Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 3, Issue. 10, October 2014, pg.349 – 355

RESEARCH ARTICLE

WEB APPLICATION SECURITY

RAVINDER YADAV[1], AAKASH GOYAL[2]
[1]CSE DEPARTMENT, DRONACHARYA COLLEGE OF ENGINEERING
GURGAON, HARYANA, INDIA
[2]CSE DEPARTMENT, DRONACHARYA COLLEGE OF ENGINEERING
GURGAON, HARYANA, INDIA

ravinderyadav2077@gmail.com, geniousdon11@gmail.com

Abstract — Web application security is a branch of Information Security that deals specifically with security of websites, web applications and web services. Simply, Web Application Security is "The securing of web applications". Web applications are one of the most prevalent platforms for information and services delivery over Internet today. As they are increasingly used for critical services, web applications become a popular and valuable target for security attacks. Although a large body of techniques have been developed to fortify web applications and mitigate the attacks toward web applications, there is little effort devoted to drawing connections among these techniques and building a big picture of web application security research.Web applications are important, common distributed systems whose current security relies primarily on server-side mechanisms. Web applications provide end users with client access to server functionality through a set of Web pages. These pages often contain script code to be executed dynamically within the client Web browser. Most Web applications aim to enforce simple, intuitive security policies, such as, for Web-based email, disallowing any scripts in untrusted email messages. Even so, Web applications are currently subject to a plethora of successful attacks, such as cross-site scripting, cookie theft, session riding, browser hijacking, and the recent self-propagating worms in Web-based email and social networking sites .This paper surveys the area of web application security, with the aim of systematizing the existing techniques into a big picture that promotes future research.

I. INTRODUCTION

Security is a critical part of your Web applications. Web applications by definition allow users access to a central resource — the Web server — and through it, to others such as database servers. By understanding and implementing proper security measures, you guard your own resources as well as provide a secure environment in which your users are comfortable working with your application.World Wide Web has evolved from a system that delivers static pages to a platform that supports distributed applications, known as web applications and become one of the most prevalent technologies for information and service delivery over Internet. The increasing popularity of web application can be attributed to several factors, including remote accessibility, cross-platform compatibility, fast development, etc. The AJAX (Asynchronous JavaScript and XML) technology also enhances the user experiences of web applications with better interactiveness and responsiveness.

As web applications are increasingly used to deliver security critical services, they become a valuable target for security attacks. Many web applications interact with back-end database systems, which may store sensitive information (e.g., financial, health),

the compromise of web applications would result in breaching an enormous amount of information, leading to severe economical losses, ethical and legal consequences.The Web platform is a complex ecosystem composed of a large number of components and technologies, including HTTP protocol, web server and server-side application development technologies (e.g., CGI, PHP, ASP), web browser and client-side technologies (e.g., JavaScript, Flash). Web application built and hosted upon such a complex infrastructure faces inherent challenges posed by the features of those components and technologies and the inconsistencies among them. Current widely-used web application development and testing frameworks, on the other hand, offer limited security support. Thus secure web application development is an errorprone process and requires substantial efforts, which could be unrealistic under time-to-market pressure and for people with insufficient security skills or awareness. As a result, a high percentage of web applications deployed on the Internet are exposed to security vulnerabilities. Motivated by the urgent need for securing web applications, a substantial amount of research efforts have been devoted into this problem with a number of techniques developed for hardening web applications and mitigating the attacks. Many of these techniques make assumptions on the web technologies used in the application development and only address one particular type of security flaws; their prototypes are often implemented and evaluated on limited platforms.

II. HOW A WEB APPLICATION WORKS

Web application is a distributed application that is executed over the Web platform. It is an integral part of today's Web ecosystem that enables dynamic information and service delivery. As shown in Fig. 1, a web application may consist of code on both the server side and the client side. The server side code will generate dynamic HTML pages either through execution (e.g., Java servlet, CGI) or interpretation (e.g., PHP, JSP). During the execution of the server-side code, the web application may interact with local file system or back-end database for storing and retrieving data. The client-side code (e.g., in JavaScript) are embedded in the HTML pages, which is executed within the browser. It can communicate with the server-side code (i.e., AJAX) and dynamically updates the HTML pages. In what follows, we describe three unique aspects of the web application development, which differentiate web applications from traditional applications.

