International Journal of Scientific Research in Computer Science, Engineering and Information Technology
ISSN : 2456-3307 (www.ijsrcseit.com)
doi : https://doi.org/10.32628/CSEIT206543

# A Survey on Web Application Security

**Danish Mairaj Inamdar, Prof .Shyam Gupta**
Savtribai Phule Pune University, Pune, Maharashtra, India

**Article Info**
Volume 6, Issue 5
Page Number: 223-228
Publication Issue :
September-October-2020

**Article History**
Accepted : 05 Sep 2020
Published : 15 Oct 2020

## ABSTRACT

Web application security has become real concern due to increase in attacks and data breaches. As Application becomes critical, complex and connected, the difficulty of achieving application security increases exponentially. Also there are tools and techniques to detect such attacks, threat and vulnerabilities that exist in application which developer prevent and mitigate the risk associated to it. This paper evaluates various web application attack detection mechanisms and how resistant they are against various attacking techniques. Such an evaluation is important for not only measuring the available attack defense against web application attacks but also identifying gaps to build effective solutions for different defense techniques on web application and use it for study. Based on the research, the limitations of these application attack detection techniques are identified and remedies proposed for improving the current state attack detection on web applications.

**Keywords** : Input Validation; Open Web Application Security (OWASP); Vulnerability Assessment

## I. INTRODUCTION

Insecure software is undermining our financial, healthcare, defense, energy, and other critical infrastructure. The rapid pace of modern software development processes makes risks even more critical to discover quickly and accurately. The flaws in the application are further exploited leading to attack on the application. Evaluating the web application security risks based on the recommendations from leading practices that are adopted as an application security standard that covers off around 80-90% of all common attacks and threats. In order to prevent attacks Open Web Application Security Top Ten list is considered as Standard for Vulnerability Assessment. It includes different vulnerabilities such as Injection, Authorization bypass, Authentication, Cross site Scripting and XML External Entities. The paper is further organized as follows: the first section introduces to different vulnerabilities in web applications. Second section comprises ways to mitigate the various vulnerabilities. Third section showcases the comparison of available attack detection mechanism based on common security flaws.

## II. LITERATURE SURVEY

*A. Vulnerabilities of Modern Web Application*

Existing work in web application security focuses especially on general security flaws: injection, cross-site scripting, sensitive data leakage and user authorization and user authentication[1].It involves comparison of pen-testing tools and ways to mitigate found flaws on use-case application.

*B. Web Application Security Approach*

The existing research works on securing the web application showcases different approaches used such as Web Application firewall, vulnerability assessment and penetration testing[2]. The current scenario of web application security has shortcomings as preventive mechanisms are not implemented at run-time. Also, Attackers are becoming smarter by finding new and clever ways to create malicious inputs that will bypass the Firewall input filters. Passive Approaches such as Vulnerability Assessment and Penetration Testing is effective in threat and attack detection but it’s time consuming process.

## III. PROPOSED METHODOLOGY

As long as code and data cannot be distinguished by machines, Injection attacks will prevail. The Proposed Methodology helps to mitigate it.Run-time Application Self Protection (RASP) is a technology that executes on a server and kicks in when an application is in running state. It's designed to detect attacks on an application in real time. When an application begins to run, it can protect it from un-trusted input or behavior by analyzing both the application’s behavior and the context of that behavior. By using this technology in the application to continuously monitor its own behavior, attacks can be identified and mitigated immediately without human intervention It incorporates security while running application and wherever it resides on a server. It intercepts all calls from the application to a system, making sure they are secure, and validates data requests directly inside the application.

Both web and non-web applications can be protected by using it. The technology doesn't affect application design because it's detection and protection features operate on the server the application’s running on. This methodology focuses on helps the application to differentiate between the code and data present in the web application to detect attacks and mitigate the vulnerabilities.

A. Block -Diagram

The Run-time Application Self protection technology injects security at runtime and prevents the application core layer from direct interaction with user level request and response through security layer protection as shown

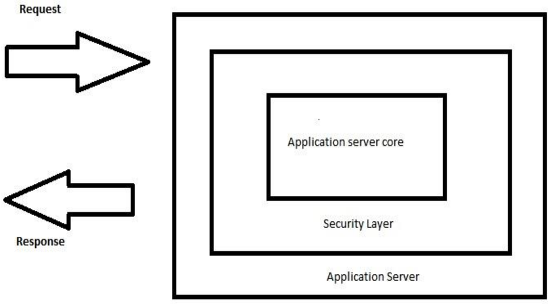


Fig.1. Block –Diagram RASP security layer

B. Working Principle

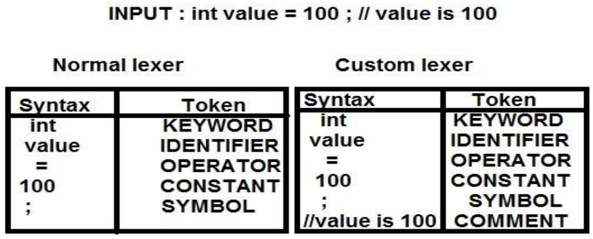
Using Run-time Application Self Protection by Application Programming Interface Instrumentation and Dynamic White-list is achieved through three methods like lexical analysis, context determination and monkey patching.

