



eLearnSecurity
Forging security professionals

BURP SUITE BASICS



PRELIMINARY SKILLS | SECTION 1 MODULE 3 | LAB #4

LAB



1. SCENARIO

A client provides you with a URL to a web application running on a remote server. The client wants to know if there are any sensitive resources exposed.

Use Burp Suite to identify if a sensitive resource was left unprotected by developers.

Note: It is preferred that you perform this lab from inside a Kali, or another pentesting distribution, Virtual Machine.

2. LEARNING OBJECTIVES

In this Lab, you will learn how to:

- Use Burp Suite intruder
- Use Burp Suite repeater
- Learn how to find hidden resources on web application

3. RECOMMENDED TOOLS

- Kali Linux
- Web browser (e.g., Firefox)
- Burp Suite community edition:
(<https://portswigger.net/burp/communitydownload>)

4. NETWORK CONFIGURATION

- Intranet Subnet: **172.16.160.0/24**
- Target application: **http://172.16.160.102**



5. TASKS

TASK 1: RUN BURP SUITE AND CONNECT TO THE TARGET WEB APPLICATION

Using the knowledge obtained from the course, run a web browser and burp suite on your Kali machine and navigate to the web application <http://172.16.160.102>.

TASK 2: PERFORM RECONNAISSANCE ACTIVITIES AGAINST THE WEBSITE

The website contains a note from the developer informing us about the upcoming launch of a new site. Using Burp suite's capabilities, try to identify resources that were hidden from regular users' sight.

Hint: During web application penetration tests, always go through the given site's HTML code.

TASK 3: AUTOMATE ENUMERATING RESOURCES

Use Burp's Intruder to automate searching for available resources.

TASK 4: USE BURP REPEATER TO EXTRACT SENSITIVE INFORMATION

Using Burp Repeater, manually interact with the web application to obtain sensitive information that the developers failed to hide.

Hint: During web application penetration tests, try fuzzing the value of any identified [parameter](#). Responses containing sensitive information may be returned by assigning an unexpected value to a parameter.



SOLUTIONS

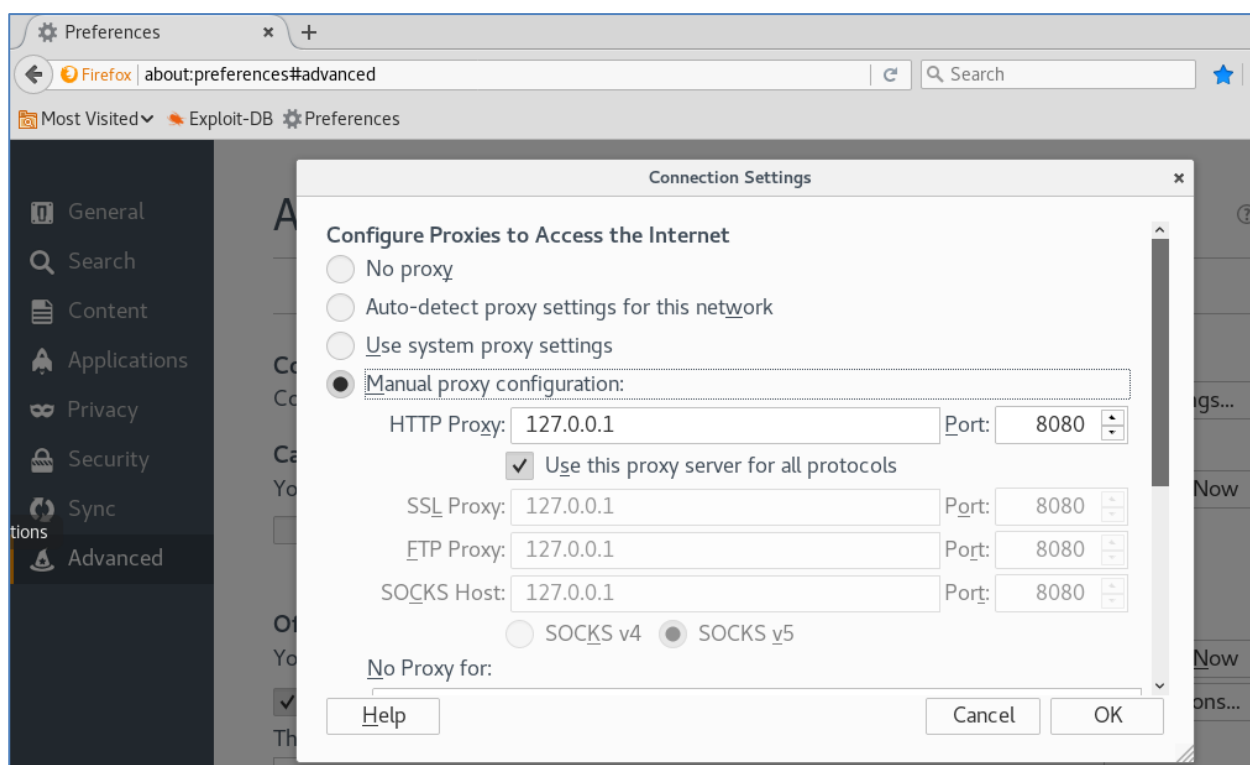


Below, you can find solutions for each task. As a reminder, you can follow your own strategy, which may be different from the one explained in the following lab.

