*IOSR Journal of Computer Science (IOSR-JCE)*
*e-ISSN: 2278-0661, p-ISSN: 2278-8727*
*PP 47-53*
*www.iosrjournals.org*

# Web Page Access Model Study for Secure Execution and Prevention of Source Code Theft of Web Pages

Ishit Shah[1], Nisha Shah[2]
[1](PG *student: Comp Eng Dept SVIT Vasad, India)*
[2](*Asst Prof: Comp Eng Dept SVIT Vasad, India)*

***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 source code stealing and other 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. This paper surveys the area of web application security, with the aim of the development of a new web page access model middleware into a big picture that reduces the risk of source code theft by inserting a layer between user request and web page. 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.*

***Keywords** -Web access model, web page encryption, source code security, web access middleware, web security, intranet source code security, source code encryption*

## I. INTRODUCTION

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 [6] 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, JAVA), 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 error-prone 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 [7]. A recent report [8] reveals that over 80% of the websites on the Internet have at least one serious vulnerability.

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.
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 propose a web access model middleware which protects the website from source code stealing. This middleware will act as a webpage access manager, which receives the request from the user and then search the page (which is stored in the encrypted form on the server), decrypt the file, run the file and obtain the result. And then it will compile the result in the web page format and forward it to the client

## II. UNDERSTAND HOW A WEB APPLICATION WORKS

### 2.1 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 in- stance, 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.

#### *2.1.1* State Maintenance

HTTP protocol is stateless, where each web request is independent of each other. However, to implement non-trivial functionalities, “state full” 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.

#### 2.1.2 Logic Implementation

The business logic defines the functionality of a web application, which is specific to each application. Such functionality is manifested as an intended application control 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.

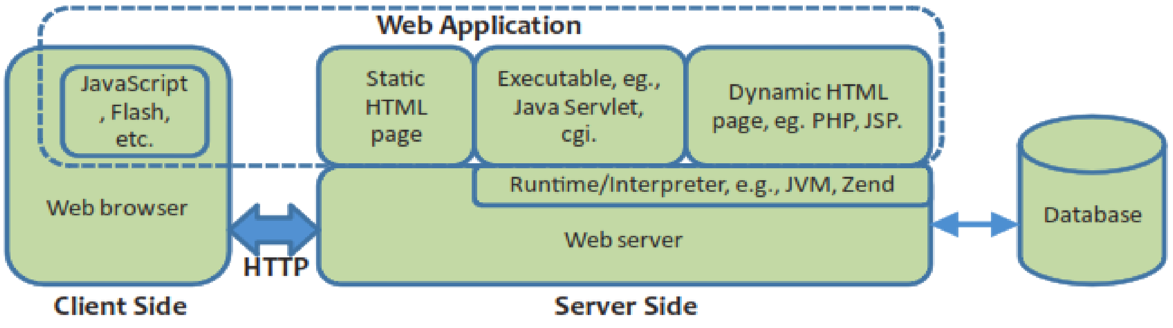

Fig. 1. Overview of Web Application

## 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 un-tampered; logic correctness

Given the threat model, user input data cannot be trusted. However, for the entrusted 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:

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, data- flow 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 by- pass, information disclosure and even the destruction of the entire database. Interested reader can refer to [9] for more details about SQL injection.

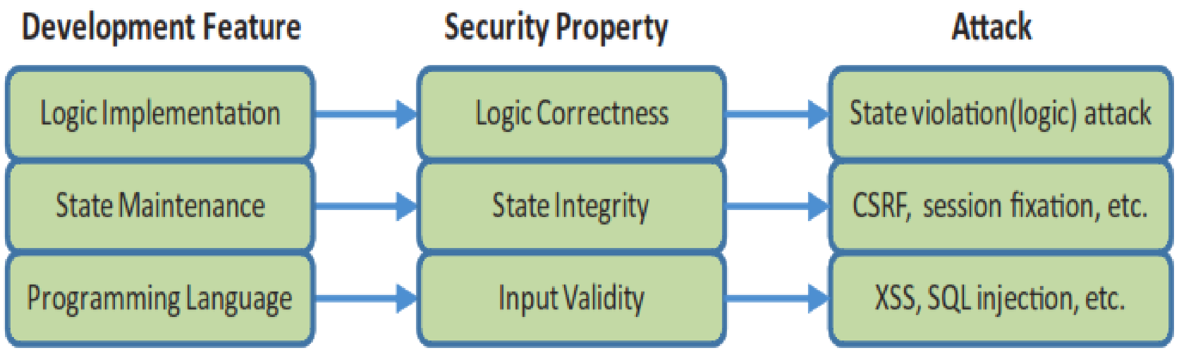

**Fig. 2. Web Application Security Properties**
2) Cross-Site Scripting: A cross-site scripting (XSS) attack is successfully launched when malicious contents within user input flow into web responses without correct validation. The web browser interprets all the web responses returned by the trusted 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 [10]). 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 [11], etc.