Lexical Analysis and Token Generation

RASP uses lexical analysis approach to scan the input program and convert it into a sequence of Tokens. Generally, Tokenization involves sequence of characters that can be treated as a unit in the grammar of the programming language and it divides the program into valid tokens.

Example of tokens:
Type token (id, number, real, ..)
Punctuation tokens (IF, void, return, )
Alphabetic tokens (keywords)

Example of Non-Tokens: Comments, pre-processor directive, macros, blanks, tabs, newline etc.



INPUT : int value = 100 ; // value is 100

Normal lexer

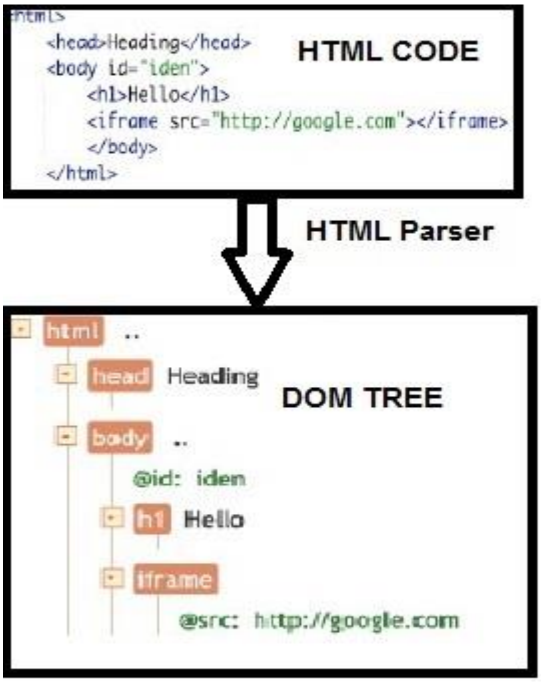
| Syntax | Token |
|---|---|
| int | KEYWORD |
| value | IDENTIFIER |
| = | OPERATOR |
| 100 | CONSTANT |
| ; | SYMBOL |

Custom lexer

| Syntax | Token |
|---|---|
| int | KEYWORD |
| value | IDENTIFIER |
| = | OPERATOR |
| 100 | CONSTANT |
| ; | SYMBOL |
| //value is 100 | COMMENT |

Fig. 2. Lexical Analysis for sample code

Context Determination helps to determine the context of code by parsing the test code into Document object Model Tree view to understand the syntax as shown.



```
<html>
    <head>Heading</head>
    <body id="iden">
        <h1>Hello</h1>
        <iframe src="http://google.com"></iframe>
        </body>
    </html>
```

Fig 3. DOM Tree

A monkey patching also know as Run-time Hooking is a way for a program to extend or modify supporting system software locally. It helps to patch functions and methods.

Based on the above methods, different flaws can be protected like SQL query injection, Operating System Command injection and Cross-site Scripting. It involves hooking Application Programming Interface by modifying behavior and flow of calls for un-trusted input based on context matching. Next step involves learning about the normal behavior of the request and create a white-list based on rules formed. Finally, Run-time Application Self Protection blocks malicious request into system by detecting attacks at run-time.

C. Algorithm Steps (Work-flow)

RASP technique can be applied to various vulnerabilities.
For Path traversal

Step 1 : Hook File Input/Output Application Programming Interface
io.open("/directory/filename","permissions")
Step 2 : Learn about directories and file extensions
Step 3: Block any unknown file directories and extensions

## IV. RESULT AND DISCUSSIONS

Before examining the results of our research, we provide a definition of Web Application Firewalls and their shortcomings and justify the usage of Run-time Application Self Protection. Web Application Firewall intercepts's requests to a potentially vulnerable web application applying rules to evaluate whether a request contains input that might exploit the application. This process requires complex configuration and it may fail open under high load, leaving web applications vulnerable. For a Firewall to function at its peak, there need to know what the vulnerable inputs to the web application are so you can apply the appropriate protections to these input fields.

