Volume 3, No. 5, May 2012

Journal of Global Research in Computer Science

ISSN-2229-371X

TECHNICAL NOTE

Available Online at www.jgrcs.info

DEFENDING AGAINST WEB VULNERABILITIES AND CROSS-SITE SCRIPTING

T.Venkat Narayana Rao[*1], V. Tejaswini[*2] ,K..Preethi[*3]

[*1] Professor, Department of Computer Science and Engineering
Hyderabad Institute of Technology and Management [HITAM], Hyderabad, A.P, India.
tvnrbobby@yahoo.com

[2]Student, B.Tech Third Year, Department of Information Technology
Hyderabad Institute of Technology and Management [HITAM], Hyderabad, A.P,
4tejaswani@gmail.com

[3] Associate Professor, Department of Computer Science and Engineering
Hyderabad Institute of Technology and Management [HITAM], Hyderabad, A.P, India

Abstract: Researchers have devised multiple solutions to cross-site scripting, but vulnerabilities persists in many Web applications due to developer's lack of expertise in the problem identification and their unfamiliarity with the current mechanisms. As proclaimed by the experts, cross-site scripting is among the serious and widespread threats in Web applications these days more than buffer overflows. Recent study shows XSS has ranked first in the MITRE Common Weakness Enumeration (CWE)/SANS Institute list of Top 25 Most Dangerous Software Errors and second in the Open Web Application Security Project (OWASP). However, vulnerabilities continue to exist in many Web applications due to developers' lack of understanding of the problem and their unfamiliarity with current guarding strengths and limitations. Existing techniques for defending against XSS exploits suffer from various weaknesses: inherent limitations, incomplete implementations, complex frameworks, runtime overhead, and intensive manual-work requirements. Security researchers can address these weaknesses from two different perspectives. They need to look beyond current techniques by incorporating more effective input validation and sanitization features. In time, development tools will incorporate security frameworks such as ESAPI that implement state-of-the-art technology. This paper focus on program verification perspective, how researchers must integrate program analysis, pattern recognition, concolic testing, data mining, and AI algorithms to solve different software engineering problems and to enhance the effectiveness of vulnerability detection. Focus on such issues would improve the precision of current methods by acquiring attack code patterns from outside experts as soon as they become available.

Keywords: XSS, vulnerability, Injection, overhead, markup.

WHAT IS CROSS-SITE SCRIPTING (XSS)

Cross-site scripting (XSS) is a type of computer insecurity vulnerability typically found in Web applications, such as web browsers which breach the security that enables attackers to infuse client-side script into Web pages viewed by other users [1]. A cross-site scripting vulnerability may be used by attackers to bypass access controls such as the same origin policy. Several major websites including Face book, Twitter, MySpace, eBay, Google, and McAfee have been the targets of XSS exploits. XSS is the result of limitations inherent in many Web applications' security mechanisms i.e. the lack or insufficient refinement of user inputs. XSS flaws exist in Web applications written in various programming languages such as PHP, Java, and .NET where application WebPages reference unrestricted user inputs. Attackers inject malicious code via these inputs, thereby causing unintended script executions through clients' browsers.

Researchers have proposed multiple XSS solutions ranging from simple static analysis to complex runtime protection mechanisms. Cross-site scripting carried out on websites accounted for roughly 80.5% of all security vulnerabilities recorded by Symantec as of 2007. Their effect may range from a petty trouble to a significant overhead of security risk, depending on the value of the data handled by the vulnerable site and the nature of any security mitigation implemented by the site's owner. From a development perspective, researchers need to craft simpler, better, and more flexible security alternatives. Cross-site scripting flaws are web-application vulnerabilities which allow attackers to bypass client-side security mechanisms normally imposed on web content by modern web browsers.

By finding ways of injecting malicious scripts into web pages, an attacker can gain elevated access-privileges to sensitive page content, session cookies, and a variety of other information maintained by the browser on behalf for user. Cross-site scripting attacks are therefore a unique case of code injection [2]. The expression "cross-site scripting" originally referred to the act of inducing the attacked, third-party web application from an distinct attack site, in a manner that executes a section of JavaScript programmed by the attacker in the security framework of the targeted domain. The definition gradually expanded to encompass other modes of code injection, including persistent and non-JavaScript vectors (including Java, ActiveX, VBScript, Flash, or even pure HTML, and SQL Queries), causing some uncertainty to newcomers to the field of information security [3]. XSS vulnerabilities have been reported and exploited since the 1990s. Well-known sites affected in the history include the social-networking sites Twitter, Face book, MySpace, and Orkut. In recent years, cross-site scripting flaws surpassed buffer overflows to become the most common publicly-reported security vulnerability, with some researchers in 2007 viewing as many as 68% of websites as likely open to XSS attacks.