TASK 1: RUN BURP SUITE AND CONNECT TO THE TARGET WEB APPLICATION

First, launch Firefox and Burp Suite and, as presented in the course slides, set Burp as your browser's proxy. The below screenshots will serve as a quick reminder on how to do so:

- Firefox:



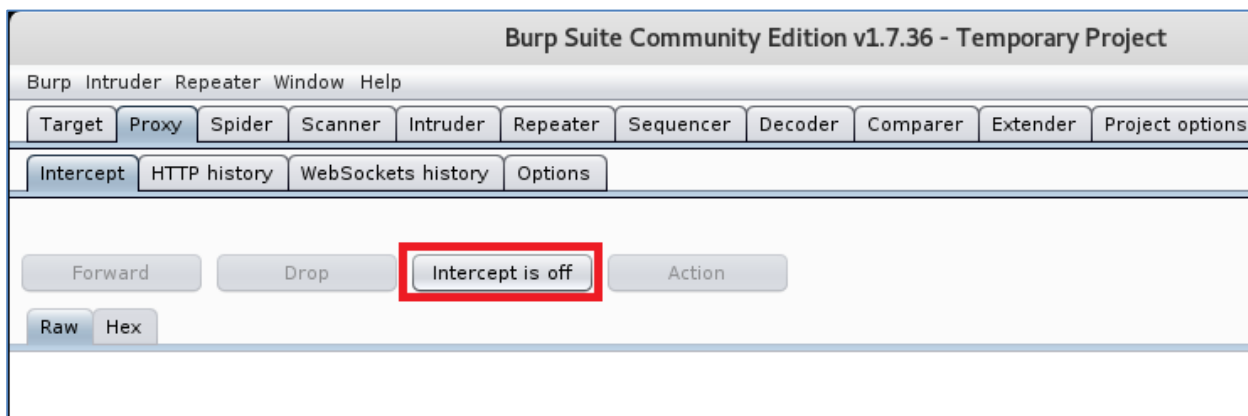
- Burp:

```
root@0x1uk3:~/Desktop# java -jar burpsuite_community_v1.7.36.jar
```

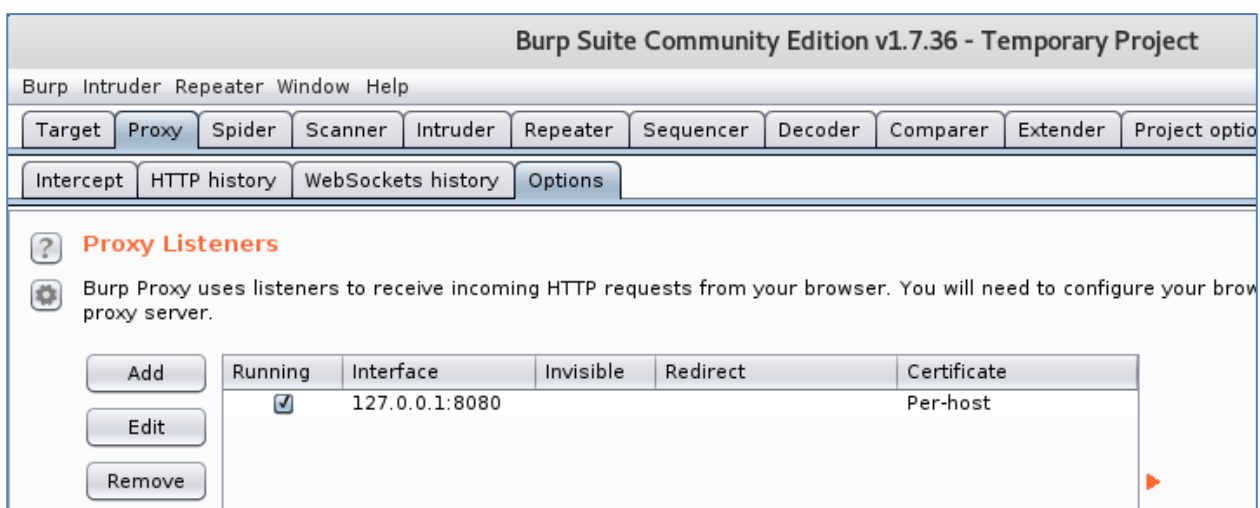
```
java -jar burpsuite_community_v1.7.36.jar
```



Go to the “Proxy” -> “Intercept” tab, and make sure it is not configured to “on”.

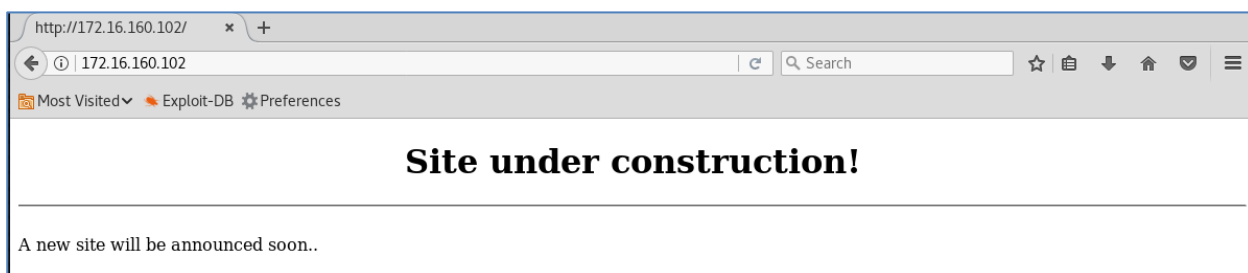


Next, go to the “Proxy” -> “Options” Tab, and check if your proxy is running and active, as per the below screenshot.



Now, use your web browser to navigate to the target web application at <http://172.16.160.102>.

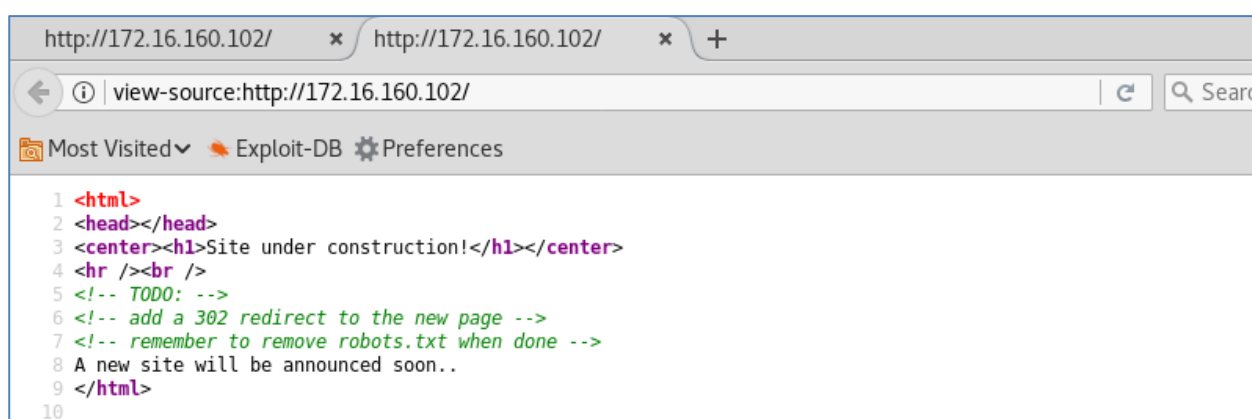
You should be presented with the below site.