## IV. SOURCE OF INSPIRATION AND ARCHITECTURE POSSIBILITIES

Ref.[1] The authors of paper defines the architecture and implementation of a inclusive system which protect the Web contents from the various web attacks. Here this system provides Web content protection : Source code files stored on web server would be in encrypted form, Secure Execution : Encrypted source code files would be decrypted, executed and re-encrypted on request generated by another page, Authorized Distribution : A request generated would be checked for integrity of its origin through some secret code or its url before processing it. If a request is generated from an unauthorized page, then it will fail the processing part. This system added new security components and extended security features in existing technologies, security standards and protocols. They are as below

- Web pages protection by strong encryption techniques
- Web contents and resources encapsulate in PKCS#7
- Secure execution environment for Java Web Server

- eXtensible Access Control Markup Language (XACML) based authorization and secure Web proxy.

This system design and implement by the generic security objects and component-based architecture. This system can run on existing web infrastructure. This system is design for and confidentiality of data which are stored on server or transmission over network between client and server. Integrity and confidentiality of data is checked for the data origin. An integrated and confidential data is generally coming from a trusted source or source file and without any kind of change into it on the transmission line or communication path. i.e. Integrity and confidentiality are techniques to check the originality of data. This system doing authorization by XACML policies.
Ref.[2] HTML is widely use for WebPages creation in computer network of Internet and Intranet. While WebPages load in browser the source code of HTML and JavaScript is also obtain. By this the anyone can get the source code easily. So, Protecting the WebPages content is very big issue. The authors of the paper describe the one method of secure HTML code by Encryption based on polymorphic JavaScript programs. This encryption method helps to protect Java Scripts from polymorphic viruses. It also uses compression, permutation and check digits which increases security levels. By the use of this algorithm shareware "Athena WebLock" can design for WebPages encryption. For the designing of "Athena WebLock" by two algorithms "Athena" and "parhelion" are use. The "Athena" algorithm provides the method for HTML code encryption and "parhelion" algorithm provide the method for Java Scripts decrypting program. These algorithms help to keep the webpage security. The encryption by shareware "Athena WebLock" can protect the WebPages in most of all cases.
Ref.[3] One general reason for document vulnerability found by authors is the data integration from various sources on different web applications. So, they propose WebSSARI tool base algorithm which base on non interferences policy by focusing on two basic axioms Denning's axioms and Strom's type state. Which have two advantages First is It captures information flow semantics very effectively than static system. So, false positive rate is lower and second is It requires no effects from programmer point of view.
The system is test by the WebSSARI tool. This tool extends script language with proposed system. A code walker contain a lexer, a parser, an AST(abstract syntax tree ) maker and program abstractor. For supporting different language the different code walker have to implement. This system developed a separate GUI featuring batch verification, result analysis, error logging, and report generation. The limitation of this system is that it finds symptoms of errors rather than their causes.
Ref.[4] Software must have to secure by limited access to other parties while the interaction occurs between them. The web applications may affect by cross-site scripting and SQL injection and they are made access easily through internet by any unauthorized entrusted parties. By the use of static code analysis computer program code can evaluate about security without executing them. Static code analysis is the automatic technique which scans the code for security problem. Many PHP web applications contain the static code analysis to evaluate security before they are added in the system on the web.PHP is highly compatible with static code analysis because PHP is highly dynamic in nature. In this paper evaluation of the static analysis tools are done by the synthetic and real-world tests. Here the capabilities and performance of these tools are examined in accuracy and speed. The suited tool is use in system. Most of tools are not compatible to run in serious production environment they contain only small numbers of self contain ideas. All of them Pixy is the one only open tool

which use in real production world. But now days it is out dated. The commercial solutions (Fortify and CodeSecure) are widely use in serious production environment in real world. There are some problems with static code analysis. One of the problems with static code analysis is that it access the some resources for running security tests and it causes the time delay in application deployment and second problem is undecidebility of fundamental problem. Programmers use Static analysis to understand the code and find the errors or problems. It is not feasible to decide that application is secure or not by static code analysis for automatic rejection system.
Ref.[5] The full source code of any websites keep safe by the owner of that websites for the security. But the hackers try to get full source code of websites and the configuration files over HTTP. This paper gives technique to secure prevalent coding flaw in web applications which hackers use to extract source code and configuration files over HTTP. Disclosure of source code and configuration files is very important for security because They usually contain database connection information like IP address, Port number, valid credentials, application test users login names and passwords and others important data. The attacks of this type are very dangers. It will completely unnotice because it only exploits a functionality of the page! It will leave no unusual trail like an error log. Secure the database connections and its information is more important because in most of cases the databases have direct accessibility over the internet. Any user can connect to database directly using client and gain complete control over the database. For the disclosure the source code over http has following steps

- Validate the folder from where the file to be downloaded is being requested
- Validate the file types that are requested by users.
- Index files to be downloaded and pass only their index numbers as the URL parameter values

## 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 [12] [13] [14] [15] [16] [17]. 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 ex- ample, when multiple web applications are integrated through APIs, their interactions may expose logic vulnerabilities [18]. Third, an increasing number of web applications are embed- ding 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 non- trivial [20]. Last but not least, new types of attacks are always emerging, e.g., HTTP parameter pollution attack [19], which requires security professionals to quickly react without putting a huge number of web applications at risk.