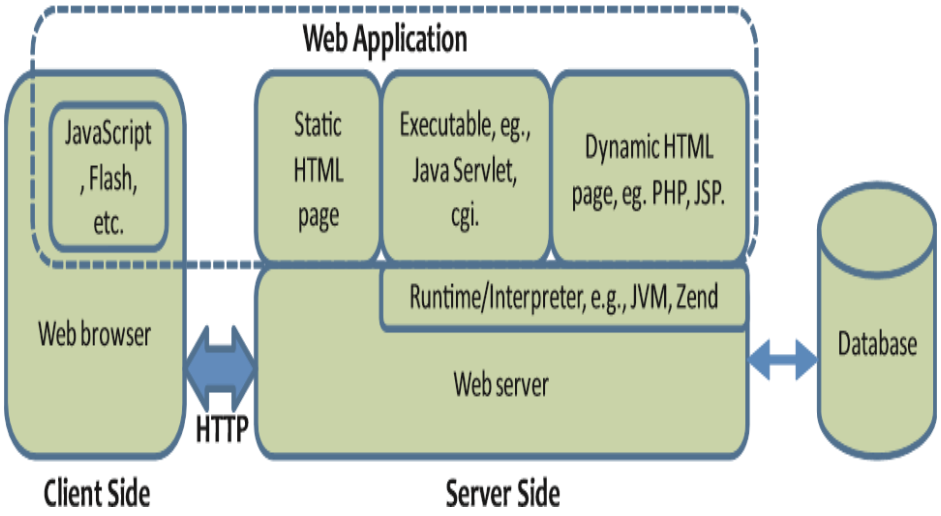


Fig. 1. Overview of Web Application

A. Programming Language

Web application development relies on web programming languages. These languages include scripting languages that are designed specifically for web (e.g., PHP, JavaScript) and extended traditional general-purpose programming languages (e.g., JSP).

B. State Maintenance

HTTP protocol is stateless, where each web request is independent of each other. However, to implement non-trivial functionalities, "stateful" web applications need to be built on top of this stateless infrastructure. The state of a web session records the conditions from the historical web requests that will affect the future execution of the web application. The session state can be maintained either at the client side (via cookie, hidden form or URL rewriting) or at the server side.

C. Logic Implementation

The business logic defines the functionality of a web application, which is specific to each application. Such a functionality is manifested as an intended application control flow and is usually integrated with the navigation links of a web application.A web application is usually implemented as a number of independent modules, each of which can be directly accessed in any order by a user. This unique feature of web applications significantly complicates the enforcement of the application's control flow across different modules.

III. 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. 2.*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 untampered; *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 cross site 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, we describe the three security properties and show how the unique features of web application development complicate the security design for web applications.

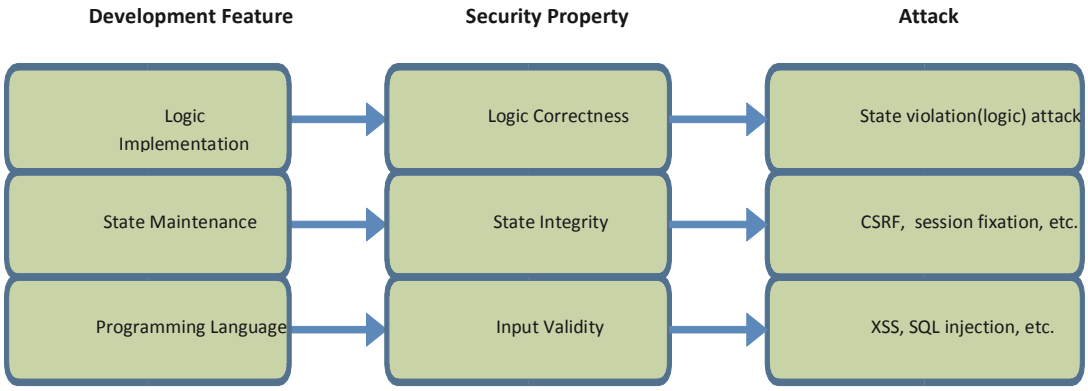


Fig. 2. Web Application Security Properties

A. Input Validity

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. 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 embeds 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, we illustrate 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.