TASK 2: PERFORM RECONNAISSANCE ACTIVITIES AGAINST THE WEBSITE

At first glance, there is nothing to do on the website. But, for a hacker, this is just the beginning.

Try to view the web page's source to check if what you see in the browser is the complete content of the website. On Firefox, you can press "Ctrl + U" to browse the page's source in a new window. If you do so, you will be presented with the HTML content of the website, which also contains comments. These comments, which are omitted when the page is rendered, seem to serve as a note from the developers about a file named *robots.txt* that is available.



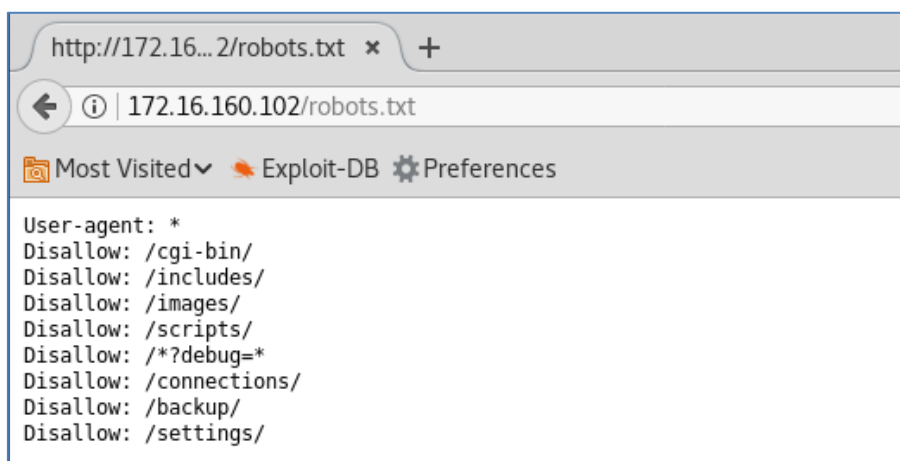
```
1 <html>
2 <head></head>
3 <center><h1>Site under construction!</h1></center>
4 <hr /><br />
5 <!-- TODO: -->
6 <!-- add a 302 redirect to the new page -->
7 <!-- remember to remove robots.txt when done -->
8 A new site will be announced soon..
9 </html>
10
```

A *robots.txt* file is common on web applications. It is an internet-exposed file that contains instructions for automated web browsing tools called “web crawlers”, which scan the whole internet for the various web search engines to present up-to-date search results.

Specifically, this file informs web crawlers about which paths of the applications should not be **indexed** to be included in search engine results. It is common that some sensitive paths can be found in *robots.txt*, as developers think that this will protect sensitive locations across the website from being scanned by search engines. In fact, this approach gives an additional opportunity to hackers, who can inspect *robots.txt* and identify these locations.

Equipped with this knowledge, point your web browser to the *robots.txt* file in order to reveal its content.





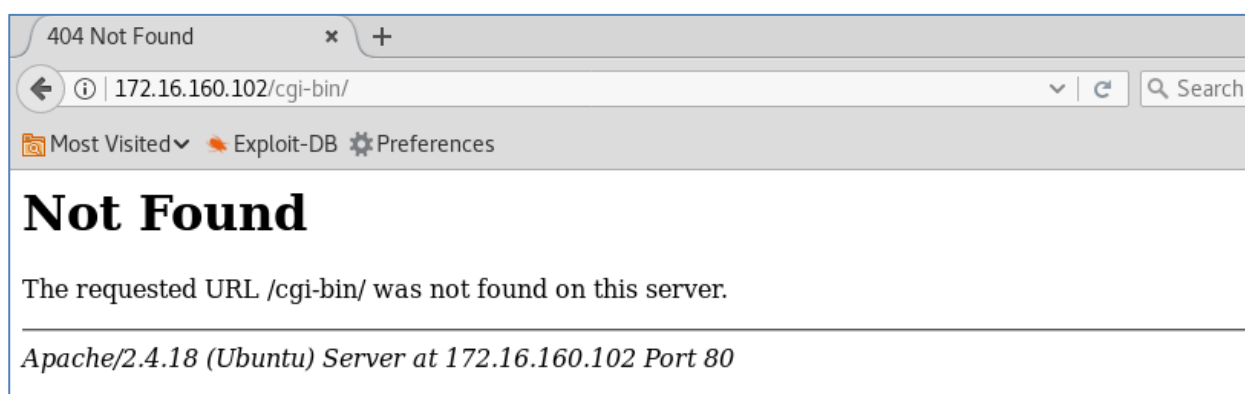
The “Disallow” instruction means that the developer of the site does not want web crawlers to include certain paths of the application in the search results.

Assume that the application path is <http://172.16.160.102> and there is an instruction, like the one below:

- Disallow: **/cgi-bin/**

This means the developer does not want <http://172.16.160.102/cgi-bin/> to be found through a search engine.

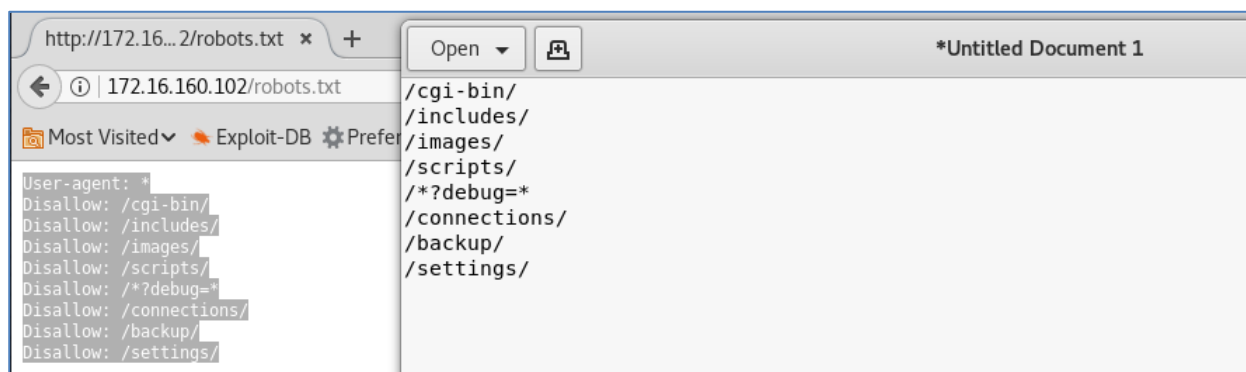
It also means that such a path **may** exist within the application. Pointing the browser to the **/cgi/bin/** path reveals that such a resource does not exist.



