

PRESS REVIEW ARCHIVE

Digital Media Monitoring & Documentation Service

Source URL: <https://security.stackexchange.com/questions/17407/how-can-i-use-this-path-bypass-exploit-local-file-inclusion>

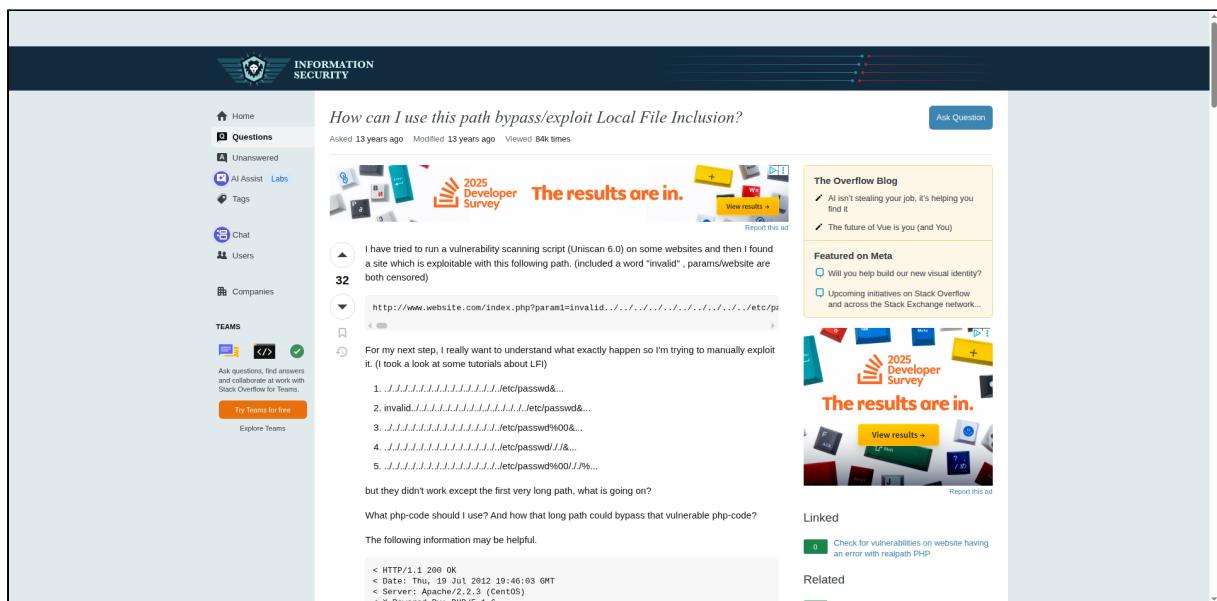
Archived Date: August 15, 2025 at 15:22

Published: July 19, 2012

Document Type: Web Page Archive

Wayback Machine: https://web.archive.org/web/*/https://security.stackexchange.com/questions/17407/how-can-i-use-this-path-bypass-exploit-local-file-inclusion

Page Screenshot



How can I use this path bypass/exploit Local File Inclusion?

Asked 13 years ago Modified 13 years ago Viewed 84k times

▲ I have tried to run a vulnerability scanning script (Uniscan 6.0) on some websites and then I found a site which is exploitable with this following path.
(included a word "invalid" , params/website are both censored)

32

```
ncpev@192.168.1.100:~$ cat /etc/passwd
param=invalid...../etc/passwd.....
```

For my next step, I really want to understand what exactly happen so I'm trying to manually exploit it. (I took a look at some tutorials about LFI)

- [illegible]

but they didn't work except the first very long path, what is going on?

What php-code should I use? And how that long path could bypass that vulnerable php-code?

The following information may be helpful.

```
< HTTP/1.1 200 OK
< Date: Thu, 19 Jul 2012 19:46:03 GMT
< Server: Apache/2.2.3 (Centos)
< X-Powered-By: PHP/5.1.6
< Set-Cookie: PHPSESSID=[blah-blah]; path=/
< Expires: Thu, 19 Nov 1981 08:52:00 GMT
  Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
< Pragma: no-cache
< Vary: Accept-Encoding
< Content-Length: 2094
< Content-Type: text/html
```

web-application php exploit


Share Improve this question Follow

edited Jul 20, 2012 at 17:38

 Rory Alsop ♦
61.7k 12 122 328

asked Jul 19, 2012 at 20:10

Smile.Hunter
599 1 5 10

- 1 I hope you are being ethical. – [Kurt](#) Jul 20, 2012 at 1:46
- 2 I want to be security researcher/exploit developer, If I just want to hack then I won't research about it. – [Smile.Hunter](#) Jul 20, 2012 at 6:20 
- 3 I think that what Prowla is saying is that you should do research on your own website. You can for example have a look at OWASP BWA project. – [Gael Muller](#) Jul 20, 2012 at 15:53

@Smile.Hunter exploiting vulnerabilities in the wild, even if you don't plan to compromise the target, is a really bad idea, and if you get caught, it will be way more difficult to work in the infocsec industry and be considered trustable. If you wish to train yourself, there are a bunch of challenge websites or downloadable vulnerable applications on which you'd be able to make experiments without breaking the law. I can't suggest you enough to follow this path instead of training on live production websites.