TYPES OF XSS EXPLOITS

Persistent or Stored Attacks:

The persistent or stored XSS vulnerability is transpired when the attacker is provided the data is saved back by the server, and then returned to other users in the course of normal browsing permanently displayed as "normal" pages, without appropriate HTML escaping. Mostly this type of vulnerability occurs in the Social websites where-in members scan the profiles of other members [2]. For privacy reasons, this site hides everybody's unique personal identity and email. These are kept secret on the server. Particularly in the case of social networking sites, the code would be further designed to self-propagate across accounts, creating a indirect kind of a client-side worm. Persistent XSS can be more significant than other types because an attacker's malicious script is turned into automatic nature, without the need to individually target victims or also lure them to a third-party website. Any data received by the web application (via email, system logs, etc.) that can be controlled by an attacker could befall into injection vector.

Non-persistent or Reflected Attacks:

Non-persistent XSS vulnerabilities in Google could permit malicious sites to attack Google users who visit them whilst logged in. A potential vector is a site search engine, given a search for a string; the search string will typically be redisplayed verbatim on the result page to indicate what was searched for. If this response does not properly escape or reject HTML control characters, a cross-site scripting flaw would result in. A reflected attack is typically delivered via email or a neutral web site. These holes show up when the data provided by a web client, most frequently in HTTP query parameters or in HTML form submissions, is used immediately by server-side scripts to generate a page of results for that user, without properly cleansing the request.

Because HTML documents have a flat, serial structure that blends control statements, formatting, and the actual content, any non-validated user-supplied data included in the resulting page without proper HTML encoding[5]. This may result in markup injection. In this class of scripting languages are also used, e.g., Action Script and VBScript. Mostly attackers would write the scripting in java language only for common practice of the attack includes a design step. In this context the attacker creates and tests an offending URI(Uniform Resource Indicator), a social engineering step, in which the offender convinces his victims to load this URI on their browsers, and the eventual execution of the offending code[4]. The web application might filter out "<script>", but might not filter %3cscript%3e which simply includes a different encoding of tags. A nice tool for testing character encodings is OWASP's CAL9000.

DOM-Based Attacks:

This name refers to the standard model for representing HTML or XML contents which is called the Document Object Model (DOM). JavaScript programs manipulate the state of a web page and populate it with dynamically-computed data primarily by acting upon the DOM. With the arrival of web 2.0 applications a new class of XSS flaws has emerged i.e. DOM-based vulnerabilities. DOM-based vulnerabilities occur in the content processing stages performed by the client, typically in client-side JavaScript [1].

TYPES OF XSS DEFENCES

XSS defenses can be broadly classified into four types:
a. Defensive coding
b. XSS testing
c. Vulnerability detection
d. Runtime attack prevention.

It compares various current techniques, which each have strengths and weaknesses.

Defensive Coding

XSS arises from the improper handling of inputs, using defensive coding practices that validate and sanitize inputs is the best way to eliminate XSS vulnerabilities. The user must make sure that the inputs are validated and conform to a required input format[3].
The four basic input sanitization options are :
a. Replacement and elimination methods search for known bad characters (blacklist).
b. The former replaces them with non-malicious characters, whereas the latter simply removes them.
c. Escaping methods search for characters that have special meanings for client-side interpreters and remove those meanings.
d. Restriction techniques limit inputs to known good inputs (white list).

Checking blacklisted characters in the inputs is more scalable, but blacklist comparisons often fall short as it is difficult to foresee every attack signature alternative. White list comparisons are considered more protected, but they can result in the denial of many unlisted valid inputs. OWASP has issued rules that define proper escaping schemes for inputs referenced in different HTML output locations.

XSS Testing:

Input validation testing could expose XSS vulnerabilities in Web applications. Specification based IVT methods generate test cases with a plan of exercising various combinations of valid or invalid input conditions stated in specifications. In general, the effectiveness of both specification and code based approaches depends largely on the completeness of specifications or the sufficiency of generated test suites for discovering XSS vulnerabilities in source code. Hossain Shahriar and Mohammad Zulkernine developed MUTEC, a fault based XSS testing tool that creates mutated programs by changing responsive program statements, or sinks, with mutation operators. Only test cases containing adequate XSS attack vectors can bring about original and mutated programs to behave.
Example 1.
document.write (escape (document.URL.substring (pos,document.URL.length)));
One such test case is User :
→ <Script>alert('XSSed!')</Script>

And then attempts to find a test case that result in a different number of HTML tags between the original statement and its mutated statement. MUTEC generates adequate test suites for exposing XSS vulnerabilities but requires intensive labor as the task of generating mutants is not automated.