There are few more paths remaining in the *robots.txt* file worth inspecting. While you are on a penetration testing engagement, you may encounter a *robots.txt* file containing lots of entries. Inspecting all of them manually can be a tedious process. To automate things, we will use Burp Intruder. Burp Intruder can quickly identify if any of the resources contained in the *robots.txt* file actually exist.



First, group up the included paths **only** and make a list out of them, with one path per line. Do so, as follows:



Next, save the file in a location of your choosing. Remember its name since it will be used soon.

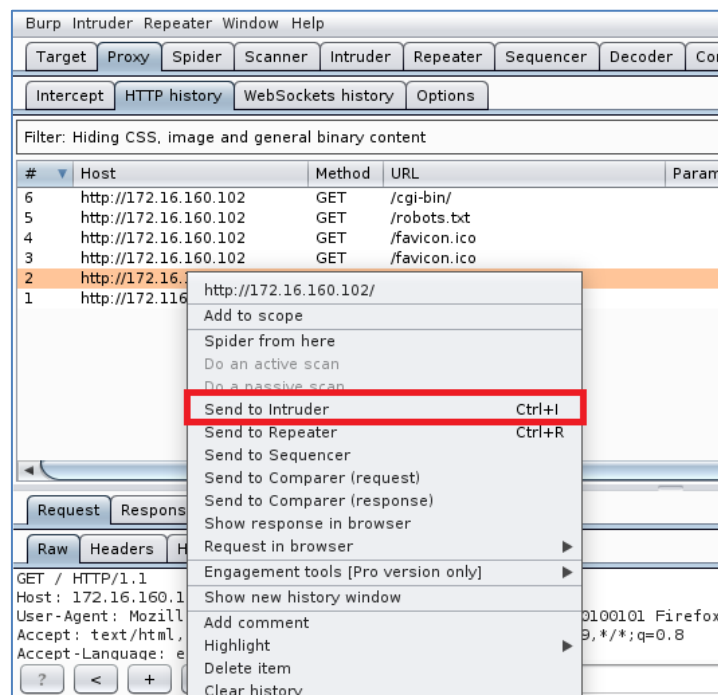
You should have the following list:

```
////////////////////////////////////  
/cgi-bin/  
/includes/  
/images/  
/scripts/  
/*?debug=*  
/connections/  
/backup/  
/settings/  
////////////////////////////////////
```

Now, go to the Burp Suite -> "Proxy" tab and choose the sub-tab "HTTP History".

Right-click any request to the application (preferably one without any path or parameter) and then click on "Send to Intruder".

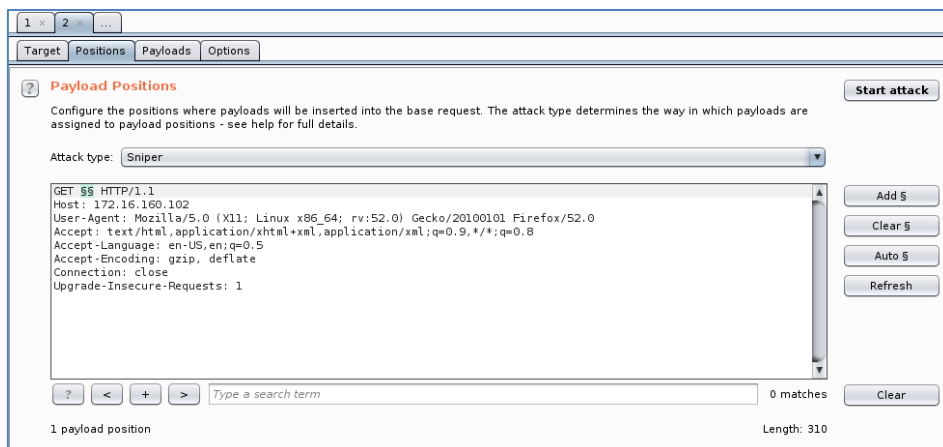




Now, go to the “Intruder” Tab and choose the “Positions” tab. Burp Intruder is a tool that allows you to send numerous similar HTTP requests to the application, with the ability to change a certain part of it each time – this is called **web fuzzing**.

To mark the place where your custom content starts, you need to put a cursor in the HTTP request in the desired place, and then click the “Add §” button. Note that you need to do it two times. The first time “opens” an **insertion point** - the place where your controlled input will start and the second one “closes” it.

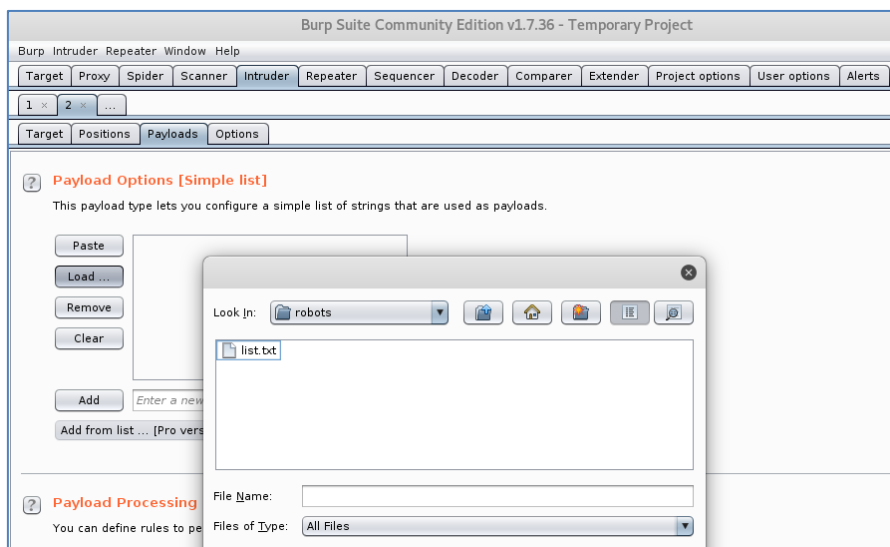
Your data will be inserted between these signs in every HTTP request issued by Burp Intruder. This time, since we are searching for web paths, the **insertion point** will be placed just after the HTTP verb. “/” was removed since the paths already contain it.



