psychoPATH – usage scenarios

Table of contents

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.

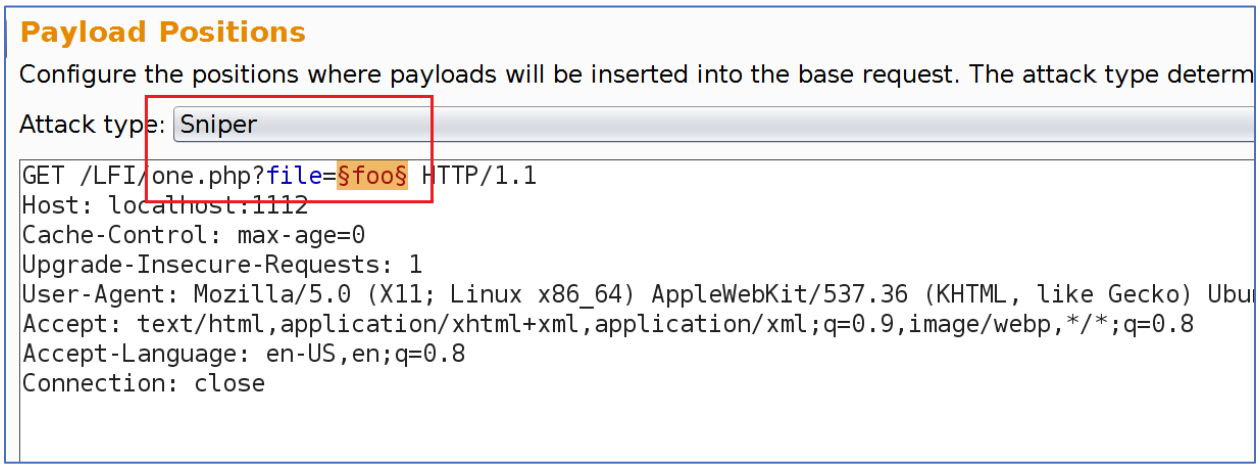
Below is a short presentation on how all three can be quickly detected with the payloads provided by psychoPATH.

test_cases/LFI/one.php:

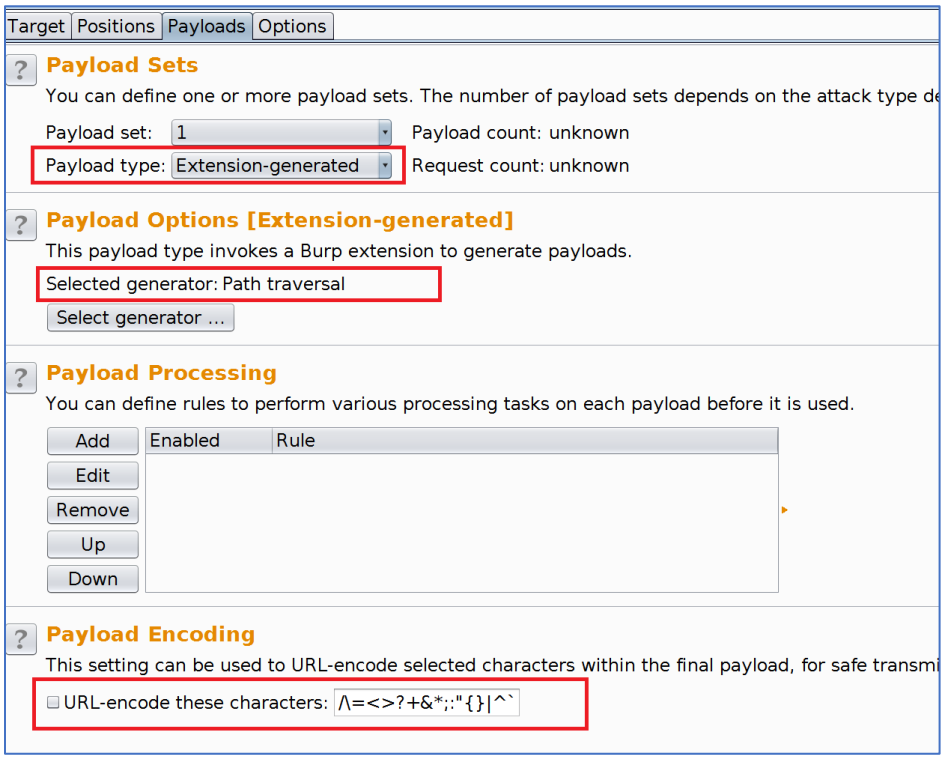
```
<?php

if(isset($_GET['file']))
{
        $file=str_replace('../','',$_GET['file']);
        echo @file_get_contents('./'.$file);
}
#removing only ../
?>
```

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:



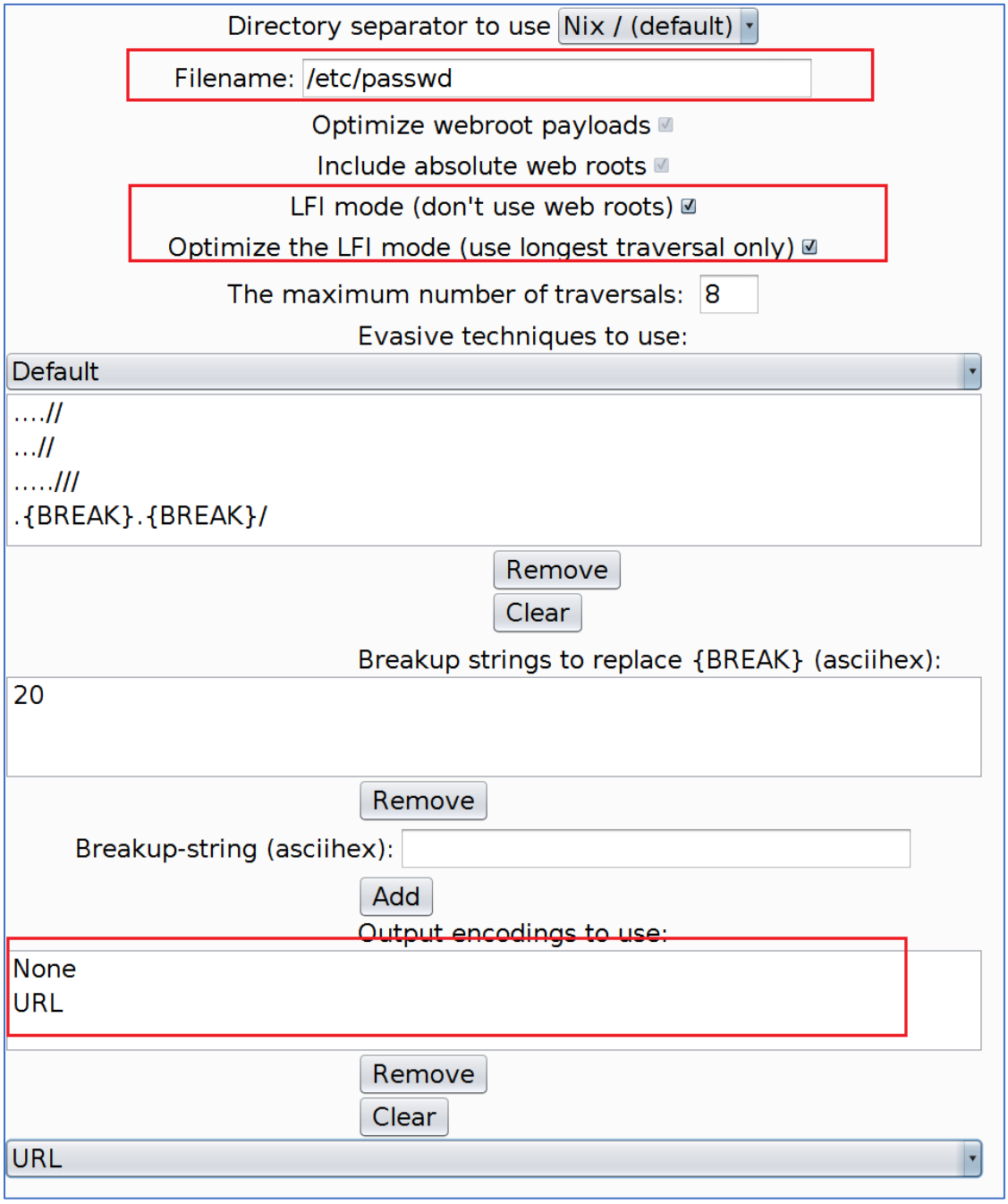
In payload options, we choose *Extension generated -> Path traversal* payload type:



