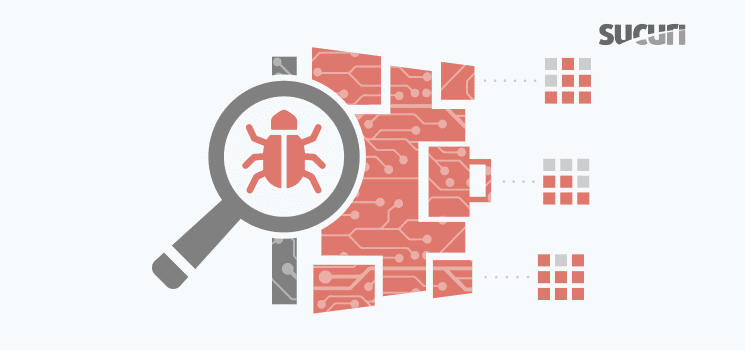
**What are Website Backdoors?**

****

[Backdoors](https://blog.sucuri.net/tag/website-backdoor) are types of malware that allow for remote control of a compromised website by bypassing appropriate authentication methods.

Even after updating a site, changing its passwords, and doing other post-hack procedures, the backdoor might not be removed. Leaving a backdoor in a website allows it to be accessed unexpectedly.

Even though there are backdoors written in all languages, the ones we see the most are done in **PHP**.

**Why Are Backdoors So Hard to Find?**

Backdoors are usually very hard to find because **they don’t have to be linked anywhere in the site**. They don’t need to be big, quite the opposite; they can be very small and go unnoticed.

Actually, most backdoors are designed so that **they can be easily confused with non-malicious code**. Some of them have **passwords**, some are heavily **encrypted**/encoded and they can be located anywhere in a website’s **file system** or **database**.

There are two main issues when trying to find a hidden backdoor.

First, in order to easily identify backdoors on your site you would have to know all the files and be familiar with all their content in order to be able to spot the differences.

Second, these functions can also be used legitimately by **plugins**. Be sure to test any changes because you could break your site by removing benign functions.

Look at this code, would you say it is a backdoor?

****

Yes, it is a **website backdoor**. This code allows attackers to **execute any type of code, add files, remove files, and perform other nefarious acts**. When you are analysing thousands of lines of code, it can be easy to miss it.

What about this code:

****

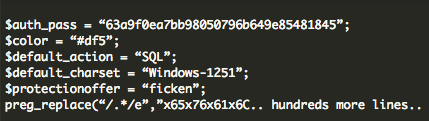
You are right if you said yes. In that code, the majority of the backdoor is hidden inside an image (void.jpg).

**Types of Backdoors**

Backdoors can appear in different types, here are some examples of backdoors:

**Big and complex backdoors:**

These are the easiest to spot because they are big. Here is an example of the “Filesman” backdoor, big, complex and easy to find:



**Short and simple backdoors:**

Backdoors can also be very short and simple and perform different actions, as seen below.

The following example of a backdoor is used to **execute any code from the “php” request**:



**CMS specific backdoors:**

Some backdoors are specific to a content management system (CMS).

The following example shows a **WordPress-based backdoor**. This time, the malicious content is hidden inside the **database** in the wp-options tables.



**Conclusion**

These are just some of the many examples of website backdoors that our [malware removal and research teams see every day](https://sucuri.net/website-security/how-we-do-it). If you believe your website might have a backdoor, but you are still unsure of how to find it. We have a dedicated team of [security analysts that will be happy to find them for you](https://sucuri.net/website-security-platform/malware-removal).

Sucuri has been involved specifically in the website security space, analyzing what attackers do and how they do it. This knowledge is at the core of how the technology is built.

Shell Uploading

## **What are file upload vulnerabilities?**

File upload vulnerabilities are when a web server allows users to upload files to its filesystem without sufficiently validating things like their name, type, contents, or size. Failing to properly enforce restrictions on these could mean that even a basic image upload function can be used to upload arbitrary and potentially dangerous files instead. This could even include server-side script files that enable remote code execution.

In some cases, the act of uploading the file is in itself enough to cause damage. Other attacks may involve a follow-up HTTP request for the file, typically to trigger its execution by the server.

## **What is the impact of file upload vulnerabilities?**

The impact of file upload vulnerabilities generally depends on two key factors:

* Which aspect of the file the website fails to validate properly, whether that be its size, type, contents, and so on.
* What restrictions are imposed on the file once it has been successfully uploaded.

In the worst case scenario, the file's type isn't validated properly, and the server configuration allows certain types of file (such as .php and .jsp) to be executed as code. In this case, an attacker could potentially upload a server-side code file that functions as a web shell, effectively granting them full control over the server.