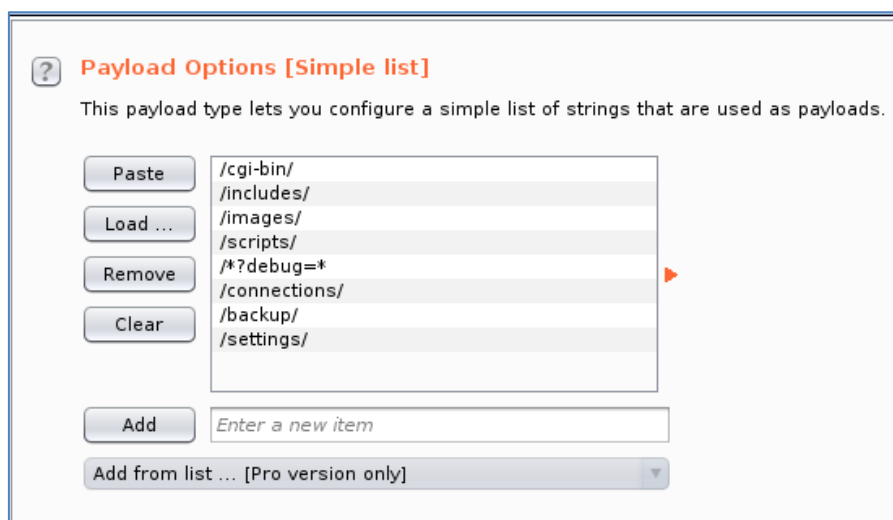
Now, we will make use of the path list file we created and saved previously.

Go to the “Payloads” tab. “Payload” will be the data which will be **inserted** into every request. For every **payload** supplied on the payload list, Burp Intruder will take it, place it into the **insertion point** and issue a request to the web application. This way, it will be possible to issue requests against the server for every path on the list automatically.

To try it, go to the “Payload” tab, click on “Load...” and navigate to the location where you saved the path list file, and then click “Open”.



The paths contained on the list should now appear in the Payload list.



One more thing is needed in order to prevent Intruder from messing with the payloads.

Staying in the same tab, scroll down to the bottom of the page and **uncheck** “URL-encode these characters”.

?

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:

Now, navigate back to the “Positions” tab and in the top right corner click on “Start attack.”

A new window will appear.

Shortly, new requests will be issued to the server, and their result will be visible in the window.

Intruder attack 1						
Attack Save Columns						
Results Target Positions Payloads Options						
Filter: Showing all items						
Request	Payload	Status	Error	Timeout	Length	Comment
0		400	<input type="checkbox"/>	<input type="checkbox"/>	483	
1	/cgi-bin/	404	<input type="checkbox"/>	<input type="checkbox"/>	466	
2	/includes/	404	<input type="checkbox"/>	<input type="checkbox"/>	467	
3	/images/	404	<input type="checkbox"/>	<input type="checkbox"/>	465	
4	/scripts/	404	<input type="checkbox"/>	<input type="checkbox"/>	466	
5	/*?debug=*	404	<input type="checkbox"/>	<input type="checkbox"/>	459	
6	/connections/	200	<input type="checkbox"/>	<input type="checkbox"/>	182	
7	/backup/	404	<input type="checkbox"/>	<input type="checkbox"/>	465	
8	/settings/	404	<input type="checkbox"/>	<input type="checkbox"/>	467	

HTTP Code “200” means that the requested resource exists.

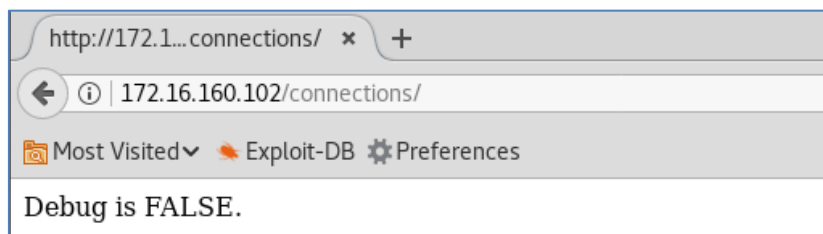
You’ve found a hidden resource, the <http://172.16.160.102/connections/> one.

Navigate to <http://172.16.160.102/connections/> to inspect it.



TASK 4: USE BURP REPEATER TO EXTRACT SENSITIVE INFORMATION

When navigating to <http://172.16.160.102/connections/>, you will be presented with the following screen.