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

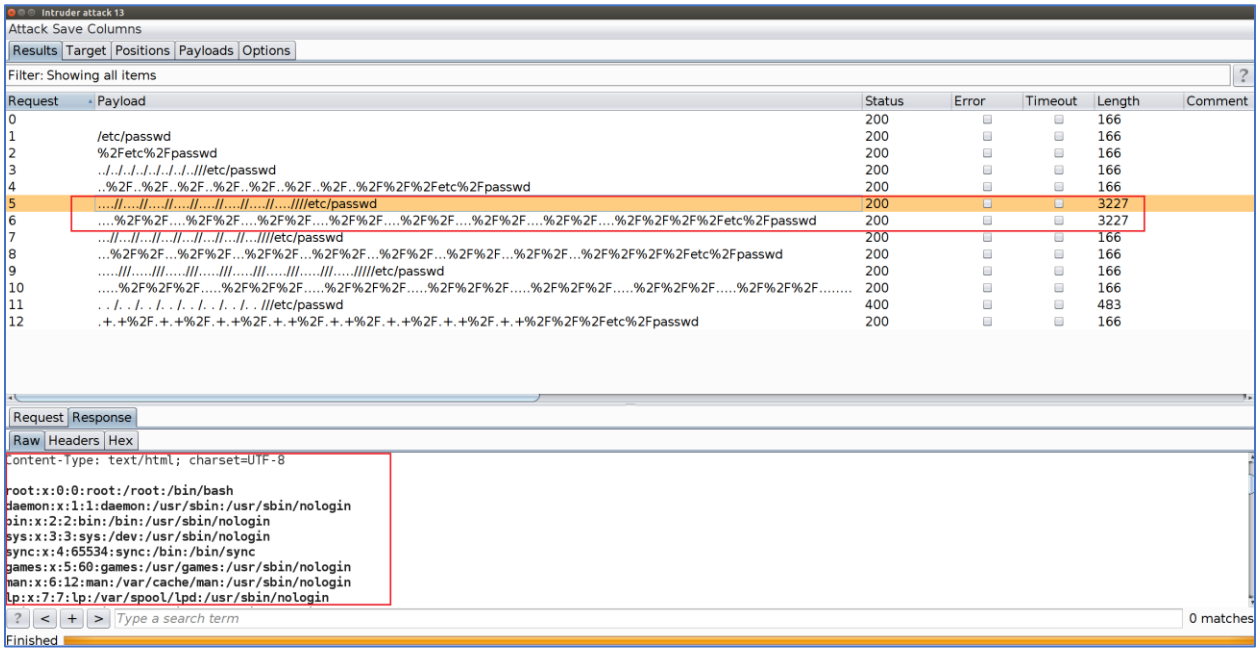
Now let's move to the psychoPATH configuration panel. We choose the file name we would like to read, for instance /etc/passwd.

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):



Also, it is safe to choose two output encoding modes; *None* and *URL-encoding*.

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



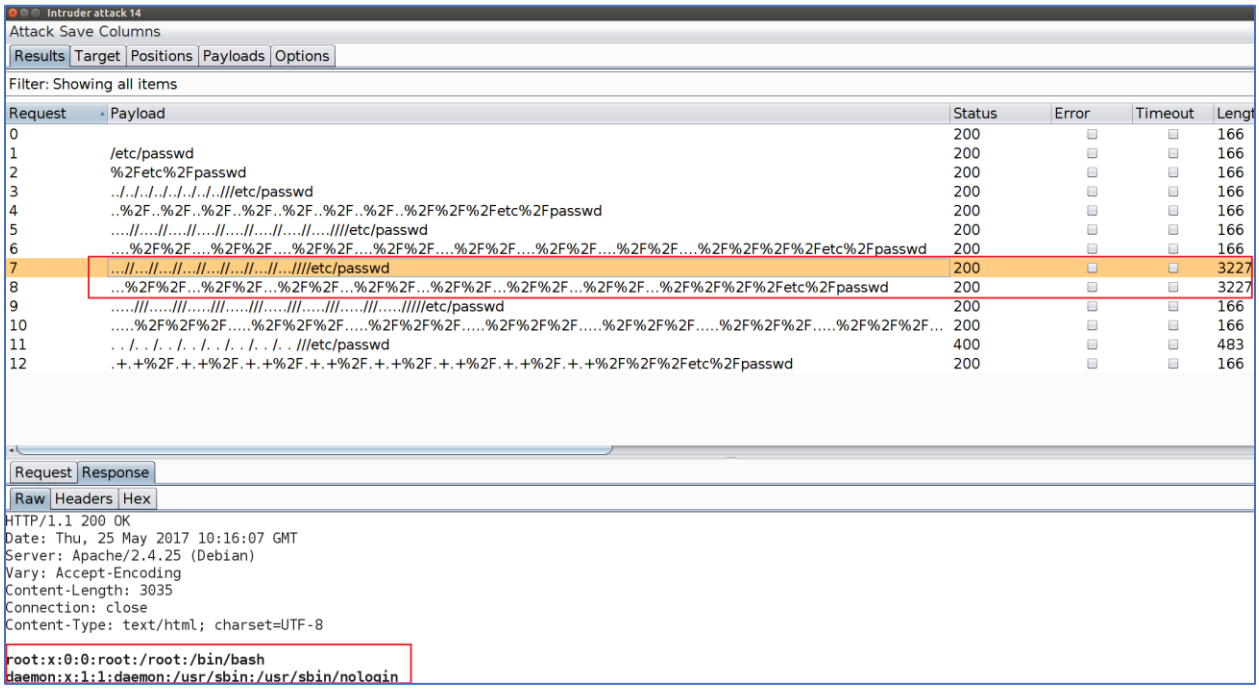
As it can be seen in Intruder's output, two variants (URL-encoded and raw) of the non-recurrent evasive payload ….// hit the file.

