
LFI
Local File Inclusion Vulnerability

Local File Inclusion

Local File Inclusion is a vulnerability that can allow an attacker to access sensitive files that shouldn’t be accessible, or worst case scenario gain Remote Code Execution (RCE) on a vulnerable server. This vulnerability is usually introduced by the programmer’s end, usually by implementing dynamic file inclusion without proper sanitization, allowing the attacker to include sensitive files.

Sensitive Files

Sensitive files could include, but not limited to, database files, password files, ssh keys, source code, etc.

Remote Code Execution

There are several techniques that can lead to remote code execution. Imagine a php powered server. When we say we have LFI on that server, the files we can include go straight into the php code before it's evaluated. That means that if we somehow include php code, it could be evaluated before being sent to us. But how could we achieve this kind of behaviour?

Writable Logs

When we send a request on a server what happens underneath?

The request gets automatically logged on an access.log file. For apache servers that file is under /var/log/apache2/access.log

But we don’t control what the server writes on that file!

Well, we actually can control a single value. That value is the User-Agent. The server logs the IP, time, date, request, and User-Agent. We can write php code on the User-Agent (Payload: <?php echo system($_GET[‘cmd’]); ?>) to have some php code be injected in that file. In order to trigger that we need to include that file and pass a parameter ?cmd=id; to execute the payload.

Does this always work?

Well, if it’s that easy why don’t we do something for it? Well we did. If we take the apache example once again, the service is running as root, but drops privileges to the actual php code to www-data. The logs now are permission protected, meaning they are only accessible by the root user and the corresponding root group, so we can’t access that file for inclusion.

Is there any other way to gain RCE from LFI?

Other ways to gain RCE

Without going into much depth analyzing the ways of RCE, we will mention a few for reference.

We can abuse php headers (expect:), the session cookies, but the most common one is to include ssh keys and log in through ssh, if ssh is supported and open.

It is even possible to email a reverse shell! Exactly, if the vulnerable server also relays mail via any means, then it is possible to send a mail to it and it will store it. We can then include the mail file and include our payload.

Programmer's POV

Well, that's all good on how to exploit this vulnerability, but why it's there in the first place? We said that LFI is introduced on the programmer's end. Let's take for example the language PHP

What does this piece of code do? If the variable file is set then it will include that file. However what files can we include? It would be entirely possible to include a file from anywhere in the filesystem since there are no filters to stop us.

```
<?php
  $file = $_GET['file'];
  if(isset($file))
  {
    include("pages/$file");
  }
  else
  {
    include("index.php");
  }
?>
```

Mitigation Techniques

The first and foremost approach is to avoid including files from user selection. Try to use choices instead of specified raw input.

If you have to include like this and can’t avoid it for any reasons, sanitize your inputs! Never trust user input. Sanitize for path traversal attacks, for expect tags, for everything.

The absolute best way is to hold a whitelist of accepted filenames. That way you are not passing your user input to the filesystem or API, but instead you only allow the user to include accepted files.

LFITester

LFITester is a Python3 program that automates the detection and exploitation of Local File Inclusion (LFI) attacks on a server.

It uses many different attacks mentioned in the next slide to find the vulnerability and exploit it returning a reverse shell to us.