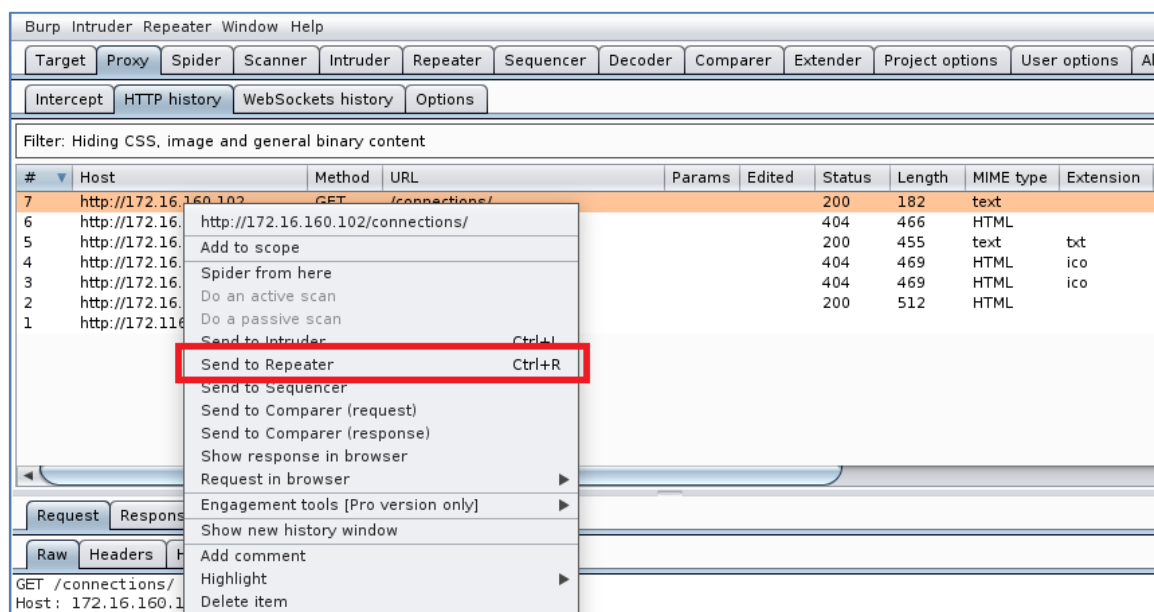
What kind of debug does it mean? Let's go back to the *robots.txt* file. There is one path that differs from others.

- `/*?debug=*`

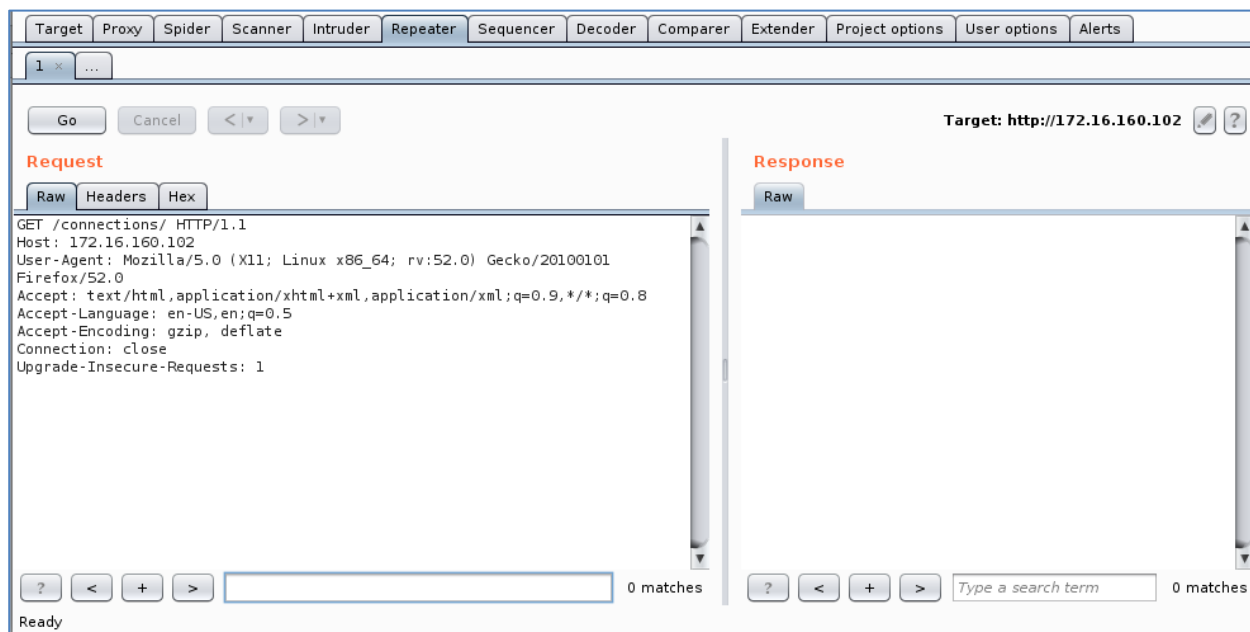
According to the [robots.txt RFC specification](#), such a path means that the web crawler should not index ANY path (* is a **wildcard** – which means anything) that contains the word **?debug=** and ends with ANYTHING (another **wildcard** *)

We will now use Burp Repeater to tamper with the debug parameter in order to reveal sensitive data.

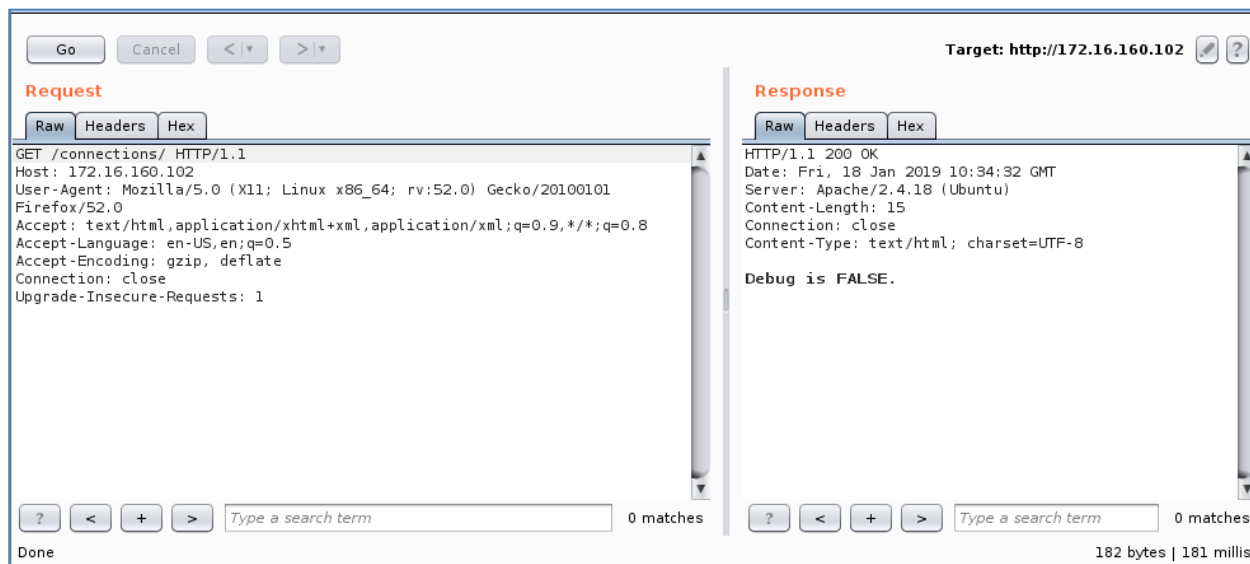
First, navigate to “Proxy” -> “HTTP History” and right click request to `/connections/`. Then, click “Send to repeater” as follows.