In contrast, Run-time application self protection integrates with the underlying code libraries and protect the vulnerable areas of the application at the source code level. When a user makes a function call containing parameters that might cause harm to the web application, it intercepts the call at run-time, logging or blocking the call, depending on the configuration. This method of protecting a web application differs fundamentally from a firewall.

The key features that differentiate run-time application self protection is to detect attacks and vulnerabilities, no hardware requirements, zero code modification and easy integration. It also eliminates false positives as it can differentiate between application and user data.

*Table 1*

| Parameters | Run time Application Self Protection | Web Application Firewall |
|---|---|---|
| Accuracy | Monitors Inbound and Outbound data and logic flaws | Detection based on pattern matching without considering input passed in application. |
| Reliability | Will not fail under high load,regardless of server load | Single point of failure under high load on server |
| Platforms | Any Instrumented Application | Only web Application |
| Visibility | May provide detailed feedback to developers to show how to re mediate code vulnerabilities. | Offers no detailed insight into application. |
| Maintenance | Automatically understands changes to application. | Can gain application context through training only |

This technology originated as a solution not only to simplify the test for application security risks, but to mitigate real-time threats to production applications. It has also evolved to provide powerful capabilities for database monitoring and application attack visibility leading to faster remediation. It ensures that application is protected with no impact on operations and performance. Early implementations of the technology could cause as much as 10 percent increase in  response times within the application tier, but performance is constantly improving.

## V. CONCLUSION

Run-time application self protection stands above any traditional Web Application Firewall, by protecting web applications out of the box with minimal (if any) configuration needed. This feature could substantially reduce risk by enabling application protection immediately upon deployment. It's capability to instrument at the Application Programming Interface layer allows it to detect attacks precisely. It reports fewer false positives because of it's ability to perform context-sensitive matching. Also there is need to deal with challenges to build ideal Run-time application self Protection Solution and  adapt other security techniques in combination with it.

## VI. REFERENCES

[1]. F. Holik, S. Neradova, "Vulnerabilities of Modern Web Applications, MIPRO 2017", May 22- 26, 2017, Opatija, Croatia

[2]. Ashikali M Hasan, "Perusal of Web Application Security Approach", 2017 International Conference on Intelligent Communication and Computational Techniques (ICCT) Manipal University Jaipur, Dec 22-23, 2017

[3]. M. Alenezi, Javed, Y., "Open source web application security: A static analysis approach," in: 2016 International Conference on Engineering MIS (ICEMIS). pp. 1–5 (Sept 2016)

[4]. S. Rafique, Humayun, M., Hamid, B., Abbas, A., Akhtar, Iqbal, K., "Web application security vulnerabilities detection approaches: A systematic mapping study," in: 2015 IEEE/ACIS

[5]. 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.

[6]. J. Sohn, Ryoo, J., "Securing web applications with better patches: An architectural approach for systematic input validation with security patterns," in: 2015 10th International Conference on Availability, Reliability and Security. pp. 486–492 (Aug 2015)

[7]. Marcelo Invert Palma Salas, "Security Testing Methodology for Evaluation of Web Services Robustness - Case: XML Injection", Paulo Lício de Geus, Eliane Martins Institute of Computing, UNICAMP, Campinas, Brazil, 2015

[8]. 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

[9]. Zhou L, J. Ping, H. Xiao, Z. Wang, GeguangPu, and Z. Ding, "Automatically Testing Web Services Choreography with Assertions, In Proceedings of the 12th international Conference on Formal Engineering Methods and Software Engineering. ICFEM'10". Springer-Verlag, Berlin, Heidelberg, 2010.

[10]. H. Hakim, Sellami, A., Abdallah, H.B., "Evaluating security in web application design using functional and structural size measurements," in: 2016 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA). pp. 182–190 (Oct 2016)

[11]. Daniel Nations, "Improve Your Understanding of Web Applications,lifewire.com", 17 October 2016 ..

[12]. OWASP Secure Coding Practices, "OWASP Secure Coding Practices - Quick Reference Guide", 11 May 2017.

[13]. WHITEHAT SECURITY, INC., "Web Applications Security Statistics Report 2016," WHITEHAT SECURITY, INC., 2016.

[14]. Danny Allan, strategic research analyst,IBM Software Group, "Web application security:automated scanning versus manual penetration testing", IBM Software Group , January 2008.

**Cite this article as :**

Danish Mairaj Inamdar, Prof .Shyam Gupta, "A Survey on Web Application Security", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 6 Issue 5, pp. 223-228, September-October 2020. Available at doi : https://doi.org/10.32628/CSEIT206543
Journal URL : http://ijsrcseit.com/CSEIT206543