ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 5, Special Issue 3, January 2018

S[3]INTEGRATED WEB APPLICATION SECURITY

ANAND TILAGUL CHANDRASHEKHARA J M SHARATH P V BADRINATH K
Asst.profDept.of ISE
arilanand@gmail.com chandujm@gmail.com
sharathpvs@gmail.com
badrik86@gmail.com
SJCIT,Chickballapur

SAGAR J
Dept.of ISE
sagar.j.sam.2010@gmail.com
SJCIT,Chickballapur

ABSTRACT

A great number of webapplication vulnerabilities are leveraged through client-side submission of unexpected inputs. While it is clear these vulnerabilities are complex and widespread, what is not clear is why after over a decade of effort they remain so prevalent. This paper explores a number of methods for combatting this class of threats and assesses why they have not proven more successful. The paper describes the current best practices for minimizing these vulnerabilities and points to promising research and development in the field.

I. INTRODUCTION

An ever-increasing number of high profile data breaches have plagued organizations over the past decade. A great number of these come about via so-called 'injection attacks'; the submission of malicious code to a web application. Indeed, the Open Source Web Application Security Project (OWASP), the leading organization in the field of web app security states; "How data input is handled by Web applications is arguably the most important aspect of security."

(OWASP, 2012). How does such a well understood, heavily researched and often warned against threat not get resolved over a period of 10+ years? Can web application security's biggest nemesis ever be bested, or are we doomed to another decade of continued breaches?

II. AN OVERVIEW OF THE THREAT

Input injection attacks may serve a number of ends. Generally, they are preferred by malicious users as a way toobtain restricted data from a back end database or to embed malicious code onto a web server that will in turn serve up malware to unsuspecting clients. These clients may find their credentials or personal information exfiltrated as a result

The Common Weakness Enumeration (CWE) is a community-developed dictionary that catalogs software weaknesses (Mitre, 2012). Mitre, in collaboration with SANS, publishes an annual analysis of the most significant software errors (in terms of their security implications). The 2011 CWE/SANS Top 25 list highlights input errors in all of the top 4 positions.

2.1Web Application Scanners &Firewalls

There are a number of possible approaches to mitigate the risk of injection vulnerabilities. According to Whitehat the use of Web Application Firewalls (WAF) has grown significantly in the past few years (Whitehat, 2011). WAFs provemost useful against injection attacks and it is estimated that a properly configured installation can mitigate over 70% of web app vulnerabilities. An excellent guide for selecting a WAF can be found in the Web Application SecurityConsortium's Web Application FirewallEvaluationCriteria (www.webappsec.org)

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 5, Special Issue 3, January 2018

Web application vulnerability scanners automate the process of identifying vulnerable systems, locating injection points and automating the exploit process. These can be a rapid way to test for XSS/SQLi vulnerability, but due to the wide variability in techniques used by these exploits, few tools will provide the comprehensive solution on their own. [3] discussed that the activity related status data will be communicated consistently and shared among drivers through VANETs keeping in mind the end goal to enhance driving security and solace. Along these lines, Vehicular specially appointed systems (VANETs) require safeguarding and secure information correspondences. Without the security and protection ensures, the aggressors could track their intrigued vehicles by gathering and breaking down their movement messages. A mysterious message confirmation is a basic prerequisite of VANETs. To conquer this issue, a protection safeguarding confirmation convention with expert traceability utilizing elliptic bend based chameleon hashing is proposed. Contrasted and existing plans Privacy saving confirmation utilizing Hash Message verification code, this approach has the accompanying better elements: common and unknown validation for vehicle-to-vehicle and vehicle-to-roadside interchanges, vehicle unlinkability, specialist following capacity and high computational effectiveness

III. FIXING THE CODE

Input validation is widely considered as the most effective mitigation technique against injection attacks (OWASP, 2011). Note, however, that no amount of input validation will defend against faulty business logic, poor authentication practices or other faults that can also be exploited via malicious (though perhaps not malformed) inputs.

Input validation is defined as the process of validating all the input to an application before using it (OWASP, 2012). It is ultimately futile to attempt to validate input in client-side code or the browser; these steps may raise the bar for attackers somewhat, but they are generally circumventable by even moderately skilled hackers. The client is under the full control of the user and all data to and from the web browser can be modified. Data verification code embedded in the page source could actually serve as a sign to potential hackers that you may have neglected to verify these items on the server side; an invitation to attempt exploitation. Proper input validation must be done on the server, outside of the user's control (Heiderich, M, et al, 2006).