Now, go to the “Repeater” tab and find the request in the left window.



The Repeater tool’s purpose is to manually issue requests to the web application and inspect its response. Its advantage over issuing standard web browser requests is that you can tamper with any part of the HTTP request by simply editing it in plaintext, and when ready, pressing the “Go” button to issue it. The response from the server will be shown in the right window when received in plain text.



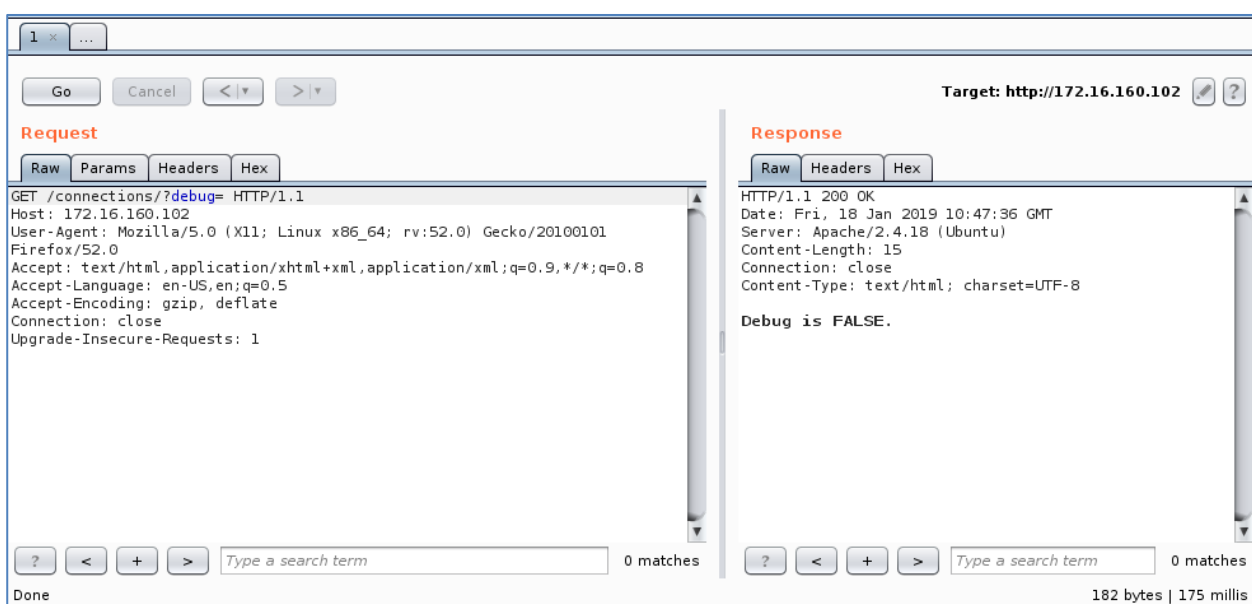
Now, using the information we obtained about the “unwanted” path, we will tamper the web with the request in Repeater’s left window. First, remove the **wildcards**. A wildcard sign (*) can mean any or no path. So, let’s take

- `/*?debug=*`

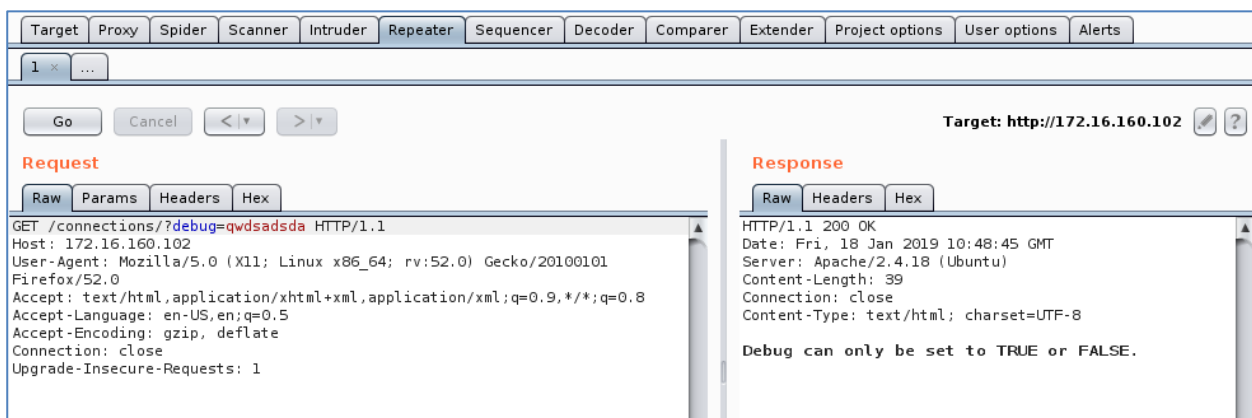
and change it to

- `/?debug=`

Let’s append the above path to the web request that is currently in the left window of Repeater and press “Go”, as follows.