If the filename isn't validated properly, this could allow an attacker to overwrite critical files simply by uploading a file with the same name. If the server is also vulnerable to [directory traversal](https://portswigger.net/web-security/file-path-traversal), this could mean attackers are even able to upload files to unanticipated locations.

Failing to make sure that the size of the file falls within expected thresholds could also enable a form of denial-of-service (DoS) attack, whereby the attacker fills the available disk space.

Practical

Here I am using DVWA a virtual machine for the practical of file uploading

Downloads metasploitable machine

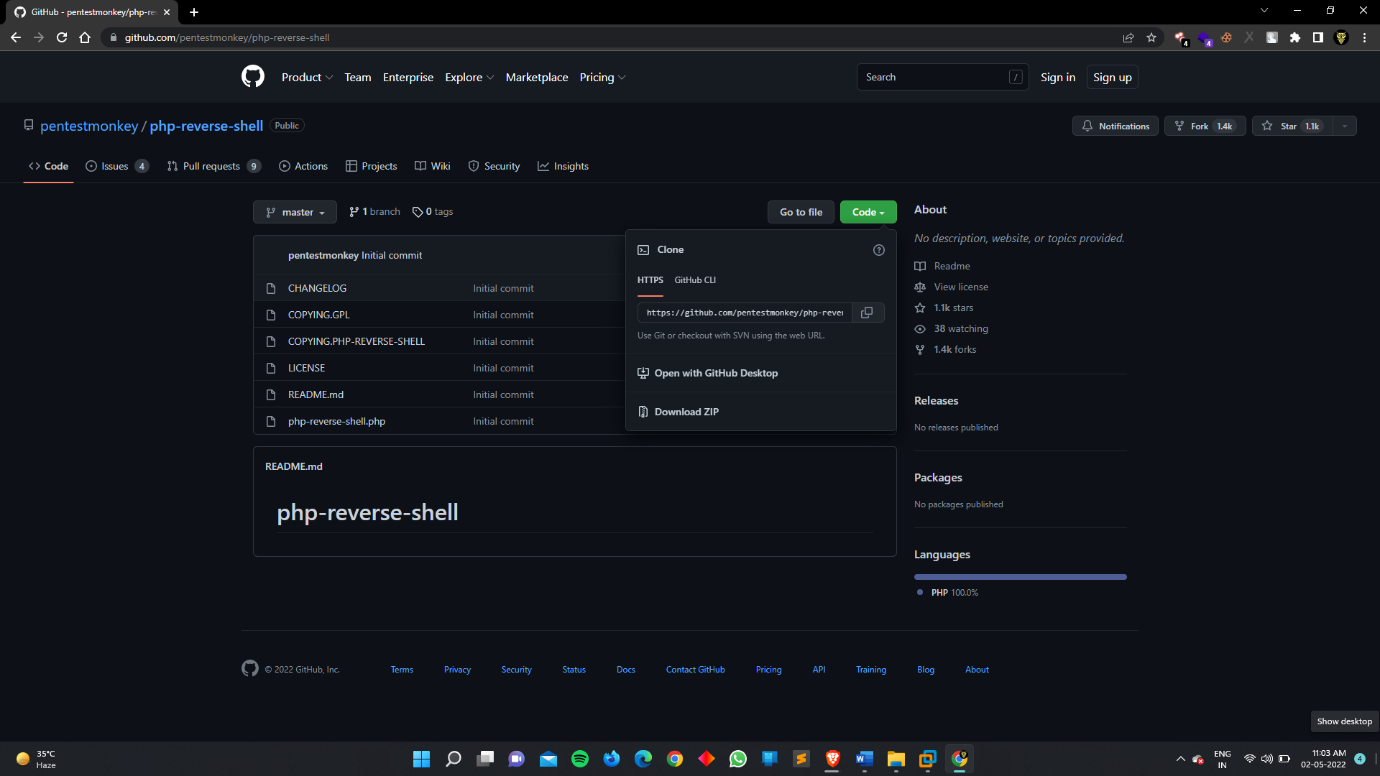
If config

Copy the ip address of the machine into the web

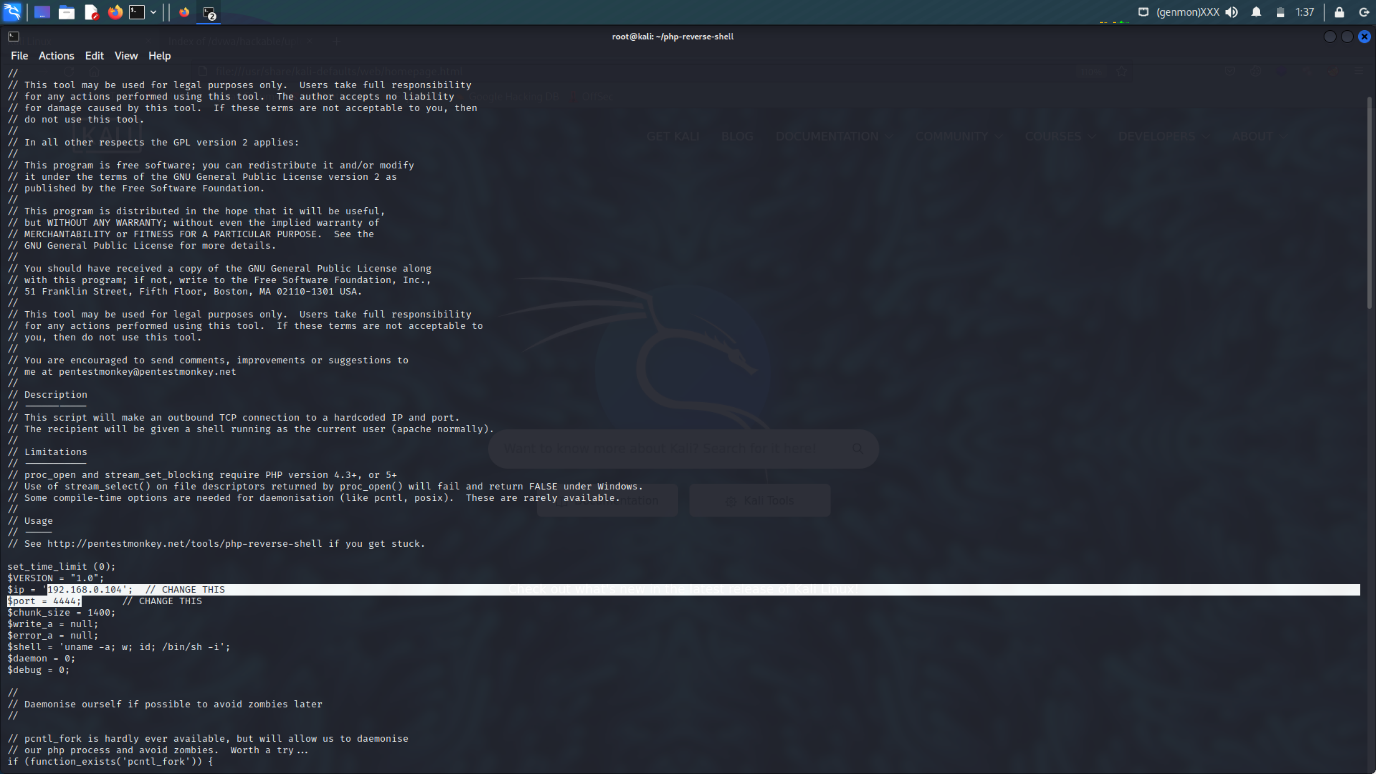
Downloads the Reverse shell.php from the github