Let's move on to another example.

two.php:

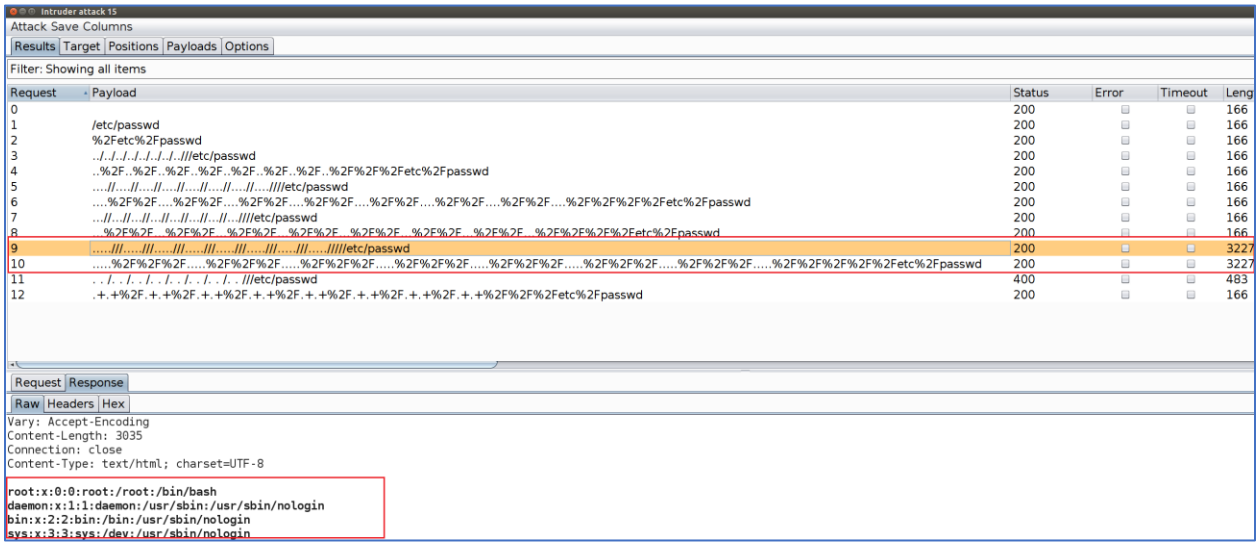
```php
<?php

if(isset($_GET['file']))
{
        $file=str_replace('./','',$_GET['file']);
        echo @file_get_contents('./'.$file);
}
#removing only ./
?>
```

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



three.php:

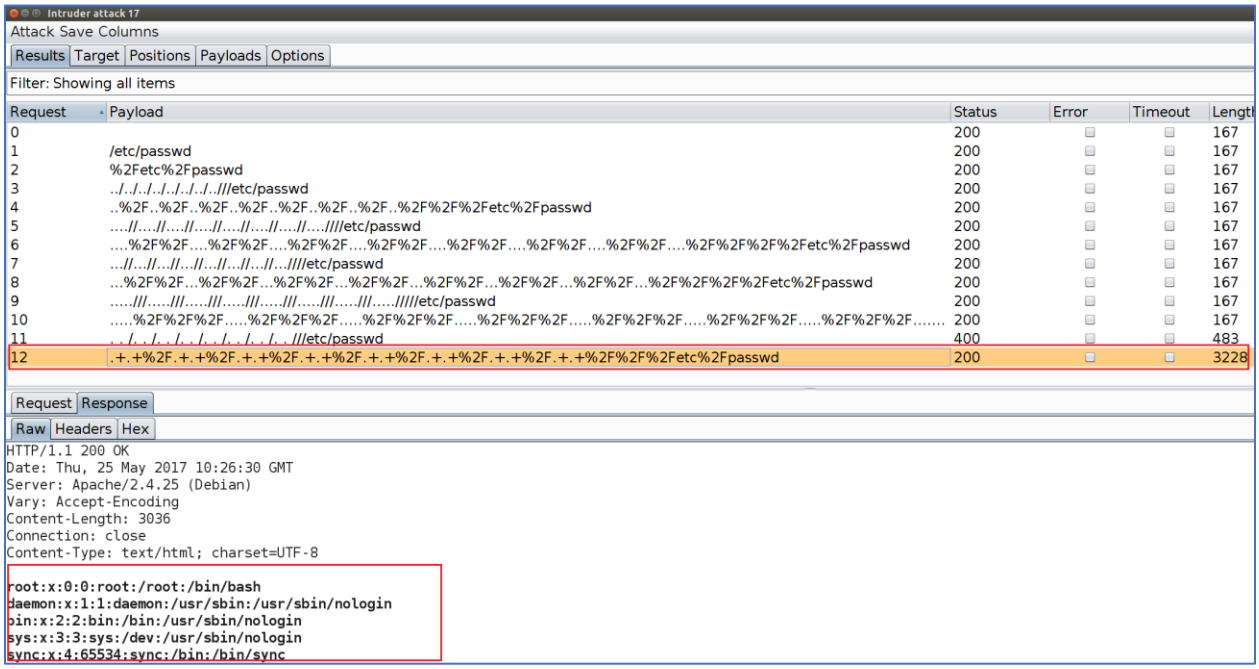
```php
<?php

if(isset($_GET['file']))
{
        $file=str_replace('../','',$_GET['file']);
        $file=str_replace('./','',$file);
        echo @file_get_contents('./'.$file);
}
#removing ../ then ./
?>
```



four.php:

```php
<?php

if(isset($_GET['file']))
{
        $file=str_replace('..','',$_GET['file']);
        $file=str_replace(' ','',$file); // removing white spaces
        echo @file_get_contents('./'.$file);
}
#removing .. then white spaces
?>
```

And the winner is:



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

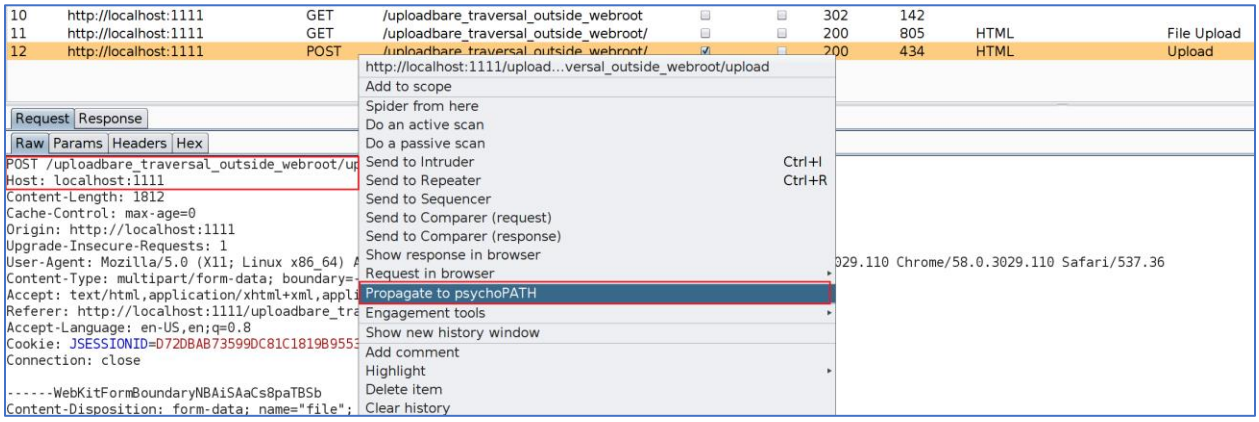
longer than 45 characters and the upload directory is */tmp* - in that case *...../// /var/lib/tomcat8/webapps/upload* would do the trick instead. If you are worried about the payload length and you care less about the number of payloads, turn optimization off.