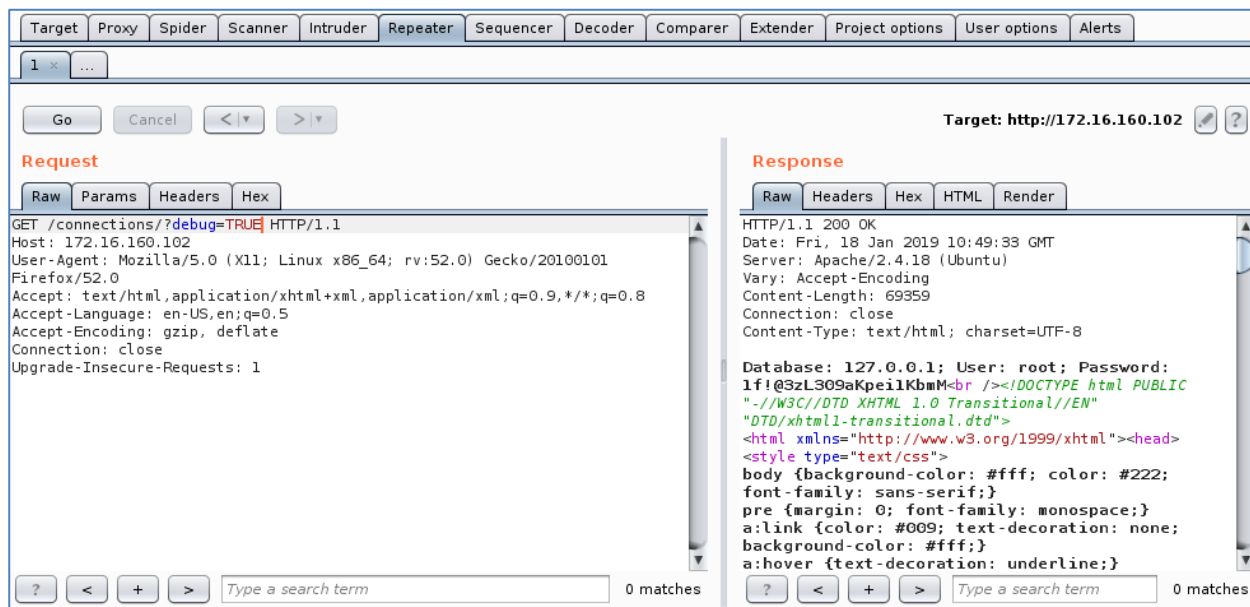


Same result. Let’s try changing what “debug” equals to and press “Go”, as follows.

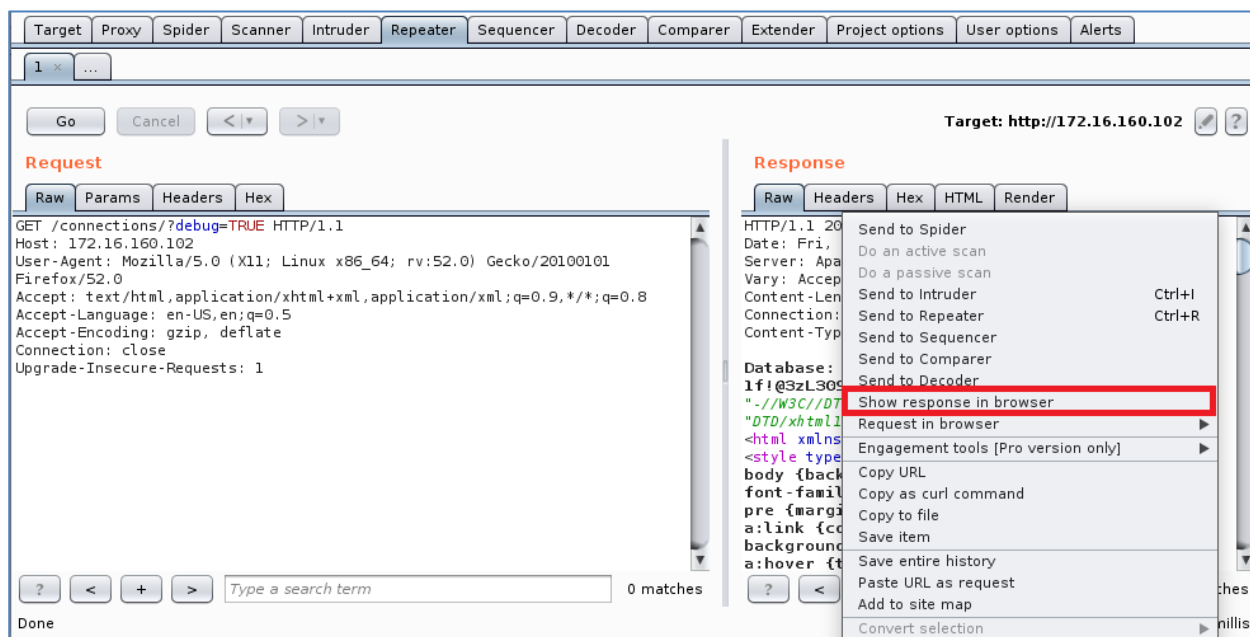


It seems that appending an “equal to” value leads to an unexpected result. The application expects that the **debug** parameter has a value set to either TRUE or FALSE.

Using Burp Repeater, let’s tamper with the request once again by setting the **debug** parameter’s value to TRUE.

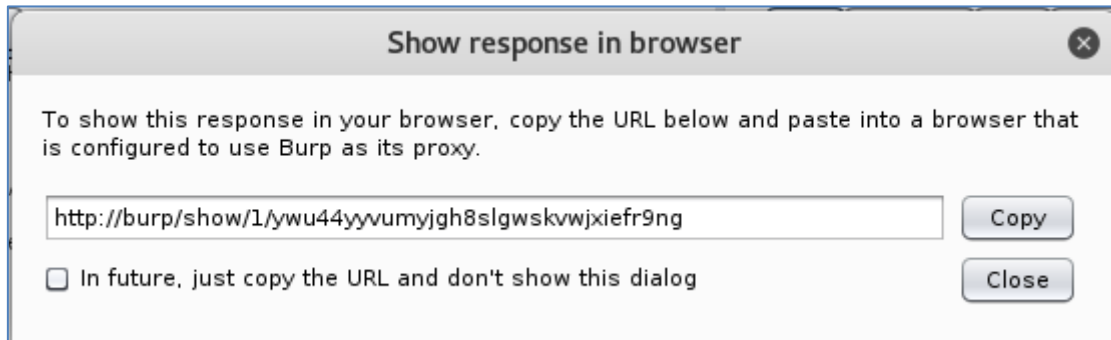


It seems that some sensitive information was leaked, but this can be more visible in a web browser. Let’s right-click on any place within the response window and choose “Show response in browser”, as seen below.



A new pop-up window appears. Copy the URL and paste it into your browser's address bar.

Note that your URL will be different from the one presented below.



Press enter to navigate to the pasted URL.

It seems that you found sensitive information which is exposed to any unauthenticated user on the internet.

Nice work!