https://github.com/kostas-pa/LFITester

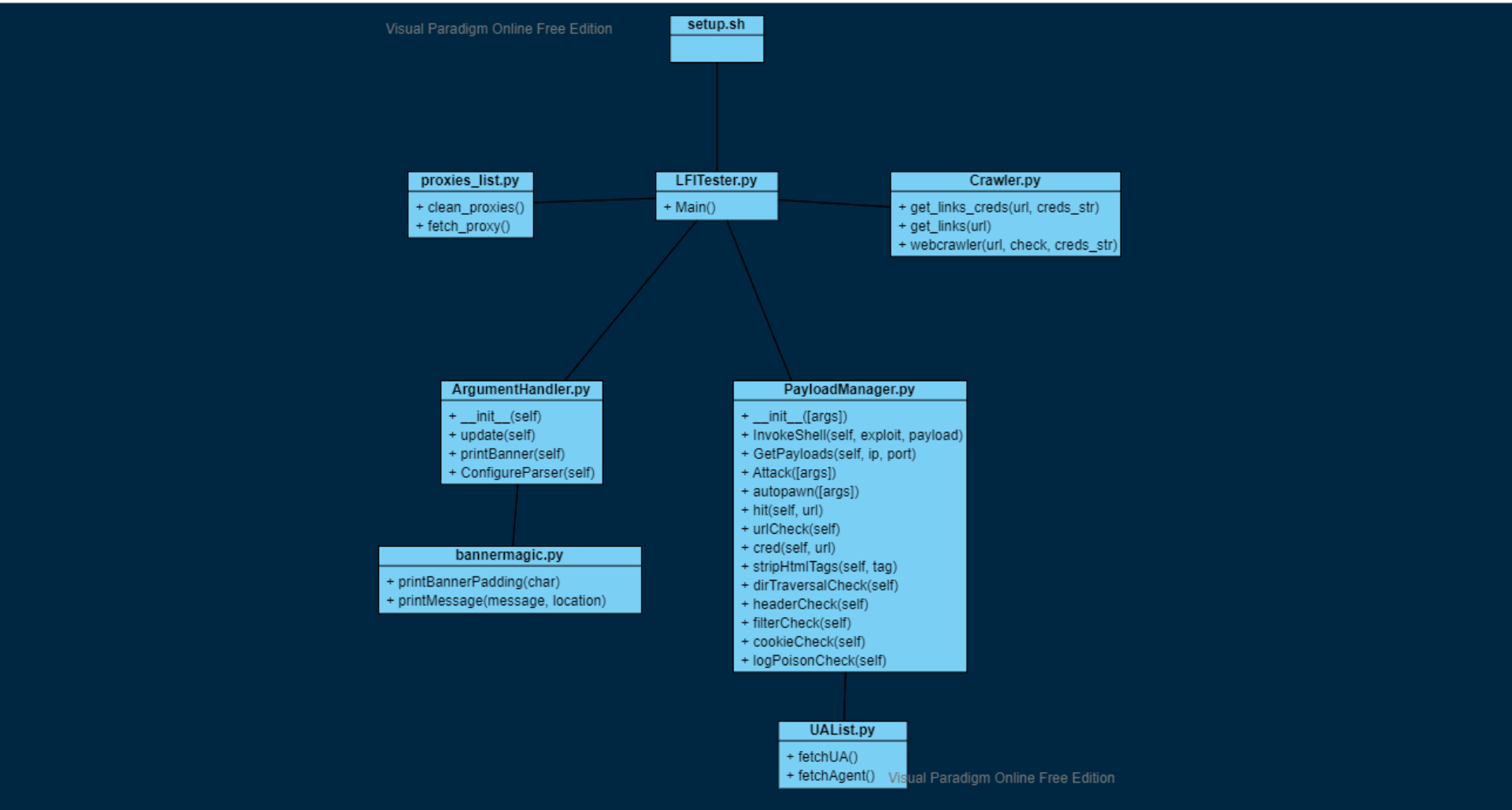
Featured Attacks

- Directory Traversal = ../../../../etc/passwd
- Bypassing Filters = URL encoding, ….//, /../, .././, ../etc/passwd%00, ../etc/passwd%2500, php://filter/read=convert.base64-encode/resource=../etc/passwd
- Log Poisoning
- PHP Wrappers = expect://[command], data://text/plain;base64,[<?php system($_GET[‘cmd’]); ?>]&cmd=[command]
- PHP Session Cookies = [<?php system($_GET[‘cmd’]); ?> and /var/lib/php/session/sess_[value]&cmd=[command] path
- Combination of all the above

Features

- We can use the **autopwn** feature to automatically attempt to exploit any valid indications of LFI.
- In order to avoid WAF protection, we implemented a **proxy** feature that will send each request from a different proxy IP address.
- It is possible to automate the endpoint detection by using the **crawler** feature that will automatically detect any LFI attack vector.
- Finally one more interesting feature, is the **creds** option which is used to login to a given webpage if login is required.

Structure


Visual Paradigm Online Free Edition
setup.sh
proxies_list.py
+ clean_proxies()
+ fetch_proxy()
LFITester.py
+ Main()
Crawler.py
+ get_links_creds(url, creds_str)
+ get_links(url)
+ webcrawler(url, check, creds_str)
ArgumentHandler.py
+ __init__(self)
+ update(self)
+ printBanner(self)
+ ConfigureParser(self)
PayloadManager.py
+ __init__([args])
+ InvokeShell(self, exploit, payload)
+ GetPayloads(self, ip, port)
+ Attack([args])
+ autopawn([args])
+ hit(self, url)
+ urlCheck(self)
+ cred(self, url)
+ stripHtmlTags(self, tag)
+ dirTraversalCheck(self)
+ headerCheck(self)
+ filterCheck(self)
+ cookieCheck(self)
+ logPoisonCheck(self)
bannermagic.py
+ printBannerPadding(char)
+ printMessage(message, location)
UAList.py
+ fetchUA()
+ fetchAgent()
Visual Paradigm Online Free Edition