Cheat Sheet

? Go to Questions

Table of Contents

Intro to File Inclu

File Disclosure

Basic Bypasses

Remote Code Execution

6 LFI and File Uploads

Automation and Prevention

File Inclusion Prevention Skills Assessment

Skills Assessment - File Inclusion

Automated Scanning

My Workstation

PHP Wrappers Remote File Inclusion (RFI)



Page 9 / Automated Scanning

Automated Scanning

It is essential to understand how file inclusion attacks work and how we can manually craft advanced payloads and use custom techniques to reach remote code execution. This is because in many cases, for us to exploit the vulnerability, it may require a custom payload that matches its specific configurations. Furthermore, when dealing with security measures like a WAF or a firewall, we have to apply our understanding to see how a specific payload/character is being blocked and attempt to craft a custom payload to work around it.

We may not need to manually exploit the LFI vulnerability in many trivial cases. There are many automated methods that can help us quickly identify and exploit trivial LFI vulnerabilities. We can utilize fuzzing tools to test a huge list of common LFI payloads and see if any of them work, or we can utilize specialized LFI tools to test for such vulnerabilities. This is what we will discuss in this section.

Fuzzing Parameters

The HTML forms users can use on the web application front-end tend to be properly tested and well secured against different web attacks. However, in many cases, the page may have other exposed parameters that are not linked to any HTML forms, and hence normal users would never access or unintentionally cause harm through. This is why it may be important to fuzz for exposed parameters, as they tend not to be as

page for common GET parameters, as follows:

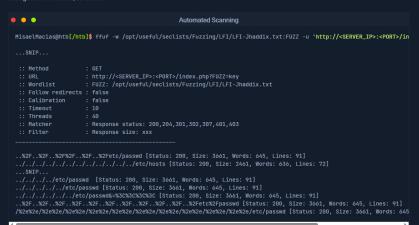


Once we identify an exposed parameter that isn't linked to any forms we tested, we can perform all of the LFI tests discussed in this module. This is not unique to LFI vulnerabilities but also applies to most web vulnerabilities discussed in other modules, as exposed parameters may be vulnerable to any other vulnerability as well.

LFI wordlists

reliable and can find LFI vulnerabilities that may not be identified otherwise, as discussed earlier. However, in many cases, we may want to run a quick test on a parameter to see if it is vulnerable to any common LFI payload, which may save us time in web applications where we need to test for various vulnerabilities.

There are a number of LFI Wordlists we can use for this scan. A good wordlist is LFI-Jhaddix.txt, as it contains various bypasses and common files, so it makes it easy to run several tests at once. We can use this wordlist to fuzz the ?language= parameter we have been testing throughout the module, as follows:



As we can see, the scan yielded a number of LFI payloads that can be used to exploit the vulnerability. Once we have the identified payloads,

Fuzzing Server Files

In addition to fuzzing LFI payloads, there are different server files that may be helpful in our LFI exploitation, so it would be helpful to know where such files exist and whether we can read them. Such files include: Server webroot path, server configurations file, and server loss.

Server Webroot

We may need to know the full server webroot path to complete our exploitation in some cases. For example, if we wanted to locate a file we uploaded, but we cannot reach its /uploads directory through relative paths (e.g. . . / . . /uploads). In such cases, we may need to figure out the server webroot path so that we can locate our uploaded files through absolute paths instead of relative paths.

To do so, we can fuzz for the index.php file through common webroot paths, which we can find in this wordlist for Linux or this wordlist for Windows. Depending on our LFI situation, we may need to add a few back directories (e.g. ../../../), and then add our index.php afterwords.

The following is an example of how we can do all of this with ffuf:

As we can see, the scan did indeed identify the correct webroot path at (/var/www/html/). We may also use the same LFI-Jhaddix.txt wordlist we used earlier, as it also contains various payloads that may reveal the webroot. If this does not help us in identifying the webroot, then our best choice would be to read the server configurations, as they tend to contain the webroot and other important information, as we'll see next.

Server Logs/Configurations

As we have seen in the previous section, we need to be able to identify the correct logs directory to be able to perform the log poisoning attacks we discussed. Furthermore, as we just discussed, we may also need to read the server configurations to be able to identify the server webroot path and other important information (like the logs path!).

To do so, we may also use the LFI-Jhaddix.txt wordlist, as it contains many of the server logs and configuration paths we may be interested in. If we wanted a more precise scan, we can use this wordlist for Linux or this wordlist for Windows, though they are not part of secLists, so we need to download them first. Let's try the Linux wordlist against our LFI vulnerability, and see what we get:

As we can see, the scan returned over 60 results, many of which were not identified with the LFI-Jhaddix.txt wordlist, which shows us that a precise scan is important in certain cases. Now, we can try reading any of these files to see whether we can get their content. We will read (/etc/apache2/apache2.conf), as it is a known path for the apache server configuration:

```
Automated Scanning

MisaelMacias@htb[/htb]$ curl http://<SERVER_IP>:<PORT>/index.php?language=../../../etc/apache2/apache2.conf
...SNIP...

ServerAdmin webmaster@localhost
DocumentRoot /var/www/html

ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined
...SNIP...
```

As we can see, we do get the default webroot path and the log path. However, in this case, the log path is using a global apache variable (APACHE_LOS_DIR), which are found in another file we saw above, which is (/etc/apache2/envvars), and we can read it to find the variable values:

```
Automated Scanning

MisaelMacias@htb[/htb]$ curl http://<SERVER_IP>:<PORT>/index.php?language=../../../etc/apache2/envvars
...SNIP...
```

As we can see, the (APACHE_LOG_DIR) variable is set to (/var/log/apache2), and the previous configuration told us that the log files are

Note: Of course, we can simply use a wordlist to find the logs, as multiple wordlists we used in this sections did show the log location.

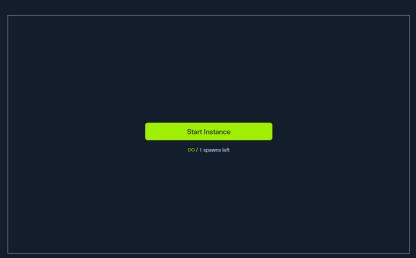
But this exercises shows us how we can manually go through identified files, and then use the information we find to further identify
more files and important information. This is quite similar to when we read different file sources in the PHP filters section, and such
efforts may reveal previously unknown information about the web application, which we can use to further exploit it.

LFI Tools

Finally, we can utilize a number of LFI tools to automate much of the process we have been learning, which may save time in some cases, but may also miss many vulnerabilities and files we may otherwise identify through manual testing. The most common LFI tools are LFISuite, LFIFreak, and liffy. We can also search GitHub for various other LFI tools and scripts, but in general, most tools perform the same tasks, with varying levels of success and accuracy.

Unfortunately, most of these tools are not maintained and rely on the outdated python2, so using them may not be a long term solution. Try downloading any of the above tools and test them on any of the exercises we've used in this module to see their level of accuracy.





Waiting to start...

