Understanding Local File Inclusion (LFI)

Local File Inclusion (LFI) is a critical web vulnerability that allows attackers to include local files from the server into a web application. This often occurs due to improper input validation in file inclusion mechanisms, leading to severe consequences such as sensitive data exposure, arbitrary code execution, or denial of service. It's a common vulnerability in older web applications built with PHP, JSP, ASP, and other server-side scripting languages.

How Local File Inclusion Works

1

User Input

Applications include files based on user-supplied input, for example, through a URL parameter.

2

Lack of Validation

Without proper sanitization, the application treats malicious input as a legitimate file path.

3

Directory Traversal

Attackers use "directory traversal" sequences (e.g., ../../) to navigate outside the intended directory.

4

File Exposure

The server then includes and executes or displays the content of the specified arbitrary file.

Example: `http://site.com/?file=../../../../etc/passwd` could expose the system's password file. A vulnerable code snippet might look like: `include($_GET['file'] . ".php")` without proper input validation.

LFI Attack Techniques and Impact

Attack Techniques

- **Null Byte Injection (%00):** Appending `%00` to bypass file extension restrictions, although largely patched in modern PHP versions.
- **Path Truncation:** Overloading the filename length to ignore appended extensions.
- **Encoding Tricks:** Using URL encoding or double encoding to bypass WAFs and filters.

Potential Impact

- **Information Disclosure:** Access to sensitive configuration files, source code, or user data.
- **Remote Code Execution (RCE):** If combined with other vulnerabilities, attackers can execute arbitrary code on the server.
- **Cross-Site Scripting (XSS):** Including files with user-controlled content can lead to XSS attacks.
- **Denial of Service (DoS):** Including large or non-existent files can crash the application.

Notable real-world breaches include the Jenkins LFI vulnerability (CVE-2024-23897) and the massive Adult Friend Finder data leak.

Testing and Detecting LFI Vulnerabilities

Identify Parameters

Look for URL parameters that accept file names or paths, like `file=`, `page=`, `template=`.

Test Traversal Payloads

Attempt common directory traversal payloads such as `../../../../etc/passwd` (Linux) or `../../../../windows/win.ini` (Windows).

Bypass Filters

Experiment with null bytes (`%00`), URL encoding, or double encoding to bypass application filters or Web Application Firewalls (WAFs).

Automated Scanners

Utilize automated vulnerability scanners like Acunetix, Burp Suite, or OWASP ZAP, which include modules specifically designed to detect LFI.

Manual testing combined with automated tools provides a comprehensive approach to LFI detection.

Preventing and Mitigating LFI Attacks

- **Strict Input Validation:** Never trust user input. Validate and sanitize all file path inputs rigorously, ensuring they conform to expected formats and characters.
- **Whitelisting:** Implement a whitelist for allowed file names and directories. Only permit inclusion of files explicitly defined in a safe list, rejecting all others.
- **Disable Dangerous Directives:** For PHP, set `allow_url_include = Off` in `php.ini` to prevent remote file inclusion. Also, consider setting `open_basedir` to restrict file access.
- **Least Privilege:** Configure web server processes to run with the minimum necessary permissions, limiting potential damage if an LFI attack occurs.
- **Regular Monitoring:** Continuously monitor server logs for suspicious file access patterns, repeated directory traversal attempts, or unusual HTTP requests.