IJCAT International Journal of Computing and Technology, Volume 1, Issue 2, March 2014
ISSN : 2348 - 6090
www.IJCAT.org

Study on Security Breaches in Php Applications

[1] Santosh Naidu P, [2] Amulya Patcha

[1] Computer Networks and Information Security

[2] Computer Networks and Information Security

Abstract **-** Php-based applications are one of the most dominant platforms for delivering information and services over Internet today. As they are mostly used for critical services, php-based applications become a common and direct target for security attacks. Although there are larger number of techniques have been developed to strengthen php-based applications and mitigate the attacks toward php-based applications, there is very slight effort committed to drawing connections among these techniques and building a big picture of php-based application security research.

This paper surveys the area of php-based application security, with the aim of systemizing the already implemented techniques into a picture that promotes future research. I present the unique aspects in the php-based application development which bring underlying challenges for building secured php-based applications. Finally, summarizes the lessons instructed and discuss future research opportunities in the area of php security.

Keywords **- ACMA, XSS, SQL Injection, Remote Inclusion.**

1. Introduction

World Wide Web has acquired from a system that delivers stable pages to a platform that supports distributed applications, known as web applications and become one of the most prevailing technologies for delivering the information and service over the Internet. The enhancing popularity of web application can be attributed to several factors, which includes remote accessibility, cross-platform compatibility, fast development, etc. Php which is a scripting language used to develop web applications enhances the user experiences of php-based applications are better interactive and responsive for the developers and programmers.

As php-based applications are mainly used to deliver security critical services, they become an indirect target for security attacks. Many web related applications interact with back-end systems like databases, which may store sensitive information, the compromise of php-based applications would result in violating an larger amount of information, leading to severe economic losses, ethical and legal consequences. Report is generated from Verizon [1] which shows that web applications now dominate in both the number of offends and the amount of data compromised. Current most commonly used web application development and frameworks, on the other hand, offer fixed security support. Thus securing web application is an error prostrate process and requires larger efforts, which could be impossible under time-to-market pressure and for people with insufficient security skills or awareness. As a result, very eminent percentages of web applications deployed over the Internet are exposed to security breaches. According to a report by the Web Application Security Consortium, about 49% of the web applications being reviewed contain vulnerabilities of high risk level and more than 13% of the websites can be compromised completely automatically [2]. A recent report [3] reveals that over 80% of the websites on the Internet have had at least one serious vulnerability.

Motivated by the urgent need for securing php-based applications, a substantial amount of research efforts have been devoted into this problem with a number of techniques developed for hardening php-based applications and mitigating the attacks.

In this paper, survey is done on the state of the art in php-based application security, with the target of systemizing the existing and also the unknown vulnerabilities

The rest of this paper's structure is as follows. At first illustration is done in three essential security properties that a secure web application should hold, as well as corresponding vulnerabilities and attack vectors in Section II. Then, illustration is done in the evolution of vulnerabilities in most widely used open source PHP web applications, and also SQL-Injection Security Evolution Analysis in PHP in Section III. In Section IV, discussed about ACMA (Access Control Model Analyzer), a model checking-based tool for the detection of access control vulnerabilities and also about RIPS - A static source code analyser for vulnerabilities in PHP scripts .In Section V, discussion is carried out about Realistic Vulnerability Injections in PHP Web Applications and securing PHP Based Web Application Using Vulnerability Injection .In Section VI, will be mentioning some of the vulnerabilities that affect php-

based applications. Then, in Section VII, concludes with future directions for web application security and finally Section VIII ends with paper contributions.

2. Understand WEB Application Security Properties, Vulnerabilities and Attack Vectors

A secure web application has to satisfy desired security properties under the given threat model. In the area of web application security, the following threat model is usually considered: 1) the web application itself is benign (i.e., not hosted or owned for malicious purposes) and hosted on a trusted and hardened infrastructure (i.e., the trust computing base, including OS, web server, interpreter, etc.); 2) the attacker is able to manipulate either the contents or the sequence of web requests sent to the web application, but cannot directly compromise the infrastructure or the application code. The vulnerabilities within web application implementations may violate the intended security properties and allow for corresponding successful exploits.

In particular, a secure web application should preserve the following stack of security properties, as shown in Fig. 1. Input validity means the user input should be validated before it can be utilized by the web application; state integrity means the application state should be kept untempered; logic correctness means the application logic should be executed correctly as intended by the developers. The above three security properties are related in a way that failure in preserving a security property at the lower level will affect the assurance of the security property at a higher level. For instance, if the web application fails to hold the input validity property, a crosstie scripting attack can be launched by the attacker to steal the victim's session cookie. Then, the attacker can hijack and tamper the victim's web session, resulting in the violation of state integrity property. In the following sections, described the three security properties and show how the unique features of web application development complicate the security design for web applications.