— mdeous Feb 15, 2016 at 20:01

3 Answers

Sorted by: Highest score (default)

▲ Fascinating! @catalyze has dug up a truly intriguing, lovely situation here. I wanted to take the time to summarize what's going on here, on this site. (Full credits to @catalyze and Francesco "ascii" Ongaro; I'm just summarizing what they explained.)

38

exploited through standard LFI methods; you need more trickiness to work out how to exploit it.

Background. First, I need to tell you two facts about PHP's file handling that were discovered by Francesco "ascii" Ongaro and others:

- **Fact 1. You can add stuff to the end of a filename.** Everyone knows that `./etc/passwd` is just another way to refer to the `/etc/passwd` file. But, here are some you may not have known about.

On PHP, it turns out that `/etc/passwd/` also refers to the `/etc/passwd` file: trailing slashes are stripped off. Wild, huh? This doesn't work on base Unix, so it is a bit surprising that PHP would accept such a filename, but it appears that PHP is itself stripping off trailing slashes before opening the file.

You can append any number of trailing slashes: `/etc/passwd////` is also OK.

And, you can append `./` (as many times as you want). For instance, `/etc/passwd/./`, `/etc/passwd/././`, and `/etc/passwd/./././` all refer to `/etc/passwd`. Go nuts! PHP doesn't care.

- **Fact 2. Long paths are truncated.** On most PHP installations, if the filename is longer than 4096 bytes, it will be silently truncated and everything after the first 4096 bytes will be discarded. No error is triggered: the excess characters are simply thrown away and PHP happily continues on.

The attack. Now I am ready to describe the attack. I'll show you the vulnerable code, why standard LFI attacks don't work, and then how to build a more-clever attack that does work. The result explains what @catalyze saw in his pentest.

The vulnerable code. Suppose we have code that looks something like this:

```
<?php
include("includes/".$_GET['param1'].".php");
?>
```

This looks like a local file include (LFI) vulnerability, right? But the situation is actually a bit trickier than it may at first appear. To see why, let's look at some attacks.

Standard attacks. The standard, naive way to try to exploit this LFI vulnerability is to supply a parameter looking something like `?param=../../../../var/www/shared/badguy/evil`. The above PHP code will then try to include the file `includes/../../../../var/www/shared/badguy/evil.php`. If we assume that the file `/var/www/shared/badguy/evil.php` exists and is controlled by the attacker, then this attack will succeed at causing the application to execute malicious code chosen by the attacker.

But this only works if the attacker can introduce a file with contents of his choice onto the filesystem, with a filename ending in `.php`. What if the attacker doesn't control any file on the filesystem which ends in `.php`? Well, then, the standard attacks will fail. No matter what parameter value the attacker supplies, this is only going to open a filename that ends with the `.php` extension.

A more sophisticated attack. With the additional background facts I gave you earlier, maybe you can see how to come up with a more sophisticated attack that defeats this limitation.

Basically, the attacker chooses a very long parameter value, so that the constructed filename is longer than 4096 bytes long. When the filename is truncated, the `.php` extension will get thrown away. And if the attacker can arrange for the resulting filename to refer to an existing file on the filesystem, the attacker is good.

Now this might sound like a far-fetched attack. What are the odds that we can find a filename on the filesystem whose full path happens to be exactly 4096 bytes long? Maybe not so good?

This is where the background facts come into play. The attacker can send a request with `?param=../../../../etc/passwd/../../../../<...>` (with the `./` pattern repeated many thousands of times). Now look at what filename gets included, after the prefix is prepended and the `.php` file extension is added: it will be something like `includes/../../../../etc/passwd/../../../../<...>.php`. This filename will be longer than 4096 bytes, so it will get truncated. The truncation will drop the file extension and leave us with a filename of the form `includes/../../../../etc/passwd/../../../../<...>`. And, thanks to the way PHP handles trailing slashes and trailing `./` sequences, all that stuff at the end will be ignored. In other words, this filename will be treated by PHP as equivalent to the path `includes/../../../../etc/passwd`. So PHP will try to read from the password file, and when it finds PHP syntax errors there, it may dump the contents of the password file into an error page -- disclosing secret information to an attacker.

So this technique allows to exploit some vulnerabilities that otherwise could not be exploited through standard methods. See the pages that @catalyze links to for a more detailed discussion and many other examples.

This also explains why @catalyze was not able to exploit the attack by sending something like `?param=../../../../etc/passwd:a`. `.php` extension got added on, and the file `/etc/passwd.php` did not exist, so the attack failed.