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.

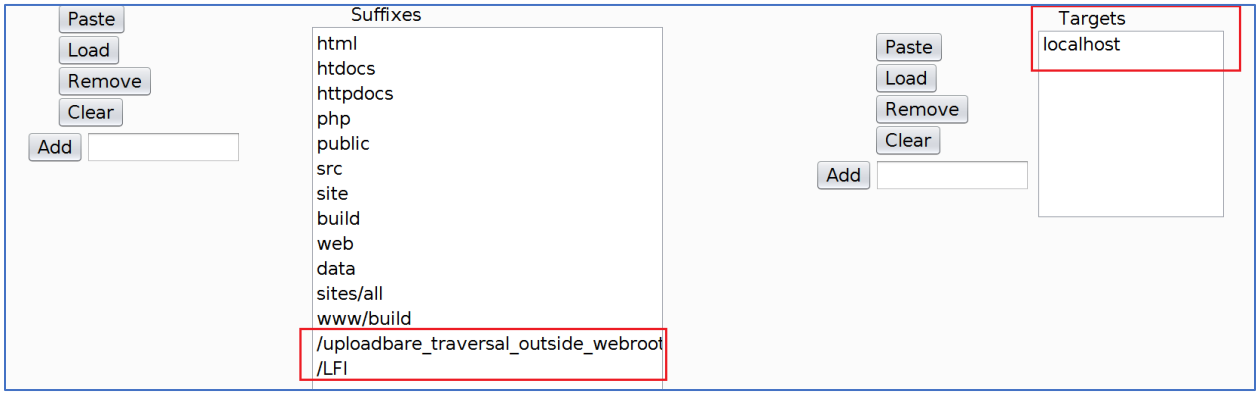
Let's see some examples.

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.).

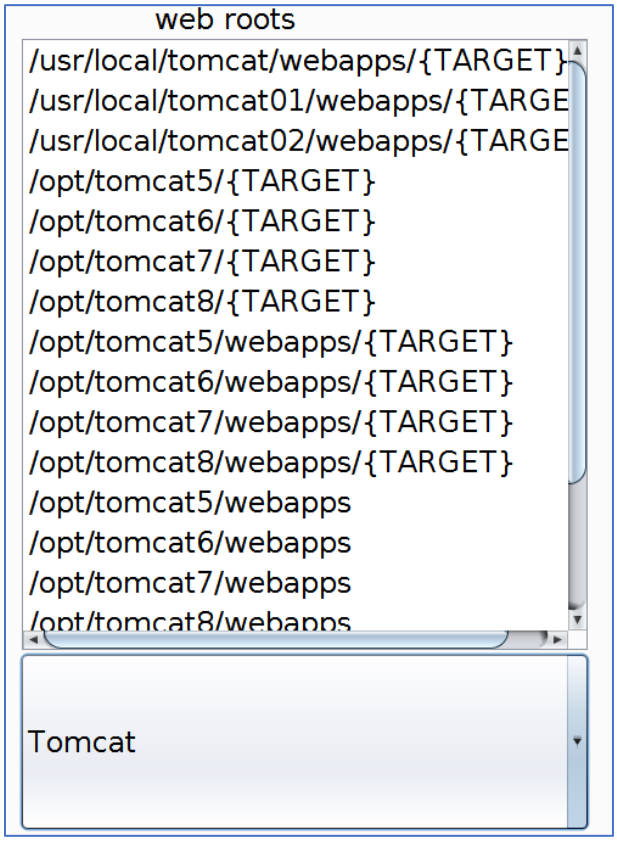
Then, we use the *Propagate to psychopath* option to feed the plugin with the data from the site map and with the host header:



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*:



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



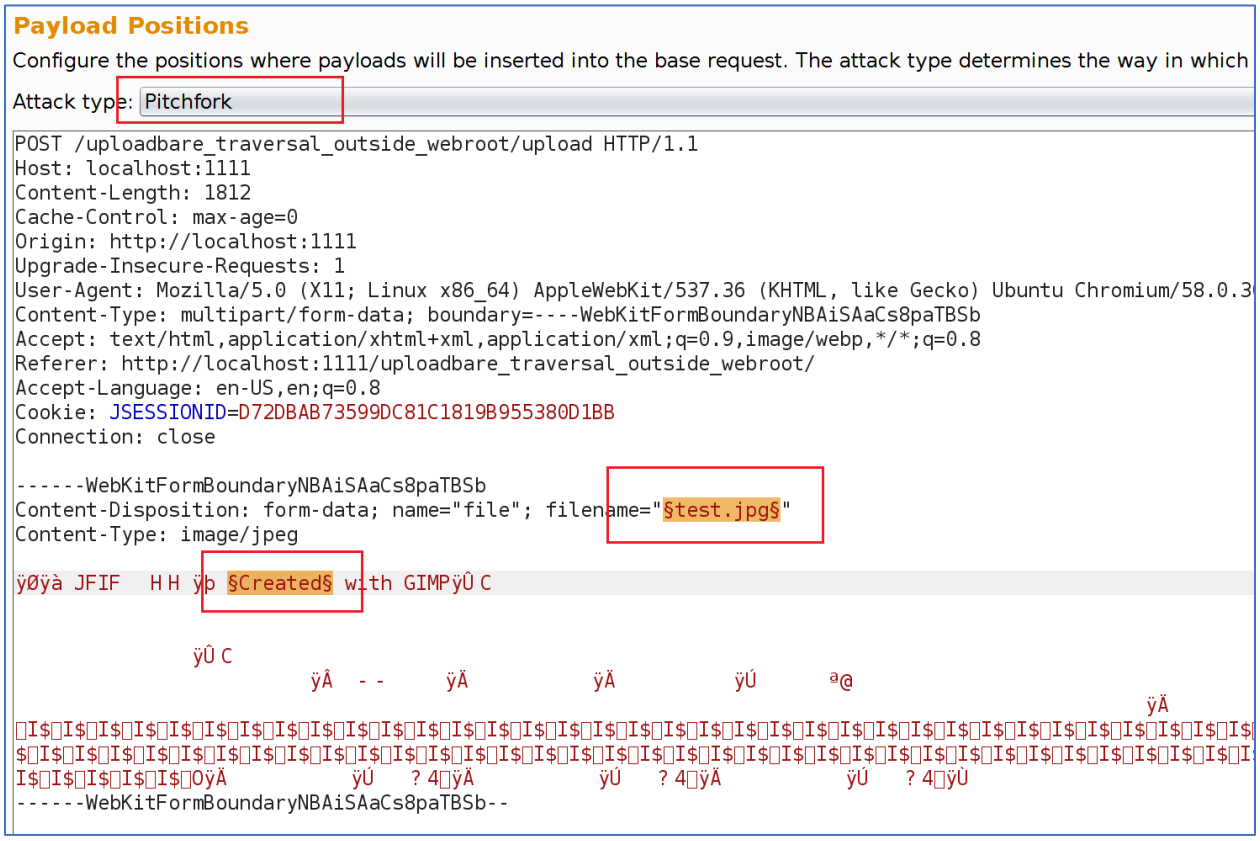
We send the file upload request to Intruder.