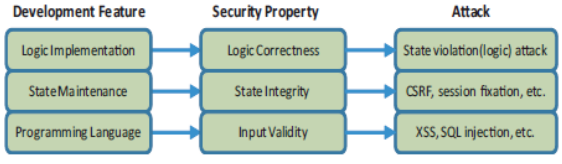

Figure 1: Pixy architecture plus modifications for Injection Tool

A. Input Validity

Given the threat model, user input data cannot be trusted. However, for the untrusted user data to be used in the application (e.g., composing web response or SQL queries), they have to be first validated. Thus, referred to this security property as input validity property: All the user input should be validated correctly to ensure it is utilized by the web application in the intended way. The user input validation is often performed via sanitization routines, which transform untrusted user input into trusted data by filtering suspicious characters or constructs within user input. While simple in principle, it is non-trivial to achieve the completeness and correctness of user input sanitization, especially when the web application is programmed using scripting languages. First, since user input data is propagated throughout the application, it has to be tracked all the way to identify all the sanitization points. However, the dynamic features of scripting languages have to be handled appropriately to ensure the correct tracking of user input data. Second, correct sanitization has to take into account the context, which specifies how the user input is utilized by the application and interpreted later either by the web browser or the SQL interpreter. Thus different contexts require distinct sanitization functions. However, the weak typing feature of programming languages makes context-sensitive sanitization challenging and error-prone.

In current web development practices, sanitization routines are usually placed by developers manually in an ad-hoc way, which can be either incomplete or erroneous, and thus introduce vulnerabilities into the web application. Missing sanitization allows malicious user input to flow into trusted web contents without validation; faulty sanitization allows malicious user input to bypass the validation procedure. A web application with the above vulnerabilities fails to achieve the input validity property, thus is vulnerable to a class of attacks, which are referred to as script injections, dataflow attacks or input validation attacks. This type of attacks embed malicious contents within web requests, which are utilized by the web application and executed later. Examples of input validation attacks include cross-site scripting (XSS), SQL injection, directory traversal, filename inclusion, response splitting, etc. They are distinguished by the locations where malicious contents get executed. In the following, illustration is done on the most two popular input validation attacks.

1) SQL Injection: A SQL injection attack is successfully launched when malicious contents within user input flow into SQL queries without correct validation. The database trusts the web application and executes all the queries issued by the application. Using this attack, the attacker is able to embed SQL keywords or operators within user input to manipulate the SQL query structure and result in unintended execution. Consequences of SQL injections include authentication bypass, information disclosure and even the destruction of the entire database. Interested reader can refer to [7] for more details about SQL injection.

2) Cross-Site Scripting: A cross-site scripting (XSS) attack is successfully launched when malicious contents within user input flow into web responses without correct validation. The web browser interprets all the web responses returned by the trusted web application (according to the same-origin policy) Using this attack, the attacker is able to inject malicious scripts into web responses, which get executed within the victim's web browser. The most common consequence of XSS is the disclosure of sensitive information, e.g., session cookie theft. XSS usually serves as the first step that enables further sophisticated attacks (e.g., the notorious MySpace Samy worm [8]). There are several variants of XSS, according to how the malicious scripts are injected, including stored/persistent XSS (malicious scripts are injected into persistent storage), reflected XSS, DOM-based XSS, content-sniffing XSS [9], etc.

B. State Integrity

State maintenance is the basis for building stateful web applications, which requires a secure web application to preserve the integrity of application states. However, The involvement of an untrusted party (client) in the application state maintenance makes the assurance of state integrity a challenging issue for web applications. A number of attack vectors target the vulnerabilities within session management and state maintenance mechanisms of web applications, including cookie poisoning (tampering the cookie information), session fixation (when the session identifier is predictable), session hijacking (when the session identifier is stolen), etc.

Cross-site request forgery (i.e., session riding) is a popular attack that falls in this category. In this attack, the attacker tricks the victim into sending crafted web requests with the victim's valid session identifier, however, on the attacker's behalf. This could result in the victim's session being tampered, sensitive information disclosed (e.g., [10]), financial losses (e.g., an attacker may forge a web request that instructs a vulnerable banking website to transfer the victim's money to his account), etc. To preserve state integrity, a number of effective techniques have been proposed [11].

Client-side state information can be protected by integrity verification through MAC (Message Authentication Code). Session identifiers need to be generated with high randomness (to defend against session fixation) and transmitted over secure SSL protocol (against session hijacking). To mitigate CSRF attacks, web requests can be validated by checking headers (Referrer header, or Origin header [12]) or associated unique secret tokens (e.g., NoForge [13], RequestRodeo [14], BEAP [15]). Since the methods of preserving state integrity are relatively mature, thus falling beyond the scope of this survey.