3.1The PEAR Validate class

PHP is currently the most popular web app programming language and forms the base for some of the most used web applications from the past decade (WordPress, Joomla!, MediaWiki). The PEAR Validate class is a useful security measure for PHP-based sites. This is a good first step to ensure non-corrupted input to a form, but will not prevent XSS on it's own(Melonfire, 2006). It employs a library of simple REGEX against which the developer can compare valid inputs for a large number of standard, structured inputs: email addresses, dates, numbers, urls, etc. This is useful for strictly formatted fields like registration forms, but not for free-form fields like discussion forum systems etc. The PEAR HTML_Template_PHPTAL package is a templating engine for HTML that provides additional protection against XSS by facilitating well-formed outputs and escapes. There are a number of other templating engines that provide some added level of verification.

3.2Database APIs or templating systems

Similar in concept to the use of PEAR to prevent XSS, the addition of layers of abstraction when developing database applications removes some of the onus from the developer to have to manually escape all vulnerable variables. Template systems such as Django aim to reduce code complexity and automatically escape all special SQL parameters for most popular database servers (Django, 2012). Django is used by many large sites such as Pinterest and Instagram and provides comprehensive protection against SQLi/XSS and a host of other vulnerabilities.

3.3 Parameterized Queries /Prepared Statements

As a better alternative then attempting to escape each string of nasty characters, the use of prepared statements when querying a SQL database is widely accepted as the best practice. Prepared statements add a crucial layer of abstraction by strictly separating user-submitted data from SQL instructions generated (Shema, M,2010). Prepared statements are more robust and less prone to error then the alternative 'string concatenation' method of building database queries.

3.4 HTML Purifier

If the input you are accepting is HTML (e.g. comments or a bulletin board) the HTML Purifier library will remove all known malicious code, vastly reducing your exposure to XSS. The following table outlines the features of a number of HTML filtering libraries and shows that while there are many good initiatives freely available, HTML Purifier is among the best supported and fully featured .

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 5, Special Issue 3, January 2018

IV. CONCLUSION

4.1The way ahead

Storage will be reduced by 70% as compared to the existing technology. Much has been written and continues to be written on this topic. Organizations such as SANS and OWASP have made excellent inroads to raising awareness of the issues and developing actionable mitigation advice. Still, the problem is far from resolved and much work remains to be done. A greater understanding of the risks by leadership and developers alike can only lead to increased pressure to allow resources for adequate security to be built in and maintained.

4.2Leadership

Management must insist that both purchased closed-source applications and in-house developed ones are architected in a secure manner, with input vulnerability mitigation at the forefront. The use of standard web development frameworks and input validation libraries is a must; while they may not remove 100% of the risk, they will help pare down the simple programming errors that are often inadvertently introduced by human error. Once the platform has been built securely with controls in place to verify client input, additional value can be realize through the use of regular security audits, penetration tests and a web app firewall. Each of these will help elevate the bar in terms against vulnerabilities, known and unknown. Leaders must maintain sight that not all security vulnerabilities hold equal risk. The prevalence of exploitation via injection flaws should logically redirect additional resources towards its prevention (Whitehat, 2011).

4.3 Developers

Software coders and development leads must ensure they employ standard methods for building web-apps and strive not to sacrifice convenience for security, particularly with input validation and database communication. There will be errors in code; minimize the opportunity for these by using proven frameworks and input validation libraries. Complexity is the enemy of clean and secure code. Conductbasic penetration testing against your web app input fields using commercial or open-source tools. Familiarize yourself with resources OWASP's many resources for building secure web apps and incorporate them into your workflow.

REFERENCES

[1]Andreu, A. (2006). Professional pen testing for Web applications. Indianapolis, Ind.: Wiley Pub. CERT. (2012). Understanding Malicious Content Mitigation for Web Developers. Recoveredfrom:http://www.cert.org/tech_tips/maliciou s_code_mitigation.html

[2]Chen, S. (2012). Top 10:The Web Application Vulnerability Scanners Benchmark, 2012. Recovered from: http://sectooladdict.blogspot.ca/2012/07/2012-web-application-scannerbenchmark.html

[3] Christo Ananth, Dr.S. Selvakani, K. Vasumathi, "An Efficient Privacy Preservation in Vehicular Communications Using EC-Based Chameleon Hashing", Journal of Advanced Research in Dynamical and Control Systems, 15-Special Issue, December 2017,pp: 787-792.

[4]Scambray, J., Shema, M., &Sima, C. (2006). Hacking exposed: Web applications (2nd ed.). New York: McGraw-Hill.

[5]Shema, M. (2003).Hacknotes web security portable reference. New York: McGraw-Hill/Osborne.

[6]Shar, L.K, &HeeBeng, K.T. (2012). Defending against Cross-Site Scripting Attacks LwinKhinShar and ns Report.Recovered from: http://www.verizonbusiness.com/resources/r eports/rp_data-breach-investigations-report-2012_en_xg.pdf.

[7]Open Web Application Security Project. (2011). XSS Prevention Cheat Sheet, 2011. https://www.owasp.org/index.php/XSS_(Cr oss_Site_Scripting)_Prevention_ Cheat_Sheet.