We set the attack type to Pitchfork.

Then we select two payload holders.

One is the file name value we are about to inject into.

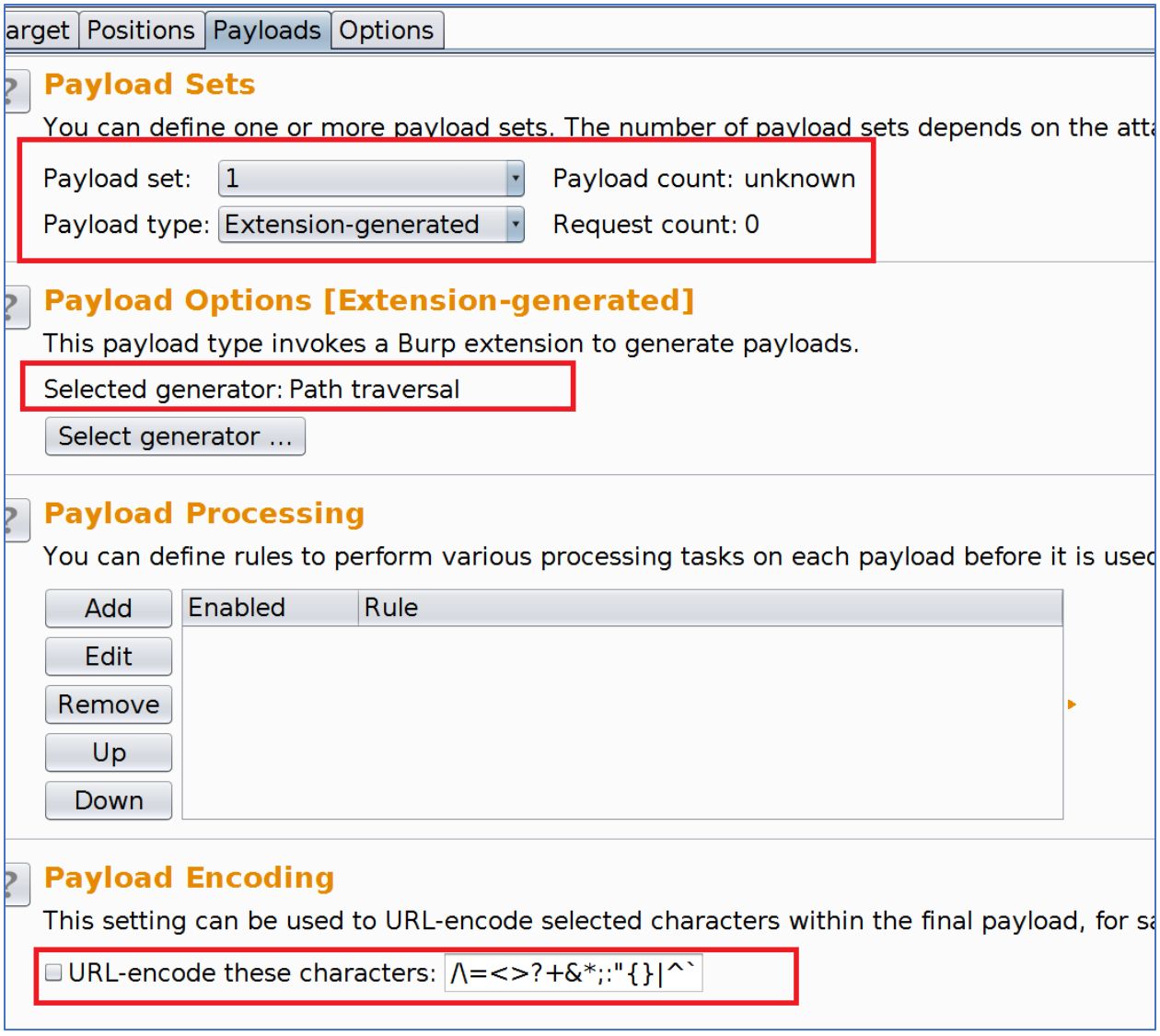
The other is just a section in the uploaded file content - this is where the unique payload mark will be put. Currently payload marks have fixed length of 7 characters. When injecting into image formats, the safest way is to mark exactly seven characters of a string - e.g. a piece of *exif* data. This way we won't encounter false negatives if the application is checking the validity of the image file before putting it into the upload directory of our interest:



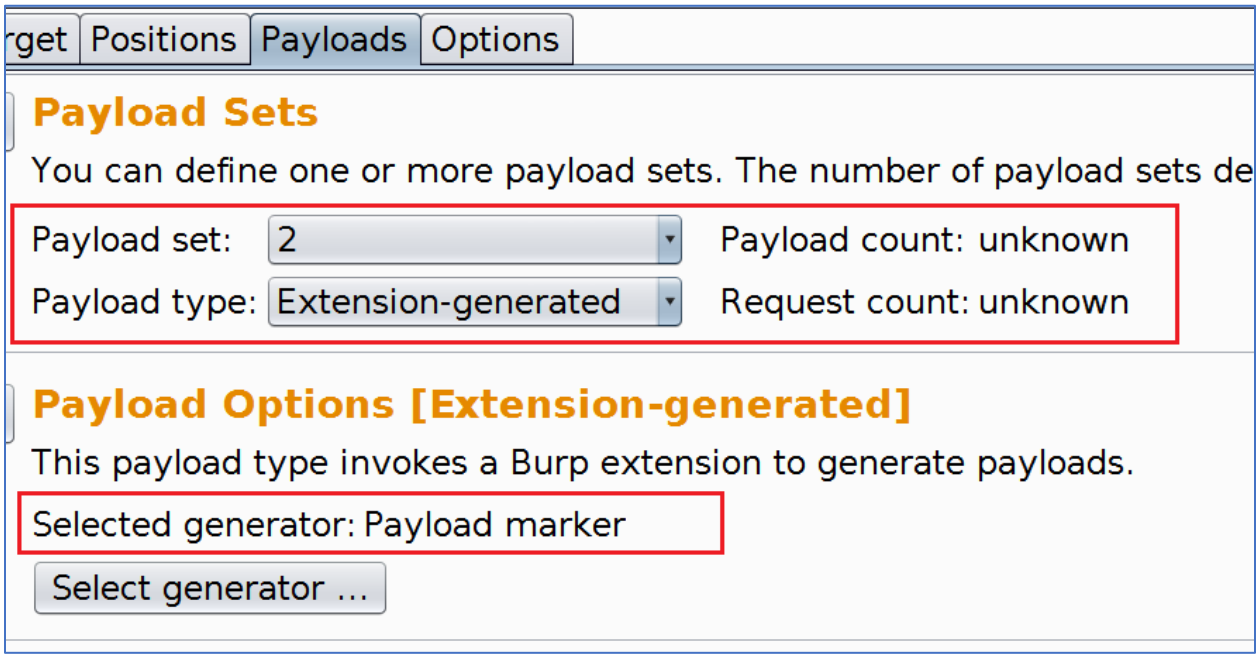
Then we move to the *Payloads* tab.