C. Logic Correctness

Ensuring logic correctness is key to the functioning of web applications. Since the application logic is specific to each web application, it is impossible to cover all the aspects by one description. Instead, a general description that covers most common application functionalities is given as follows, which referred to as logic correctness property: Users can only access authorized information and operations and are enforced to follow the intended workflow provided by the web application.

To implement and enforce application logic correctly can be challenging due to its state maintenance mechanism and "decentralized" structure of web applications. First, interface hiding technique, which follows the principle of "security by obscurity", is obviously deficient in nature, which allows the attacker to uncover hidden links and directly access unauthorized information or operations or violate the intended workflow. Second, explicit checking of the application state is performed by developers manually and in an ad-hoc way. Thus, it is very likely that certain state checks are missing on unexpected control flow paths, due to those many entry points of the web application. Moreover, writing correct state checks can be error-prone, since not only static security policies but also dynamic state information should be considered. Both missing and faulty state checks introduce logic vulnerabilities into web applications.

A web application with logic flaws is vulnerable to a class of attacks, which are usually referred to as logic attacks or state violation attacks. Since the application logic is specific to each web application, logic attacks are also idiosyncratic to their specific targets. Several attack vectors that fall (or partly) within this category include authentication bypass, parameter tampering, forceful browsing, etc. There are also application specific logic attack vectors. For example, a vulnerable ecommerce website may allow a same coupon to be applied multiple times, which can be exploited by the attacker to reduce his payment amount.

3. Evolution of Php Web Application Security and SQL Injection Security Evolution Analysis in Php

Web applications are mainly subjected to massive attacks, with vulnerabilities found easily in both open source and commercial applications as showed by the fact that approximately maximum number of vulnerabilities are found in web applications. And also investigate whether complexity metrics or a security resources indicator (SRI) metric can be used to identify vulnerable web application showing that average cyclomatic complexity is an effective predictor of

vulnerability for several applications, especially for those with low SRI scores.[16]

Web sites are often a mixture of static sites and programmes that integrate comparative databases as a back-end. Software that implements Web sites continuously evolve, newer versions of programs, interactions and functionalities are summed up and existing ones are removed or modified. Websites require configuration and programming attention to assure security, confidentiality, and trustiness of the promulgated information. During evolution of Web software, from one version to the next one, security breaches may be introduced, corrected, or ignored. [17]

4. ACMA - A Model Checking-Based Tool for Detection of Access Control Vulnerabilities in and Rips

Access control vulnerabilities in php-based web applications are on the advance. In 2010 "Top 10 Most Critical Web Applications Security Risks", the OWASP reported that the prevalence of access control vulnerabilities in web applications increased compared to 2007. However, in counterpoint to SQL injection and cross-site scripting breaches, access control vulnerabilities comparatively received very less attention from the research area. One of the main challenges for the detection of access control vulnerabilities lies in the identification of pages that should have a restricted access. Recently,[18] some researchers [4], observed that privileged pages are rarely left entirely unprotected. They argue that access control vulnerabilities usually occur because of "hidden" execution paths that lack access control checks rather than because of a complete absence of access control. For example, web applications will typically hide links to privileged pages from unprivileged users. While this is a good practice, it is not sufficient to ensure security, as a malicious user can guess the URL of the hidden page and access the privileged information through this "hidden" path. In the context of this paper, we refer to this kind of access control vulnerabilities as "forced browsing" vulnerabilities. Forced browsing vulnerabilities are a subset of "hidden" paths vulnerabilities in access control models. The term was introduced by Sun et al. [5] as they were the first to tackle the automatic identification of such security flaws.

In order to contain the risks of vulnerable web applications source code has to be reviewed by the developer or by penetration testers. Mentioned in the fact that many applications can have thousands of codelines and time is determined by costs, a manual source code review might be incomplete. Tools can help penetration testers to minimize time and costs by automating time intense processes while reviewing a source code. the concept of web [6] application vulnerabilities is introduced and how they can be detected by static source code analysis automatically. Also a tool was developed named RIPS is introduced which automates the process of identifying potential security breaches in PHP source code. RIPS is open source and freely available at source forge website. The result of the analysis can easily be reexamined by the penetration tester in its circumstance without reviewing the whole source code again. [19]

5. Realistic Vulnerability Injections in Php Web Applications and Securing using Vulnerability Injections

Vulnerability injection is a field that has received relatively little attention by the research community, probably because its objective is apparently contrary to the purpose of making applications more secure. It can however be used in a variety of areas that make it worthy of research, such as the automatic creation of educational examples of vulnerable code, testing defense in depth mechanisms, and the evaluation of both vulnerability scanners and security teams.A prototype implementing the architecture was developed and evaluated to analyze the feasibility of the architecture. The prototype targets PHP web applications.[20]

This study shows that the vulnerabilities that the prototype is able to inject in applications which results in maximum number of vulnerabilities that appear in PHP-based web applications. Finally, several applications were used in the evaluation and were subjected to injections using the prototype, after which they were analysed by hand to see whether a vulnerability breach was raised or not. The results show that the prototype can not only inject a great amount of vulnerabilities but that they are actually attackable. The architecture from Figure 2[21] shows the original Pixy architecture in dark with the modifications in red.