Vulnerability Detection:

This type of XSS defenses focus on identifying vulnerabilities in server-side scripts. Static-analysis based approaches can demonstrate the absence of vulnerabilities, but they tend to produce many false positives. Recent approaches combine static analysis with dynamic analysis techniques to improve accuracy.

Static Analysis:

Benjamin Livs and Monica Lam used binary decision diagrams to relate points to analysis to server-side scripts. Their approach requires users to specify vulnerability patterns in Program Query Language [6] .Yichen Xie and Alex Aiken proposed a static analysis technique that acquire block and function summary information from symbolic execution Pixy, an open source vulnerability scanner and also includes alias analysis to improve precision. These techniques identify tainted inputs accessed from exterior data sources, track the flow of tainted data, and check if any reached sinks such as SQL statements and HTML output statements. For example, for the program travelerTip it reports the following statements as vulnerable:

Example 2.

```
out.println("Your Post has been added
under Place '" + HTMLencode(place)+"'");
out.println("Your Message: '"+
new_tip+ "' is too long!");
out.println(""+tip+"");
```

XSS vulnerabilities in source code and are relatively easy for security personnel to implement and adopt. However, they cannot check the correctness of input sanitization functions and, instead, generally assume that unhandled or unknown functions return unsafe data. These approaches also miss DOM-based XSS vulnerabilities as they do not target client-side scripts.

Static String Analysis:

The enhancement provides more accuracy as it can examine string operations effects on inputs. However, when conducting static string analysis, it is complicated to model complex operations such as string-numeric interaction; thus, this approach can result in false positives if analysts make conventional approximations when handling such operations. Gary Wassermann and Zhendong Su enhanced the original taint-based approaches with string analysis. Their technique uses context-free grammars (CFGs) to signify the values a string variable can hold at a certain program point, which facilitates the checking of blacklisted string values in sensitive program statements. Static string analysis also suffers from the limitations of blacklist comparisons.

Runtime attack prevention:

In general, these methods set up a proxy between the client and server to capture incoming or outgoing HTTP traffic. The proxy then checks the HTTP data for illegal scripts or verifies the resulting URL connections against safety policies. XSS defenses focus on preventing real time attacks using intrusion detection systems or runtime monitors, which can be deployed on either the server side or client side.

Client Side Prevention:

Its main disadvantage is that it requires client actions whenever a connection violates the filter rules. Moreover, this approach addresses all types of XSS attacks, it only detects abuse that send user information to a third-party server, not any other exploit such as those involving Web content manipulation. Noxes acts as a private firewall that allows or blocks connections to websites on the basis of filter rules, which are basically user-specified URL white lists and blacklists. When the browser sends an HTTP request to an anonymous website, Noxes immediately alerts the client, who chooses to allow or deny the connection, and remembers the client's action for prospect use. Client-side prevention provides a personal protection layer for clients so that they need not depend on the security of Web applications.

Server Side Prevention:

Users specify prerequisites of sensitive functions i.e. those that contain HTML outputs and post conditions of sanitization functions. During runtime, instrumented guards ensure for conformance of these user-specified conditions. The WebSSARI (Web Security via Static Analysis and Runtime Inspection) tool, which executes type based static analysis to identify potentially weak code sections and implement them with runtime guards. Other approaches use dynamic taint-tracking mechanisms to monitor the stream of input data at runtime [6]. They ensure that these inputs are syntactically restricted (only treated as literal values) and do not hold unsafe content defined in user-specified security policies. Some server side prevention mechanisms require the collaboration of browsers. One example is BEEP (Browser-Enforced Embedded Policies), a mechanism that modifies the browser so that it cannot execute unlawful scripts. Security policies dictate what data the server sends to BEEP-enabled-browsers.

IMPLEMENTATION OF XSS DEFENCES

We all consent that cross-site scripting is a serious problem, but what continues to amaze me is the lack of good documentation on the subject. It is easy to find instructions how to execute attacks against applications vulnerable to XSS, but finding something adequate to cover defense is a real challenge [2]. No wonder programmers keep making the same errors over and over again. I am sure that one page that describes the problems and the solutions is somewhere out there, but I have been unable to find it. All I am getting is a page after page after page of half-truths and partial information, and even people saying that XSS is impossible to defend against [3]. To help developers practice its defensive coding rules, OWASP has created the Enterprise SecurityAPI

(https://owasp.org/index.php/Category:OWASP_Enterprise_Security_API) i.e. ESAPI, an open source library for many different programming languages. Microsoft also provides the Web Protection Library (http://wpl.codeplex.com) for .NET developers. To produce web applications that are safe against XSS and other injection attacks [7]. Every such function must be aware of the character encoding used in the application. Then, for every piece of code that sends data from one component into another, make sure you use the correct function to encode data to make it safe check that every piece of data you receive is in the correct character encoding and that the format matches that of the type you are expecting (input validation). One must use white listing (as blacklisting does not work) in preventing attackers from executing JavaScript code in data pretending to be an Internet address.