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

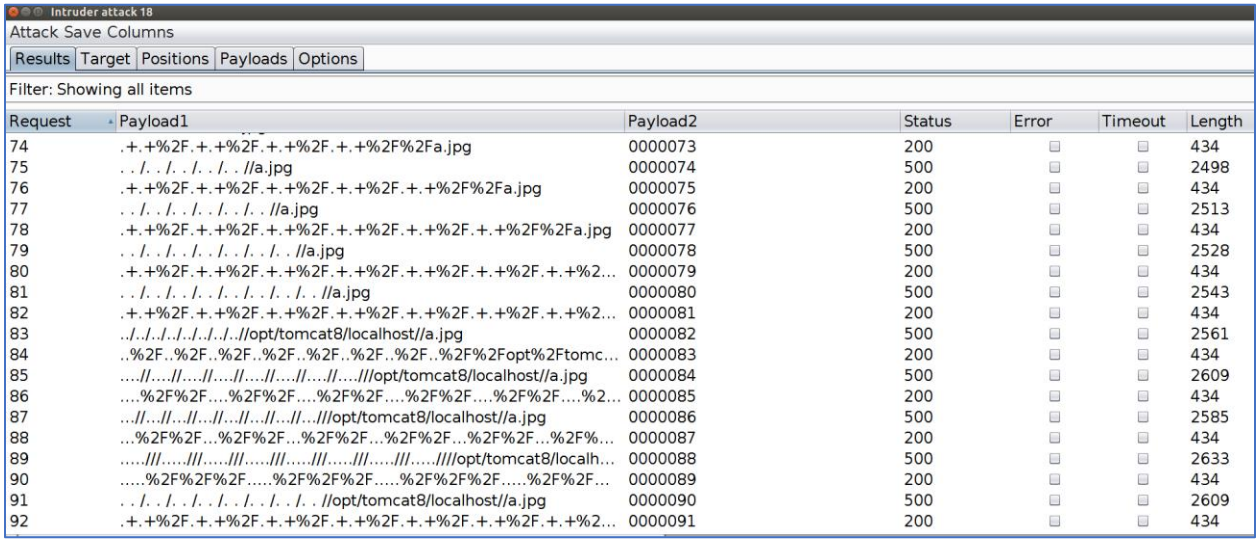
Again, we UNCHECK the *URL-encode these characters* box:



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



We start the attack:



Request	Payload1	Payload2	Status	Error	Timeout	Length
74	.+.+%2F.+.+%2F.+.+%2F.+.+%2F%2Fa.jpg	0000073	200			434
75	. . /. . /. . /. . /. . //a.jpg	0000074	500			2498
76	.+.+%2F.+.+%2F.+.+%2F.+.+%2F.+.+%2F%2Fa.jpg	0000075	200			434
77	. . /. . /. . /. . /. . /. . //a.jpg	0000076	500			2513
78	.+.+%2F.+.+%2F.+.+%2F.+.+%2F.+.+%2F.+.+%2F%2Fa.jpg	0000077	200			434
79	. . /. . /. . /. . /. . /. . /. . //a.jpg	0000078	500			2528
80	.+.+%2F.+.+%2F.+.+%2F.+.+%2F.+.+%2F.+.+%2F.+.+%2...	0000079	200			434
81	. . /. . /. . /. . /. . /. . /. . /. . //a.jpg	0000080	500			2543
82	.+.+%2F.+.+%2F.+.+%2F.+.+%2F.+.+%2F.+.+%2F.+.+%2...	0000081	200			434
83	../../../../../../../..//opt/tomcat8/localhost//a.jpg	0000082	500			2561
84	..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F%2Fopt%2Ftomc...	0000083	200			434
85	....//....//....//....//....//....//....///opt/tomcat8/localhost//a.jpg	0000084	500			2609
86	....%2F%2F....%2F%2F....%2F%2F....%2F%2F....%2F%2F....%2...	0000085	200			434
87	.../.../.../.../.../.../.../.../.../.../.../.../opt/tomcat8/localhost//a.jpg	0000086	500			2585
88	...%2F%2F...%2F%2F...%2F%2F...%2F%2F...%2F%2F...%2F%...	0000087	200			434
89	...../////...../////...../////...../////...../////...../////...../////.../////opt/tomcat8/localh...	0000088	500			2633
90	.....%2F%2F%2F.....%2F%2F%2F%2F.....%2F%2F%2F%2F.....%2F%2F%2F...	0000089	200			434
91	. . /. . /. . /. . /. . /. . /. . /. . //opt/tomcat8/localhost//a.jpg	0000090	500			2609
92	.+.+%2F.+.+%2F.+.+%2F.+.+%2F.+.+%2F.+.+%2F.+.+%2F.+.+%2...	0000091	200			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:

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: 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: Directory checker

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 safe transmission within HTTP requests.

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

We run the attack and watch the result:

Intruder attack 19

Attack Save Columns

Results | Target | Positions | Payloads | Options

Filter: Showing all items

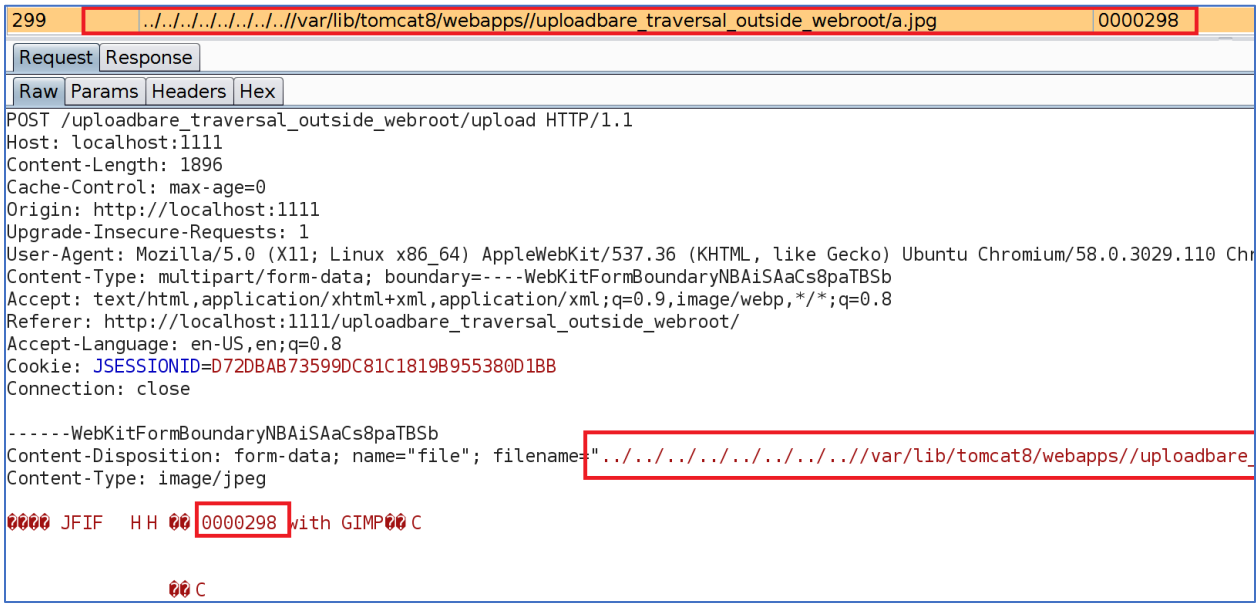
Request	Payload	Status	Error	Timeout	Length
0		200	☐	☐	805
1	/a.jpg	404	☐	☐	1240
2	/uploadbare_traversal_outside_webroot/a.jpg	200	☐	☐	1849
3	/LFI/a.jpg	404	☐	☐	1244

