3.2 Data Needs

To properly analyze the security of web application system, certain system dependent on user inputs are required. These inputs may include such things as probabilities of success and costs for attacks and protections. The focus in this research is not on how the inputs are derived but rather on what to measure and how the user inputs can be used in the security analysis.

3.3 Analytic Techniques

Testing software for XSS vulnerabilities can be done in two ways: static and dynamic testing.
Static testing is typically done by performing source code analysis. A method that creates a control flow graph of information that is processed by a server page. The graph consists of input and output nodes. An input node can be a statement that processes input data from a form, reads the value of a query string, a database field, a cookie, or data from a file. Output nodes are associated with statements that write to database fields, a file, a cookie, or output in the page. The server page is potentially vulnerable if a path in the control flow graph exists that connects an input to an output node. However, it is possible that data from one server page is sent to another page, the web application might not be vulnerable to a certain type of attack if only one of the

individual server pages has potential security problems. For example, a page may read input and store it in a database field. The result of the static analysis says that this page has a potential vulnerability. But another page that reads data from this field may encode everything in the output of the page and therefore, the web application as a whole is not vulnerable.
In dynamic testing, known attacks are executed against web applications [2, 3]. Either database with generated attacks for a specific web application is used or a database that contains generic attacks. In [3], the authors implement dynamic testing as a second stage of their web application assessment. More precisely, server pages that are potentially vulnerable according to a previous static analysis step are tested again in a dynamic test with specific attacks for the potential vulnerability. The crawler described in [2] does black box testing with a generic database. It analyses the generated pages of the web application and then chooses attacks to perform. This method tests web applications without requiring user interaction and interprets the response of the web application to the chosen attack.
Traffic Analysis proxy system is proposed in [4], which can be installed on the user side to prevent XSS attacks. This proxy monitors the HTTP-requests and responses [5] of the user who is surfing the web. It has two modes called „response change mode" and „request change mode". In the „response change mode" the proxy stores information about requests which contain special tags.

3.4 Proposed Research

During the research following domain are explored to make secure web application.

- Input validation vulnerable detection
- With the use of Input validation vulnerable attack, attacker may direct and complete access to the databases as well as the system to steal personal information. Hacker can also damage the system and the control of the system.
- Taint analysis of JavaScript code
- To "Taint" user data is to insert some kind of tag or label for each object of user data, which allows tracking the influence of tainted object along with the execution of program.
- Auditing JavaScript code execution.

For auditing the JavaScript code, requires the completeness and correctness where the execution environment must be initialized and callbacks to the compiler. These hooks are used when scripts try to access a specific property or method that is not native in the engine. Other than this it is used to represent the logging information, which would be used to detect vulnerable code.

- Policy enforcement for JavaScript code. With the policy enforcement with JavaScript, enforce security and reliability policies, analyze web gadgets, and also to enhance the performance utilization.

4. Conclusion

The JavaScript language is used to enhance the client side display of web pages. JavaScript code is downloaded into browsers and executed on-the-fly by an embedded interpreter. Browsers provide sand-boxing mechanisms to prevent JavaScript code from compromising the security of the client's environment, but, unfortunately, a number of attacks exist that can be used to steal users' credentials (e.g., cross site scripting attacks) and lure users into providing sensitive information to unauthorized parties (e.g., phishing attacks). We propose an approach to solve this problem that is based on monitoring JavaScript code execution in the browser.

References

1. Yao-Wen Huang, Shih-Kun Huang, and Tsung-Po Lin. Web Application Security Assessment by Fault Injection and Behavior Monitoring. WWW 2003 Budapest Hungary, May 2003.
2. G.A. Di Lucca, A.R. Fasolino, M. Mastroianni, and P. Tramontana. Identifying Cross Site Scripting Vulnerabilities in Web Applications. In Sixth IEEE International Workshop on Website Evolution (WSE'04), pages 71 – 80, September 2004.
3. Omar Ismail, Masashi Etoh, Youki Kadobayashi, and Suguru Yamaguchi. A Proposal and Implementation of Automatic Detection/Collection System for Cross-Site Scripting Vulnerability. In Proceedings of the 18th International Conference on Advanced Information Networking and Application (AINA04), March 2004.
4. T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.0. http://www.rfc-editor.org/rfc/rfc1945.txt, 1996.
5. Shaukat Ali, Shah Khusro, AzharRauf / University of Peshawar, Peshawar, Pakistan/ IEEE, A Cryptography-Based Approach to Web Mashup Security,2011
6. Ms. R. Priyadarshini, Ms. Jagadiswaree. D, Ms. Fareedha. A, Mr. Janarthanan. M / B. S. Abdur Rahman University Chennai/IEEE, A Cross Platform Intrusion Detection System using Inter Server Communication Technique, 2011.
7. Jan-Min Chen/ Yu Da University Miaoli, Taiwan, Chia-Lun Wu/ Tatung University Taipei, Taiwan /IEEE, An Automated Vulnerability Scanner for Injection Attack Based on Injection Point, 2010.
8. Hossain Shahriar and Mohammad Zulkernine, Queen's University, Kingston, Canada, "Injecting Comments to Detect JavaScript Code Injection

Attacks", IEEE 978-0-7695-4459-5/11, DOI 10.1109/COMPSACW.2011.27,104-109, 2011.

9. Blake Anderson and Daniel Quist, Los Alamos National Lab, Terran Lane, University of New Mexico, "Detecting Code Injection Attacks in Internet Explorer", IEEE 978-0-7695-4459-5/11, DOI 10.1109/COMPSACW.2011.25, 90-95, 2011.
10. Ryan Riley, Purdue University, Xuxian Jiang, George Mason University, DongyanXu, Purdue University, "An Architectural Approach to Preventing Code Injection Attacks", IEEE 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks 0-7695-2855-4/07, 2007.
11. O. Hallaraker and G. Vigna. " Detecting Malicious JavaScript Code in Mozilla ",In proceedings of the IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), 2005.
12. E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic, "Noxes: A client-side solution for mitigating cross-site scripting attacks", In 21st ACM Symposium on Applied Computing (SAC), 2006.
13. K. Selvamani, A.Duraisamy, A.Kannan "Protection of Web Applications from Cross-Site Scripting Attacks in Browser Side" (IJCSIS) International Journal of Computer Science and Information Security, Vol. 7, No. 3, March 2010.
14. P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna. "Cross site scripting prevention with dynamic data tainting and static analysis". In Proceeding of the Network and Distributed System Security Symposium (NDSS07), 2007.
15. E. Gal´an, A. Alcaide, A. Orfila, J. Blasco "A Multi–agent Scanner to Detect Stored–XSS Vulnerabilities" in ICITST, Technical Co-Sponsored by IEEE UK/RI Communications, 2010
16. Zhang Xin-hua, Wang Zhi-jian /Hohai University, China/ IEEE, A Static Analysis Tool for Detecting Web Application Injection Vulnerabilities for ASP Program, 2010.
17. Masaru Takesue, Dept. Applied Informatics, Hosei University, Tokyo/IEEE, an HTTP Extension for Secure Transfer of Confidential Data, 2009.
18. Abdul Razzaq, Ali Hur, NasirHaider, Farooq Ahmad/NUST School of Electrical Engineering and Computer Sciences, Pakistan/ IEEE, Multi-Layered Defense against Web Application Attacks, 2009.