Figure 2: Pixy architecture plus modifications for Injection Tool

6. Vulnerabilities that Affect Php-Based Applications

Table 1: Most common vulnerabilities in php-based applications

1	Cross Site Scripting vulnerabilities
2	Security Bypass vulnerabilities
3	Multiple Directory Traversal vulnerabilities
4	Remote Command Injection vulnerabilities
5	Parameter Cross Site Scripting vulnerability
6	Local File Inclusion vulnerabilities
7	SQL Injection vulnerabilities
8	Webtester multiple vulnerabilities
9	Remote File Inclusion vulnerabilities
10	Database Injection vulnerabilities

7. Future Directions of Php-Based Application Security

This paper provided a comprehensive survey of recent research papers in the area of php-based application security and described unique characteristics of php-based application development, identified important security properties that secure applications should preserve and also pointed out several open issues that still need to be addressed. After surveying several papers about vulnerabilities in php, I would like to identify more unknown vulnerabilities that affect the php applications and also counter-measures to eradicate them as a future direction of approach in securing php-based applications.

References

[1] Verizon 2010 Data Breach Investigations Report, http://www.verizonbusiness.com/resources/reports/rp2010-databreach- report en xg.pdf.

[2] Web Application Security Statistics, "http://projects.webappsec.org/w/page/13246989/Web Application SecurityStatistics."

[3] WhiteHatSecurity,"WhiteHat website security statistic report 2010."

[4] F. Sun, L. Xu, and Z. Su, "Static detection of access control vulnerabilities in web applications," in USENIX Security, 2011, pp. 155–170.

[5] S. Son, K. McKinley, and V. Shmatikov, "Rolecast: finding missing security checks when you do not know what checks are," in OOPSLA '11. ACM, 2011, pp. 1069–1084.

[6] MySpace Samy Worm, http://namb.la/popular/tech.html,, 2005.

[7] W. G. Halfond, J. Viegas, and A. Orso, "A Classification of SQLInjection Attacks and Countermeasures," in Proc. of the International Symposium on Secure Software Engineering, March 2006.

[8] MySpace Samy Worm, http://namb.la/popular/tech.html, 2005.

[9] A. Barth, J. Caballero, and D. Song, "Secure content sniffing for web browsers, or how to stop papers from reviewing themselves," in Oakland'09: Proceedings of the 30th IEEE Symposium on Security and Privacy, 2009, pp. 360–371.

[10] Gmail CSRF Security Flaw, "http://ajaxian.com/archives/gmail-csrfsecurity- flaw," 2007.

[11] M. Johns, "Sessionsafe: Implementing xss immune session handling,"in ESORICS'06: Proceedings of the 11th European Symposium On Research In Computer Security, 2006.

[12] A. Barth, C. Jackson, and J. C. Mitchell, "Robust defenses for cross-site request forgery," in CCS'08: Proceedings of the 15th ACM conference on Computer and communications security, 2008, pp. 75–88.

[13] N. Jovanovic, E. Kirda, and C. Kruegel, "Preventing cross site request forgery attacks," in SecureComm'06: 2nd International Conference on Security and Privacy in Communication Networks, 2006, pp. 1 –10.

[14] M. Johons and J. Winter, "Requestrodeo: Client-side protection against session riding," in OWASP AppSec Europe, 2006.

[15] Z. Mao, N. Li, and I. Molloy, "Defeating cross-site request forgery attacks with browser-enforced authenticity protection," in FC'09: 13 th International Conference on Financial Cryptography and Data Security, 2009, pp. 238–255.

[16] An Empirical Study of the Evolution of PHP Web Application Security By Maureen Doyle, James Walden (2012 Third International Workshop on Security Measurements and Metrics).

[17] SQL-Injection Security Evolution Analysis in PHP By Ettore Merlo*, Dominic Letarte, Giuliano Antoniol.

[18] Fast Detection of Access Control Vulnerabilities in PHP Applications By Franc¸ois Gauthier, Ettore Merlo (2012 19th Working Conference on Reverse Engineering).

[19] RIPS - A static source code analyser for vulnerabilities in PHP scripts By Johannes Dahse, 2010.

[20] Realistic Vulnerability Injections in PHP Web Applications By Francisco José Marques Vieira, 2011.

[21] Securing PHP Based Web Application Using Vulnerability Injection By Jamang Jayeshbha Bhalabha, 2013