Link :- <https://github.com/pentestmonkey/php-reverse-shell>

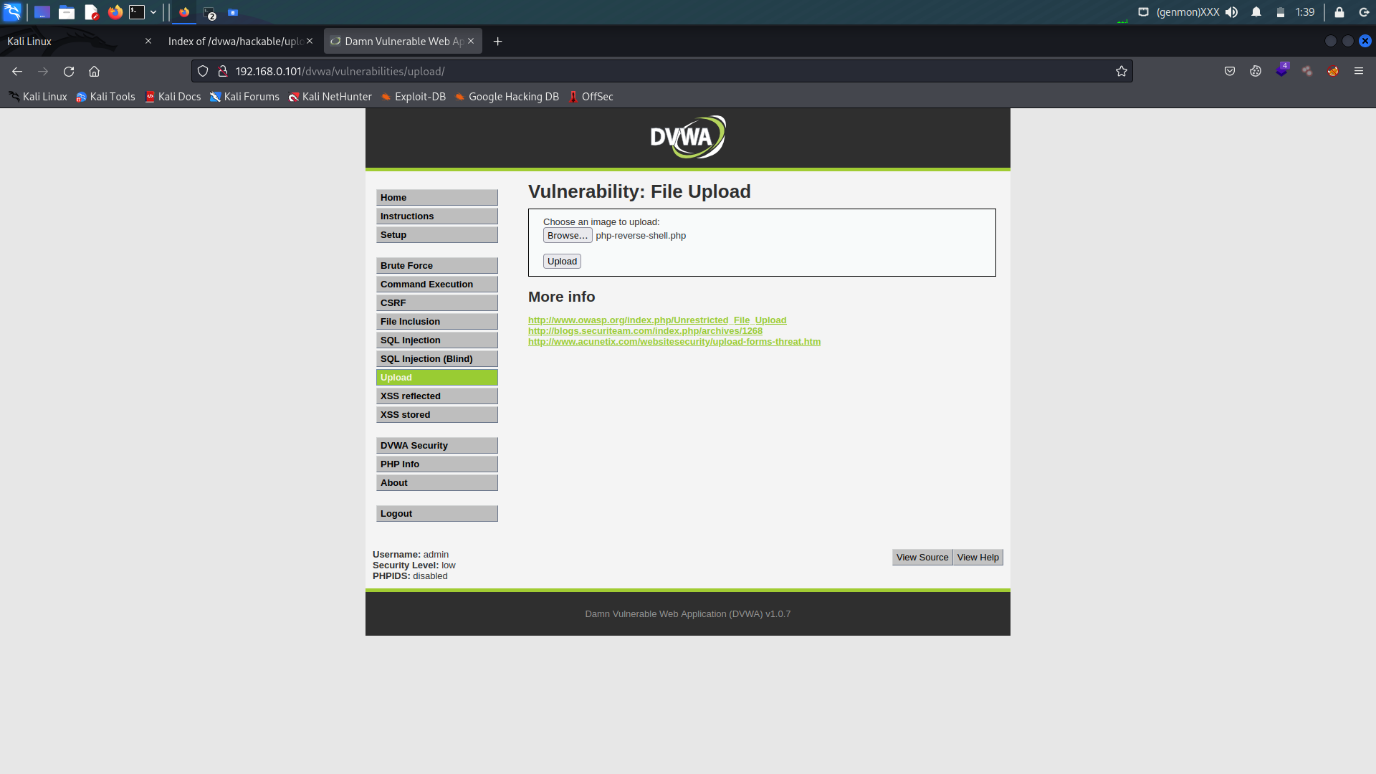
Photo



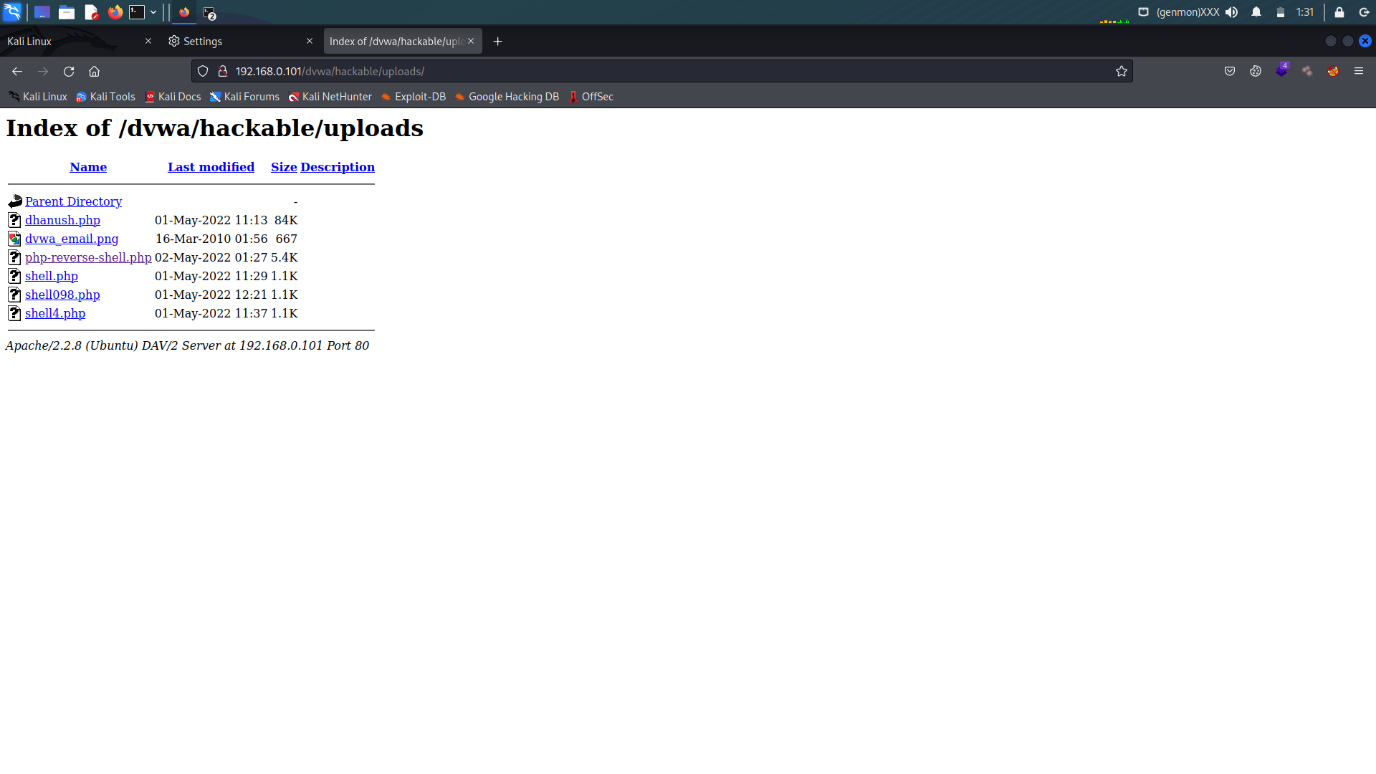
Change the ip address and port no.