Request | Response

Raw | Headers | Hex | Render

```
HTTP/1.1 200
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
Connection: close

JFIF  H H  0000298 with GIMP C
```

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

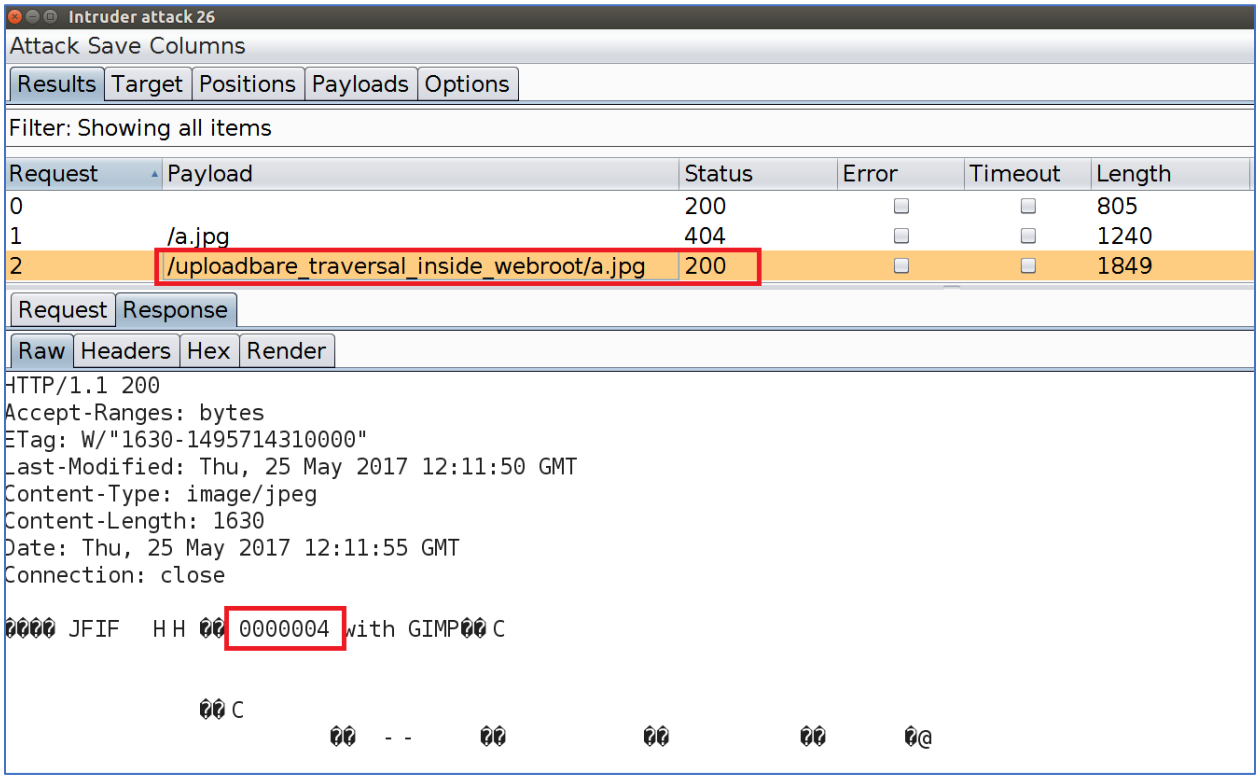


Now we know the golden payload to reach the document root was *./../../../../../../..//var/lib/tomcat8/webapps//uploadbare_traversal_outside_webroot/a.jpg*.

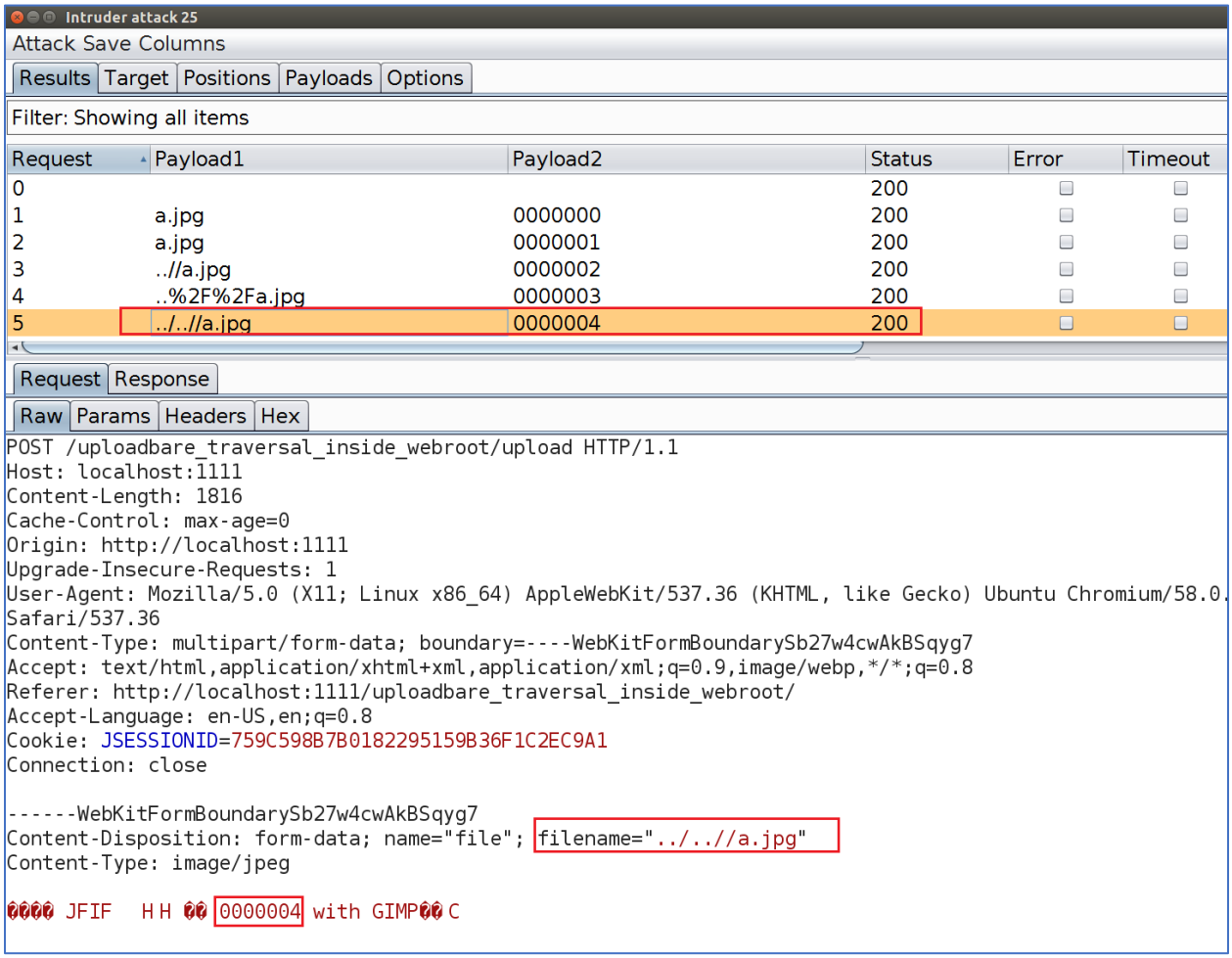
For other two examples, the results for the payloads that have worked, would look as follows, respectively.

