Web Security

Pramod Kumar & Ruchi Yadav
Dept. of Information & Technology, Dronacharya College of Engineering
Farruhknagar, Gurgaon, India
Email: ruchiyadav477@gmail.com

Abstract
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 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. 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. We first present the unique aspects in the web application development which bring inherent challenges for building secure web applications. Then we identify three essential security properties that a web application should preserve: input validity, state integrity and logic correctness, and describe the corresponding vulnerabilities that violate these properties along with the attack vectors that exploit these vulnerabilities. We organize the existing research works on securing web applications into three categories based on their design philosophy: security by construction, security by verification and security by protection. Finally, we summarize the lessons learnt and discuss future research opportunities in this area.

I. INTRODUCTION

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. A breach report from Verizon [1] shows that web applications now reign supreme in both the number of breaches and the amount of data compromised. 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. 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 . A recent report reveals that over 80% of the websites on the Internet have had at least one serious vulnerability. 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. A practitioner may wonder whether these techniques are suitable for their scenarios. And if they can not be directly applied, whether these techniques can be extended and/or combined. Thus, it is desirable and urgent to provide a systematic framework for exploring the root causes of web application vulnerabilities, uncovering the connection between the existing techniques, and sketching a big picture of current research frontier in this area. Such a framework would help both new and experienced researcher to better understand web application security challenges and assess existing defenses, and inspire them with new ideas and trends. In this paper, we survey the state of the art in web application security, with the aim of systematizing the existing techniques into a big picture that promotes future research. Based on the conceptual security framework by Bau and Mitchell, we organize our survey using three components for assessing the security of a web application (or equipped with a defense mechanism): system model, threat model and security property. System model describes how a web application works and its unique characteristics; threat model. describes the power and resources attackers possess; security property defines the aspect of the web application behavior intended by the developers. Given a threat model, if one web application fails to preserve certain security property under all scenarios, this application is insecure or vulnerable to corresponding attacks. This survey covers the techniques which consider the following threat model:

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*.

We note here that although browser security ([5], [6]) is also an essential component in end-to-end web application security, research works on this topic usually have a different threat model, where web applications are considered as potentially malicious. This survey does not include the research works on browser security so that it can focus on the problem of building secure web applications and protecting vulnerable ones. The contributions of this paper are:

(1) We present three aspects in web application development, which poses inherent challenges for building secure web applications, and identify three levels of security properties that a secure web application should hold: *input validity*, *state integrity* and *logic correctness*. Failure of web applications to fulfill the above security properties is the root cause of corresponding vulnerabilities, which allow for successful exploits.

(2)We classify existing research works into three categories: *security by construction*, *security by verification* and *security by protection*, based on their design principle (i.e., constructing vulnerability-free web applications, identifying and fixing vulnerabilities, or protecting vulnerable web applications against exploits at runtime, respectively) and how security properties are assured at different phases in the life cycle of web application. We are not trying to enumerate all the existing works but have covered most of the represented works.

(3) We identify several open issues that are insufficiently addressed in the existing literature. We also discuss future research opportunities in the area of web application security and the new challenges that are expected ahead. We structure the rest of this paper as follows. We first describe how a web application works and its unique characteristics in Section II. Then, we illustrate three essential security roperties that a secure web application should hold, as well as corresponding vulnerabilities and attack vectors in Section III. In Section IV, we categorize and illustrate the state-of-theart of proposed techniques systematically. Then, in Section V, we discuss future directions for web application security. We conclude our survey paper in Section VI.

II. UNDERSTAND 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 serverside 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.

*

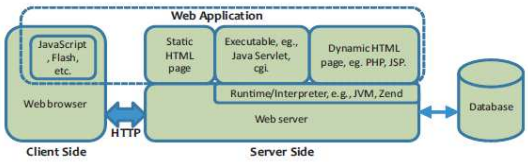

Fig. 1. Overview of Web Application

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). A distinguishing feature of many web programming languages is their *type systems*. For example, some scripting languages (e.g., PHP) are *dynamically typed*, which means that the type of a variable is determined at runtime, instead of compile time. Some languages (e.g., JavaScript) are *weakly typed*, which means that a statement or a function can be performed on a variety of data types via implicit type casting. Such type systems allow developers to blend several types of constructs in one file for runtime interpretation. For instance, a PHP file may contain both static HTML tags and PHP functions and a web page may embed executable JavaScript code. The representation of application data and code by an unstructured sequence of bytes is a unique feature of web application that helps enhance the development efficiency.

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. Thus, the abstraction of web session is adopted to help the web application to identify and correlate a series of web requests from the same user during a certain period of time. 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. In the latter case, a unique identifier (session ID) is defined to index the explicit session variables stored at the server side and issued to the client. For example, most of web programming languages (e.g., PHP, JSP) offer developers a collection of functions for managing the web session. For example, in PHP, *session start()* can be called to initialize a web session and a pre-defined global array $ *SESSION* is employed to contain the session state. In either case, the client plays a vital role in maintaining the states of a web application.