Open the DVWA machine



Upload the php shell



Now finally the shell is uploading in to the website

**Local File Inclusion (LFI)**

LFI is a web vulnerability that results from mistakes at the website or web application programmers’ end. A hacker can take advantage of this vulnerability to include malicious files which are then executed by the vulnerable website or web application.

In an LFI vulnerability, the included file is already present on the local application server, targeted by the hacker. If successful, the attacker can read important files, access more sensitive information, or run arbitrary commands.

### **How Does Local File Inclusion Work?**

In Local File Inclusion, perpetrators exploit vulnerable PHP programs to access confidential data or run malicious scripts on the target server. This can expose critical data or allow threat actors to launch remote code execution or [Cross-site Scripting (XSS)](https://spanning.com/blog/cross-site-scripting-web-based-application-security-part-3/) attacks. LFI occurs when an application includes a file as user input without properly validating it. This allows an attacker to include malicious files by manipulating the input.

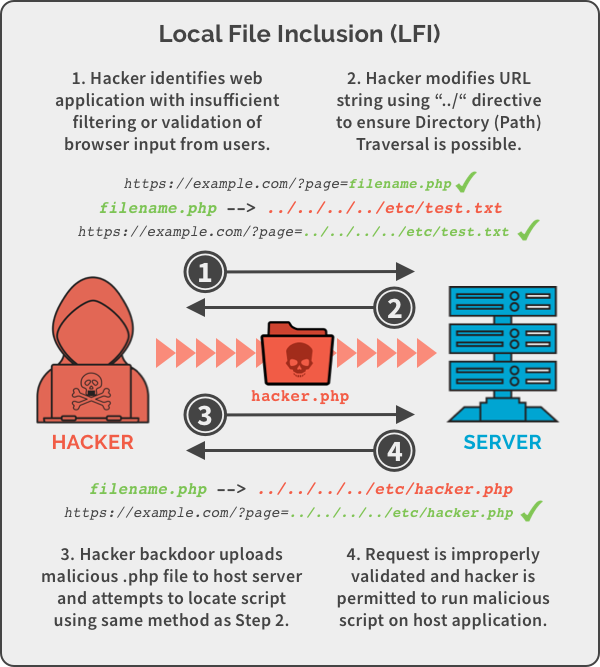
Here’s an example of a vulnerable PHP code that could lead to LFI:

**https://example.com/?page=filename.php**

Without proper input sanitizing, an attacker could easily modify the input (as shown below) to manipulate the application into accessing unauthorized files and directories from the host server using the “../” directive. This is known as [Directory (Path) Traversal](https://spanning.com/blog/directory-traversal-web-based-application-security-part-8/):

**https://example.com/?page=../../../../etc/test.txt**

In this example, a hacker was able to successfully exploit the vulnerability by simply replacing the “**filename.php**” with “**../../../../etc/test.txt**” in the path URL to access the test file. If this can be accomplished, a hacker can then backdoor upload a malicious script to the host server and use LFI to access the script. A simplified version of the process would look something like this:

****

**Practical**

Now start your machine and login to DVWA, then go to DVWA security tab and change the difficulty level to low.

****

Go to file inclusion tab and change the URL from **incude.php** to **?page=../../../../../../etc/passwd.**

****

****

change the URL from**?page=../../../../../../etc/passwd** to **?page=../../../../../../proc/version**.

****