Summary. Peculiarities in PHP's handling of file paths enable all sorts of subtle attacks on vulnerabilities that otherwise would appear unexploitable. For pentesters, these attack techniques may be worth knowing about.

For developers, the lesson is the same: validate your inputs; don't trust inputs supplied by the attacker; know about classic web vulnerabilities, and don't introduce them into your code.

Share Improve this answer Follow

edited Jul 30, 2012 at 3:29

answered Jul 30, 2012 at 3:16

 D.W.
101k ● 34 ● 282 ● 615

3 then, what I want to know is which PHP versions are affected? Path normalization issue not occur in my Apache/2.2.22 (Ubuntu) + PHP 5.3.10-1ubuntu3.2 with Suhosin-Patch (built: Jun 13 2012 17:19:58) – [Smile.Hunter](#) Jul 30, 2012 at 17:24

3 @catalyze is now Smile.Hunter (user 11373) – [serv-inc](#) Jul 31, 2015 at 11:17

3 Isn't it easier to do `../../../../target%00`? – [user50312](#) Aug 22, 2015 at 1:54

It's possible that the website filters the `%00` but doesn't filter for this kind of attack – [Jannes Braet](#) Mar 29, 2018 at 12:15



Finally, I found the solution!



This LFI's bypass techniques are called **Path Truncation attack**



Scenario:



- No white/black lists, `open_base_dir` or any restrict access configuration
- There is `magic_quotes_escape` nullbytes as addslashes() is implicitly called on all GPC and SERVER inputs. (in this case `etc/passwd%00` would become `etc/passwd%0`, so it cannot evaluate as correct file.)
- `include_path` (within `php.ini`) contains at last one **absolute path** to trigger a part of complex vulnerable in sourcecode of PHP (for example, `include_path = ".:usr/share/php"`)
- PHP < ? (Who know?)

Payload:

- Has to start with a non-existing directory
- Continue with the traversal sled, point to the path to include
- End with the normalization/truncation sled.

Smart people are here..

<http://www.ush.it/2009/02/08/php-file-system-attack-vectors/>

<http://www.ush.it/2009/07/26/php-file-system-attack-vectors-take-two/>

Share Improve this answer Follow

edited Jul 29, 2012 at 22:53

answered Jul 29, 2012 at 21:32

 Smile.Hunter
599 ● 1 ● 5 ● 10

2 Wow! Brilliant catch, @catalyze. That explains everything you saw. P.S. You might notice that the attack string that does work is exactly 4096 bytes long -- exactly as long as it needs to be to exploit the path truncation property, and no longer. Neat. – [D.W.](#) Jul 30, 2012 at 3:31



I am going to answer this question with the caveat that I am making an assumption this is used for legal purposes, and for security research only.

If we're talking about a PHP website, this is probably what's happening in the backend:

```
$file = fopen($_GET["param"], "r");  
/* Do some operation on the file handler, like maybe read the file and output it */  
$contents = fread($file, $size);  
print $contents
```

You could potentially exploit this LFI to upload your webshell, and run system commands on the web shell. The simplest way to do this is to inject into access.log, and accessing access.log. The simplest way to do this is to modify the User Agent, or maybe even the GET request, to include some PHP code that would help you setup a stager. For example, a telnet into the website, and the following request, should inject into access.log:

```
GET/ <?php phpinfo() ?>
```

Obviously, all this will do is get you the PHP Information from access.log, but you get the idea. Now, on the same lines, you could easily do something like:

```
GET/ <?php data = $_REQUEST['data']; $filename = $_REQUEST['filename']; file_put_contents($filename,base64_decode($data)); ?>
```

and then upload a base64 encoded PHP script into it, and get your web shell up there. I'll leave it up to you to figure out the rest of it, it shouldn't be hard at all. There's a really multi-part tutorial for this on [Kaotic Creations](#) that you should really read, if you are interested in learning more about this.

Share Improve this answer Follow

answered Jul 21, 2012 at 8:19



Karthik Rangarajan
830 ● 6 ● 18

- 2 This is very useful stuffs but I want to know how that very long path `../../../../etc/passwd../../../../blah blah` could included (exploit with LFI) but `../../../../etc/passwd` or `../../../../passwd%00` did not included – [Smile.Hunter](#) Jul 21, 2012 at 18:33 ✓
- I have no idea why the LFI param is that way, it doesn't even make sense to have it that way. Removing the `../../../../` etc., should work perfectly, so without actually looking at the specific issue at hand, I wouldn't be able to give you a good answer. – [Karthik Rangarajan](#) Jul 22, 2012 at 7:30

Start asking to get answers

Find the answer to your question by asking.

Ask question

Explore related questions

[web-application](#) [php](#) [exploit](#)

See similar questions with these tags.