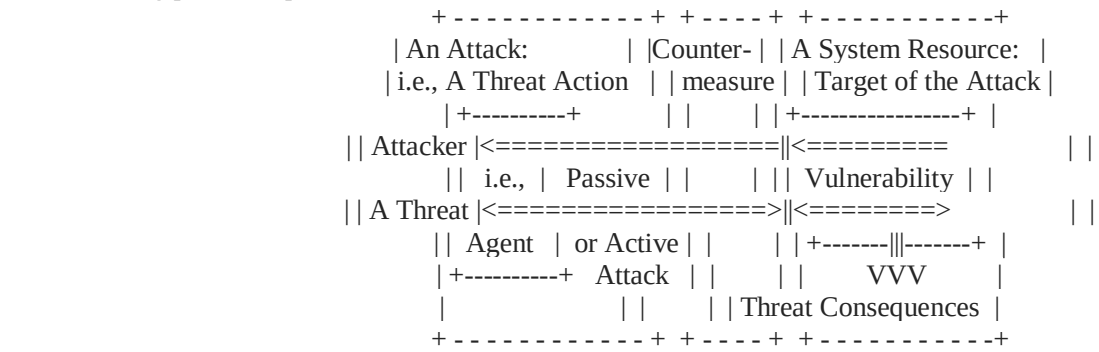
C. Logic Correctness

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 we refer 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.

IV. COUNTER MEASURES

A **countermeasure** is an action, device, procedure, or technique that reduces a threat, a vulnerability, or an attack by eliminating or preventing it, by minimizing the harm it can cause, or by discovering and reporting it so that corrective action can be taken. The meaning of countermeasure is: The deployment of a set of security services to protect against a security threat.

The following picture explains the relationships between these concepts and terms:



A resource (both physical or logical) can have one or more vulnerabilities that can be exploited by a threat agent in a threat action. The attack can be active when it attempts to alter system resources or affect their operation: so it compromises Integrity or Availability. A "passive attack" attempts to learn or make use of information from the system but does not affect system resources: so it compromises Confidentiality. A large number of countermeasures have been developed to secure web applications and defend against the attacks towards web applications. These methods address one or more security properties and instantiate them into concrete security specifications/policies (either explicitly or implicitly) that are to be enforced at different phases in the lifecycle of web applications. We organize existing countermeasures along two dimensions. The first dimension is the security property that these techniques address. The second dimension is their design principle, which we outline as the following three classes:

(1) Security by construction: this class of techniques aim to construct secure web applications, ensuring that no potential vulnerabilities exist within the applications. Thus, the desired security property is preserved and all corresponding exploits would fail. They usually design new web programming languages or frameworks that are built with security mechanisms, which automatically enforce the desired security properties. These techniques solve security problems from the root and thus are most robust. However, they are most suitable for new web application development. Rewriting the huge number of legacy applications can be unrealistic.

(2) Security by verification: this class of techniques aim to verify if the desired security properties hold for a web application and identify potential vulnerabilities within the application. This procedure is also referred to as vulnerability analysis. Efforts have to be then spent to harden the vulnerable web application by fixing the vulnerabilities and retrofitting the application either manually or automatically. Techniques within this class can be applied to both new and legacy web applications.

Existing program analysis and testing techniques are usually adopted by the works from this class. They have to be overcome a number of technical difficulties in order to achieve the completeness and correctness of vulnerability analysis. In particular, program analysis involves static analysis (i.e., code auditing/review performed on the source code without execution) and dynamic analysis (i.e., observing runtime behavior through execution). Static analysis tends to be complete at identifying all potential vulnerabilities, however, with the price of introducing false alerts. On the other hand, dynamic analysis guarantees the correctness of identified vulnerabilities within explored space, but cannot assure the completeness. Program testing focuses on generating concrete attack vectors that expose expected vulnerabilities within the web application. Similar to dynamic analysis, it also faces the inherent challenge of addressing completeness.

(3) Security by protection: this class of techniques aims to protect a potentially vulnerable web application against exploits by building a runtime environment that supports its secure execution. They usually either 1) place safeguards (i.e., proxy) that separate the web application from other components in the Web ecosystem, or 2) instrument the infrastructure components (i.e., language runtime, web browser, etc.) to monitor its runtime behavior and identify/quarantine potential exploits.