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:

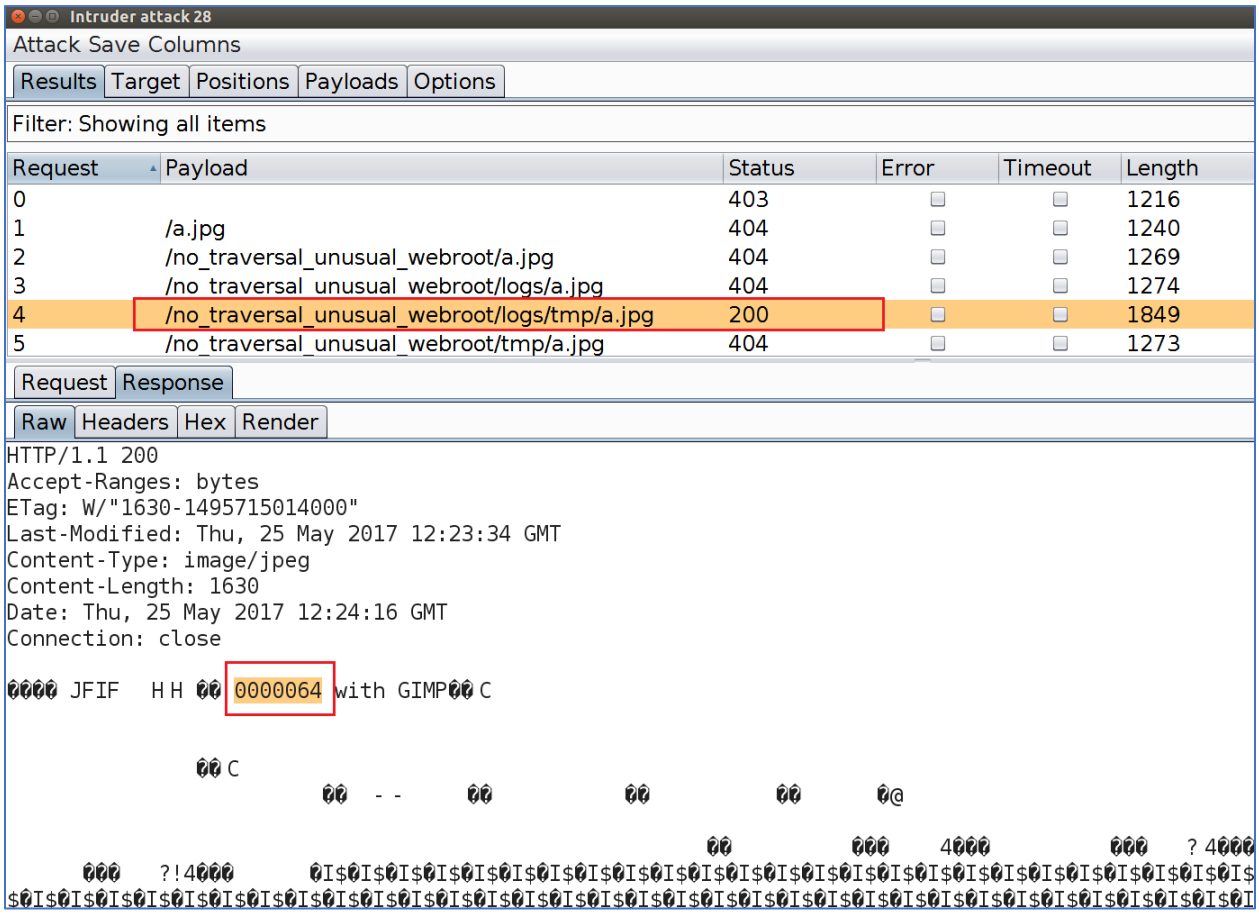


The golden payload:

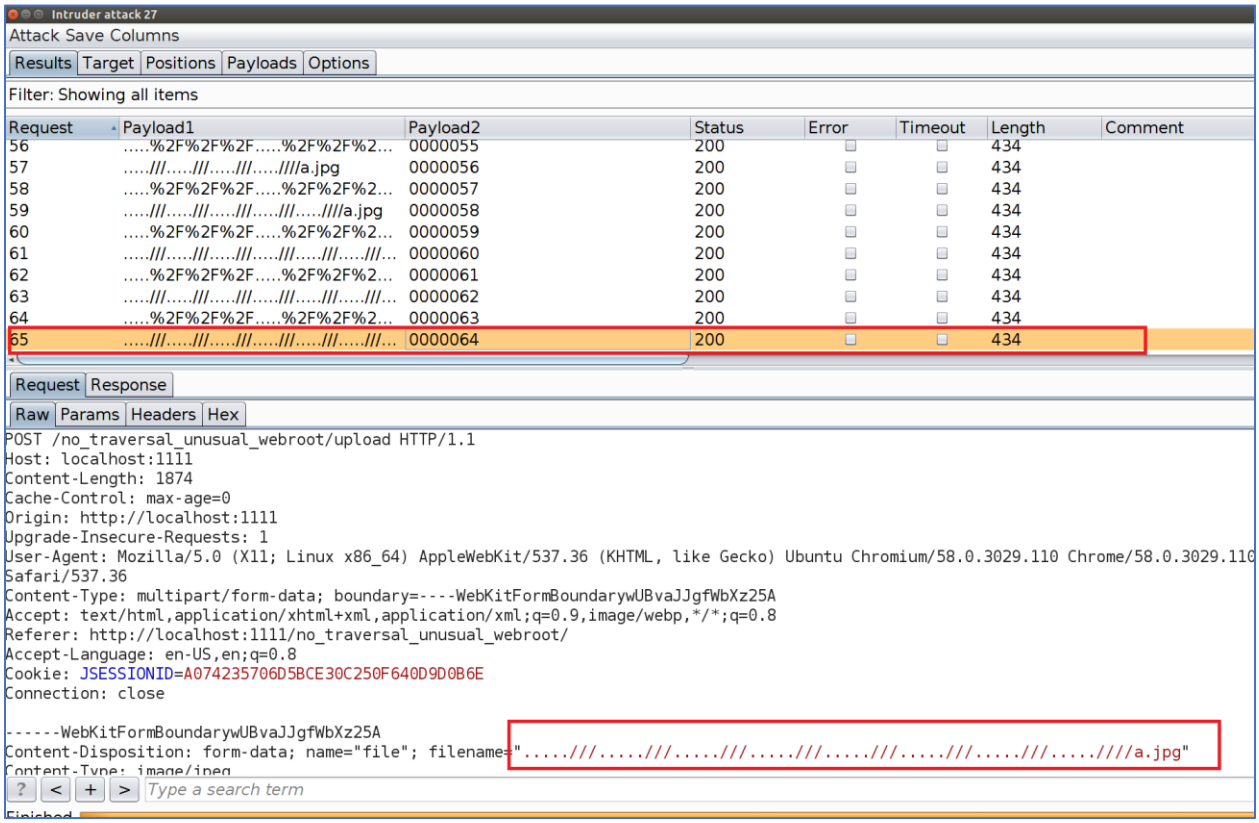


Case 3: no_traversal_unusual_webroot

Verification:



The golden payload(s):



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!