Example 3.
instead of http://www.btech.com, which you use to create a link <a href="http://www.btech.com">btech</a>,
we get javascript:alert('xss')

CONCLUSION

Inherent limitations, unfinished implementations, complex frameworks, runtime overhead and rigorous manual-work requirements. These are the existing techniques for defending against XSS exploits suffer from various weaknesses. Security researchers can deal with these weaknesses from two different perspectives. Researchers need to craft simpler, better, and more flexible security defenses [6] . They need to look beyond current techniques by incorporating more effective input validation and sanitization features. In time to come many development tools would be incorporated for security frameworks such as ESAPI that implement state-of-the-art technology. Researchers must integrate program analysis, pattern recognition, concolic testing, data mining, and AI algorithms would be used rigorously in future to solve different software engineering problems to improve the effectiveness of vulnerability detection. They can also improve the precision of current methods by gaining attack code patterns from outside experts.

REFERENCES

[1]. M.S. Lam et al., “Securing Web Applications with Static and Dynamic Information Flow Tracking,” Proc. 2008 ACM SIGPLAN Symp. Partial Evaluation and Semantics-Based Program Manipulation (PEPM 08), ACM, 2008, pp. 3-12.

[2]. Y. Xie and A. Aiken, “Static Detection of Security Vulner-abilities in Scripting Languages,” Proc. 15th Usenix Security Symp. (Usenix-SS 06), vol. 15, Usenix, 2006, pp. 179-192.

[3]. D. Balzarotti et al., “Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications,” Proc. 29th IEEE Symp. Security and Privacy (SP 08), IEEE CS, 2008, pp. 387-401.

[4]. CERT:"CERT Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web-Requests", http://www.cert.org/advisories/CA-2000-02.html.

[5]. Aung Khant:"What XSS Can do - Benefits of XSS From-Attacker's-view" http://yehg.net/lab/papers/Do.pdf

[6]. N. Li et al., “Perturbation-Based User-Input Validation Testing of Web Applications,” J. Systems and Software, Nov. 2010, pp. 2263-2274.

[7]. H. Shahriar and M. Zulkernine, “MUTEC: Mutation-Based Testing of Cross Site Scripting,” Proc. 5th Int’l Workshop Software Eng. for Secure Systems (SESS 09), IEEE, 2009, pp. 47-53.

Short Bio Data for the

#1.Professor T.Venkat Narayana Rao, received B.E in Computer Technology and Engineering from Nagpur University, Nagpur, India, M.B.A (Systems), holds a M.Tech in Computer Science from Jawaharlal Nehru Technological University, Hyderabad, A.P., India and a Research Scholar in JNTU. He has 21 years of vast experience in Computer Science and Engineering areas pertaining to academics and industry related I.T issues. He is presently Professor and Head, Department of Computer Science and Engineering, Hyderabad Institute of Technology and Management [HITAM], Gowdavally, R.R.Dist., A.P, INDIA. He is nominated as an Editor and Reviewer to 26 International journals relating to Computer Science and Information Technology. He is currently working on research areas which include Digital Image Processing, Digital Watermarking, Data Mining, Network Security and other emerging areas of Information Technology. He can be reached at tvnrbobby@yahoo.com

Vedula Tejaswini is Pursing B.Tech Third year in Information Technology from Hyderabad Institute of Technology and Management [HITAM], Gowdavelly, R.R.Dist., A.P, INDIA, Affiliated to Jawaharlal Nehru Technological University (JNTU) Hyderabad.

K.Preethi, B.Tech, Graduate in Computer Science and Engineering from Jawaharlal Nehru Technological University, Hyderabad, India and M.Tech in Computer Science and Engineering from Jawaharlal Nehru Technological University, Hyderabad, India, Hyderabad, A.P, India . She is presently working as an Associate Professor in the department of Computer Science and Engineering in Hyderabad Institute of Technology and Management (HITAM), Gowdavelly,

K.Preethi, B.Tech, Graduate in Computer Science and Engineering from Jawaharlal Nehru Technological University, Hyderabad, India and M.Tech in Computer Science and Engineering from Jawaharlal Nehru Technological University, Hyderabad, India, Hyderabad, A.P, India . She is presently working as an Associate Professor in the department of Computer Science and Engineering in Hyderabad Institute of Technology and Management (HITAM), Gowdavelly,