Property/Technique	Input Validity Property		Logic Correctness Property	
Security by Construction		[17],[19],[20],[24],[81]		[17],[81],[83],[84], [85]
Security by Verification	Program analysis	static:[25],[26],[27],[28], [29],[30],[31],[32],[33] dynamic: [34],[35] hybrid: [36],[37]	Static analysis	[87],[88],[89],[90]
	Program testing	[38],[39],[40],[41],[42],[43]	Dynamic analysis	[91],[92],[93]
Security by Protection	Taint-based protection	[44],[45],[47],[48],[49],[58], [60],[59],[61]		[94],[95],[96],[97],[98],[99]
	Taint-free protection	[62],[63],[64],[67],[68], [69],[71],[72],[75]		

Fig. 3. Summary of existing techniques

V. CONCLUSION

Web applications reach out to a larger, less-trusted user base than legacy client-server applications, and yet they are more vulnerable to attacks. Many companies are starting to take initiatives to prevent these types of break-ins. Code reviews, extensive penetration testing, and intrusion detection systems are just a few ways that companies are battling a growing problem. Unfortunately, most of the solutions available today are using negative security logic (working with a list of attacks and trying to prevent against them). Negative security logic solutions can prevent known, generalized attacks, but are ineffective against the kind of targeted, malicious hacker activity.

So, for WEB APPLICATION SECURITY ,Keep server and third-party applications and library up-to-date, Do not trust user input, Review code & design and identify possible weaknesses and Monitor run-time activity to detect ongoing attacks/probes.

REFERENCES

1. Wikipedia "Web Application Security".
2. Xiaowei Li and Yuan Xue,Department of Electrical Engineering and Computer ScienceVanderbilt University xiaowei.li, yuan.xue@vanderbilt.edu "A Survey on Web Application Security".
3. Ashwini garg ,Shekhar singh - *Assistant Professors, Deptt. Of CSE Panipat Institute of Engg. & Technology, Samalkha "A Review on Web Application Security Vulnerabilities".*
4. J. Bau and J. C. Mitchell, "Security modeling and analysis," *IEEE Security & Privacy*, vol. 9, no. 3, pp. 18–25, 2011.
5. H. J. Wang, C. Grier, A. Moshchuk, S. T. King, P. Choudhury, and H. Venter, "The multi-principal os construction of the gazelle web browser," in *USENIX'09: Proceedings of the 18th conference on USENIX security symposium*, 2009, pp. 417–432.
6. S. Tang, H. Mai, and S. T. King, "Trust and protection in the illinois browser operating system," in *OSDI'10: Proceedings of the 9 th USENIX conference on Operating systems design and implementation*, 2010, pp. 1 – 8.
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. M. Cova, V. Felmetsger, and G. Vigna, "Vulnerability Analysis of Web Applications," in *Testing and Analysis of Web Services*, L. Baresi and E. Dinitto, Eds. Springer, 2007.
17. S. Chong, K. Vikram, and A. C. Myers, "Sif: Enforcing confidentiality and integrity in web applications," in *USENIX'07: Proceedings of the 16th conference on USENIX security symposium*, 2007.
18. L. Z. Andrew C. Myers, "Jif: Java information flow." [Online].
19. Available: http://www.cs.cornell.edu/jif
20. S. Chong, J. Liu, A. C. Myers, X. Qi, K. Vikram, L. Zheng, and X. Zheng, "Secure web applications via automatic partitioning," in *SOSP '07: Proceedings of the 21st ACM SIGOPS symposium on*
21. *Operating systems principles*, 2007, pp. 31–44.
22. W. Robertson and G. Vigna, "Static enforcement of web application integrity through strong typing," in *USENIX'09: Proceedings of the 18th conference on USENIX security symposium*, 2009, pp. 283–298.
23. H. Fisk., "Prepared Statements," 2004. [Online]. Available: http://dev.mysql.com/tech-resources/articles/4.1/preparedstatements.html

24. R. A. McClure and I. H. Kruger,¨ "Sql dom: compile time checking of dynamic sql statements," in *ICSE'05: Proceedings of the 27 th international conference on Software engineering*, 2005, pp. 88–96.
25. Verizon 2010 Data Breach Investigations Report, "http://www.verizonbusiness.com/resources/reports/rp 2010databreach-report en xg.pdf."
26. Web Application Security Statistics, "http://projects.webappsec.org/w/page/13246989/Web Application Security Statistics."
27. WhiteHat Security, "WhiteHat website security statistic report 2010."