## REFERENCES

[1] Abbasi, Abdul Ghafoor, Sead Muftic, and Ikrom Hotamov. "Web Contents Protection, Secure Execution and Authorized Distribution." In Computing in the Global Information Technology (ICCGI), 2010 Fifth International Multi-Conference on, pp. 157-162. IEEE, 2010.

[2] Zhongying, Bai, and Qin Jiancheng. "Webpage Encryption Based on Polymorphic Javascript Algorithm." In Information Assurance and Security, 2009. IAS'09. Fifth International Conference on, vol. 1, pp. 327-330. IEEE, 2009.

[3] Huang, Yao-Wen, Fang Yu, Christian Hang, Chung-Hung Tsai, Der-Tsai Lee, and Sy-Yen Kuo. "Securing web application code by static analysis and runtime protection." In Proceedings of the 13th international conference on World Wide Web, pp. 40-52. ACM, 2004.

[4] De Peol, N. L. "Automated security review of php web applications with static code analysis." Master's thesis (2010).

[5] Anant Kochar. Whit paper Source Code Disclosure over HTTP, by SecureEyes Infusing Security ” www.secureyes.net/downloads/Source_Code_Disclosure_over_HTTP.pdf ”

[6] Verizon 2010 Data Breach Investigations Report, “http://www.verizonbusiness.com/resources/reports/rp 2010-data- breach-report en xg.pdf.”

[7] Web Application Security Statistics, “http://projects.webappsec.org/w/page/13246989/WebApplication SecurityStatistics.”

[8] White Hat Security, “WhiteHat website security statistic report 2010.”

[9] W Halfond, W. G., Jeremy Viegas, and Alessandro Orso. "A classification of SQL-injection attacks and countermeasures." In Proceedings of the IEEE International Symposium on Secure Software Engineering, Arlington, VA, USA, pp. 13-15. 2006

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

[11] A Barth, Adam, Juan Caballero, and Dawn Song. "Secure content sniffing for web browsers, or how to stop papers from reviewing themselves." In Security and Privacy, 2009 30th IEEE Symposium on, pp. 360-371. IEEE, 2009.

[12] Chong, Stephen, Jed Liu, Andrew C. Myers, Xin Qi, Krishnaprasad Vikram, Lantian Zheng, and Xin Zheng. "Secure web applications via automatic partitioning." In ACM SIGOPS Operating Systems Review, vol. 41, no. 6, pp. 31-44. ACM, 2007.

[13] Saxena, Prateek, Steve Hanna, Pongsin Poosankam, and Dawn Song. "FLAX: Systematic Discovery of Client-side Validation Vulnerabilities in Rich Web Applications." In NDSS. 2010.

[14] Bisht, Prithvi, Timothy Hinrichs, Nazari Skrupsky, Radoslaw Bobrowicz, and V. N. Venkatakrishnan. "NoTamper: automatic blackbox detection of parameter tampering opportunities in web applications." In Proceedings of the 17th ACM conference on Computer and communications security, pp. 607-618. ACM, 2010.

[15] Bisht, Prithvi, Timothy Hinrichs, Nazari Skrupsky, and V. N. Venkatakrishnan. "WAPTEC: whitebox analysis of web applications for parameter tampering exploit construction." In Proceedings of the 18th ACM conference on Computer and communications security, pp. 575-586. ACM, 2011.

[16] A Guha, Arjun, Shriram Krishnamurthi, and Trevor Jim. "Using static analysis for Ajax intrusion detection." In Proceedings of the 18th international conference on World wide web, pp. 561-570. ACM, 2009.

[17] Vikram, K., Abhishek Prateek, and Benjamin Livshits. "Ripley: automatically securing web 2.0 applications through replicated execution." In Proceedings of the 16th ACM conference on Computer and communications security, pp. 173-186. ACM, 2009.

[18] Wang, Rui, Shuo Chen, XiaoFeng Wang, and Shaz Qadeer. "How to shop for free online--Security analysis of cashier-as-a-service based Web stores." In Security and Privacy (SP), 2011 IEEE Symposium on, pp. 465-480. IEEE, 2011

[19] Balduzzi, Marco, Carmen Torrano Gimenez, Davide Balzarotti, and Engin Kirda. "Automated Discovery of Parameter Pollution Vulnerabilities in Web Applications." In NDSS. 2011.

[20] Krishnamurthy, Akshay, Adrian Mettler, and David Wagner. "Fine-grained privilege separation for web applications." In Proceedings of the 19th international conference on World wide web, pp. 551-560. ACM, 2010.

[21] Wang, Helen J., Chris Grier, Alexander Moshchuk, Samuel T. King, Piali Choudhury, and Herman Venter. "The Multi-Principal OS Construction of the Gazelle Web Browser." In USENIX Security Symposium, pp. 417-432. 2009.