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. For example, authentication and authorization are a common part of the control flow in many web applications, through which an web application restricts its sensitive information and privileged operations from unauthorized users. As another example, e-commerce websites usually manage the sequence of operations that the customers need perform during shopping and checkout. 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. This task needs to be performed through a tight collaboration of two approaches. The first approach, which is practiced by most web applications, is interface hiding, where only accessible resources and actions of the web application are presented as web links and exposed to users. The second approach requires explicit checks of the application state, which is maintained by session variables (or persistent objects in the database), before sensitive information and operations can be accessed.

III. 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. 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 crosssite 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.

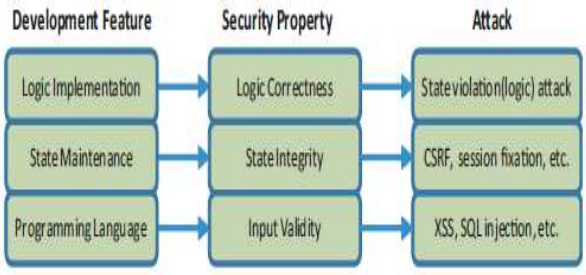

Fig. 2. Web Application Security Properties

A. Input Validity

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, we refer 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, 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 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). 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, 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, 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. 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 eader) or associated unique secret tokens (e.g., NoForge , Request Rodeo, BEAP). 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 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. CATEGORIZE EXISTING COUNTERMEASURES

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 aim 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. These techniques can be independent of programming languages or platforms, thus scale well. However, runtime performance overhead is inevitably introduced. Compared to a previous survey which only focuses on vulnerability analysis, this survey is more comprehensive and covers the complete lifecyle of a web application, from development, auditing/testing to deployment. For each individual technique, we identify its unique strengthes and limitations, compared with other techniques. We also discuss open issues that remain insufficiently addressed. Fig. 3 shows a summary of existing techniques we have covered.

A. Input Validity

We first recall the 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 root cause for input validation vulnerabilities is that untrusted user input data flows into trusted web contents without sufficient and correct validation, which is an instance of insecure information flow. Thus, the information flow security model can be naturally applied into addressing the input validity property, which we refer to as *information flow (taint propagation) specification.* This specification is modeled as follows in web applications. First, user input data is marked as tainted at entry points (i.e., *sources*) of the web application. for composing SQL queries or web responses), it has to be validated and becomes untainted. If the above specification is not enforced, the web application has input validation vulnerability. To enforce the information flow specification, three tasks have to be performed:

(1) *user input identification*, which requires all the untrusted user data to be reliably identified and separated from the trusted web contents;

(2) *user input tracking*, which requires the user data to be reliably recognized throughout its flow within the application at a certain granularity;

(3) *user input handling*, which requires the user data to be correctly handled, and thus utilized by the application in a secure way. In practice, user input identification and tracking can be achieved through strong typing, variable/byte tainting and tracking, etc. There are two general approaches for handling untrusted user input. One is to transform it into trusted data by sanitization routines (i.e., *sanitizers*), which are usually regarded as a black-box; the other is to quarantine it based on certain predefined security policies, so that potentially malicious user input cannot be executed and the structure integrity of web contents (e.g., web pages or SQL queries) is preserved. Although the latter approach requires certain manual intervention for specifying security policy, it circumvents reasoning about the correctness of sanitization routine, which can be challenging due to its context-sensitiveness

V. CONCLUSION

This paper provided a comprehensive survey of recent research results in the area of web application security. We described unique characteristics of web application

*

development identified important security properties that secure web applications should preserve and categorized existing works into three major classes. We also pointed out several open issues that still need to be addressed. Web applications have been evolving extraordinarily fast with new programming models and technologies emerging, resulting in an ever-changing landscape for web application security with new challenges, which requires substantial and sustained efforts from security researchers. We outline several evolving trends and point out several pioneering works as follows. First, an increasing amount of application code and logic is moving to the client side, which brings new security challenges. Since the client-side code is exposed, the attacker is able to gain more knowledge about the application, thus more likely to compromise the server-side application state. Several works have been trying to address this problem [19], [43], [97], [98], [92], [93]. Second, the business logic of web applications is becoming more and more complex, which further exacerbates the absence of formal verification and robust protection mechanisms for application logic. For example, when multiple web applications are integrated through APIs, their interactions may expose logic vulnerabilities [100]. Third, an increasing number of web applications are embedding third-party programs or extensions, e.g., iGoogle gadgets, Facebook games etc. To automatically verify the security of third-party applications and securely integrate them is nontrivial [85]. Last but not least, new types of attacks are always emerging, e.g., HTTP parameter pollution attack [101], which requires security professionals to quickly react without putting a huge number of web applications at risk.

REFERENCES

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

[2] Web Application Security Statistics, http://projects.webappsec.org/w/page/13 246989/WebApplication SecurityStatistics.

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

[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 9th USENIX conference on Operating systems design and implementation*, 2010, pp. 1–8.

*