iGNITE Technologies

BURP SUITE FOR PENTESTER

TURBO INTRUDER

READY FOR FIGHT

WWW.HACKINGARTICLES.IN

Contents

Introduction to Turbo Intruder

What is Turbo Intruder

Turbo Intruder is one of the greatest burp suite extensions scripted by "James Kettle" to send a large number of HTTP requests and analyze the results. However, the functionality of this extension is as similar as that of Burp's Intruder carries. Yes, it is fuzzier. But as it uses the HTTP-stack thereby some features make it a bit different and faster

- High speed with least latency during fuzzing
- Low memory usage with a million payloads
- Customizable python scripts for different attack scenarios
- Reliable for Multi-day attacks

Burp Intruder v/s Turbo Intruder

No doubt, the intruder tab is the most powerful part of the burp suite. Whether it's about to fuzz an application over at a single injection point or multiple, it does work seamlessly. However, this tool provides whatever we wish to, whether it's about payloads or the error detections, it is good-to-go. But when it comes to the fuzzing speed or the memory usage it drops it out. If the dictionary exceeds about a lakh payload, the latency and the memory consumption will be at their peak. But why does this all happen?

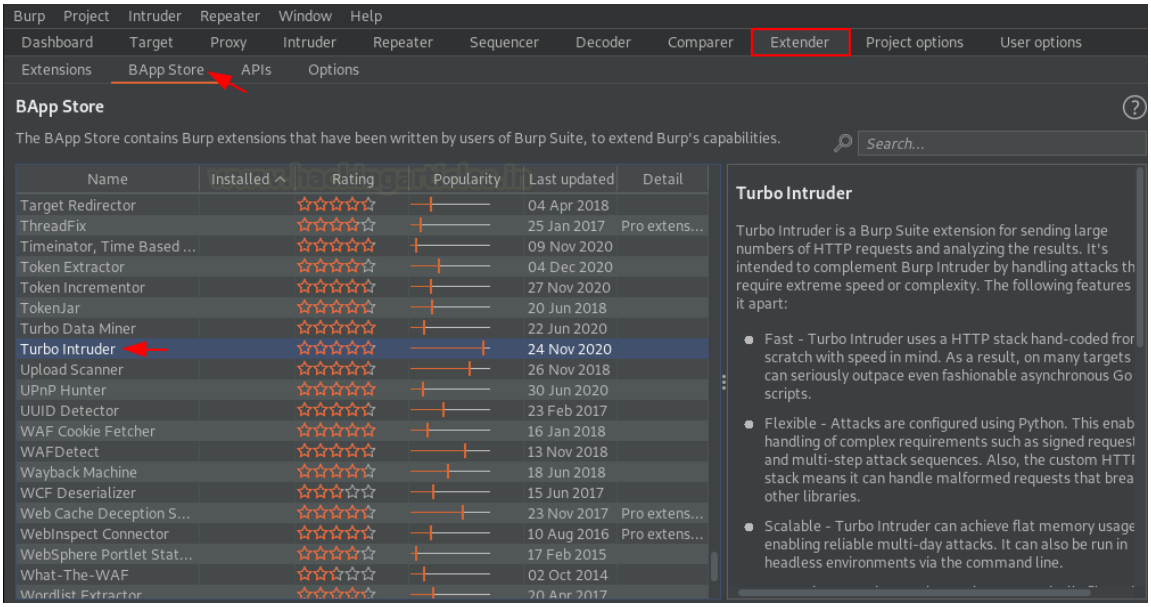
To analyze this, we need to understand the mechanism it carries while fuzzing.

All the major brute forcers, create a TCP handshake for a single request and send the request to the server, thereby the server waits and shares the response back, as soon as the tool gets the response, it then reads out and the same happens again. However, the handshake consumes a lot of time and the sending & reading phases contain a much latency too, thereby giving a high load on the CPU and consuming a lot of memory reducing the speed to fuzz that.

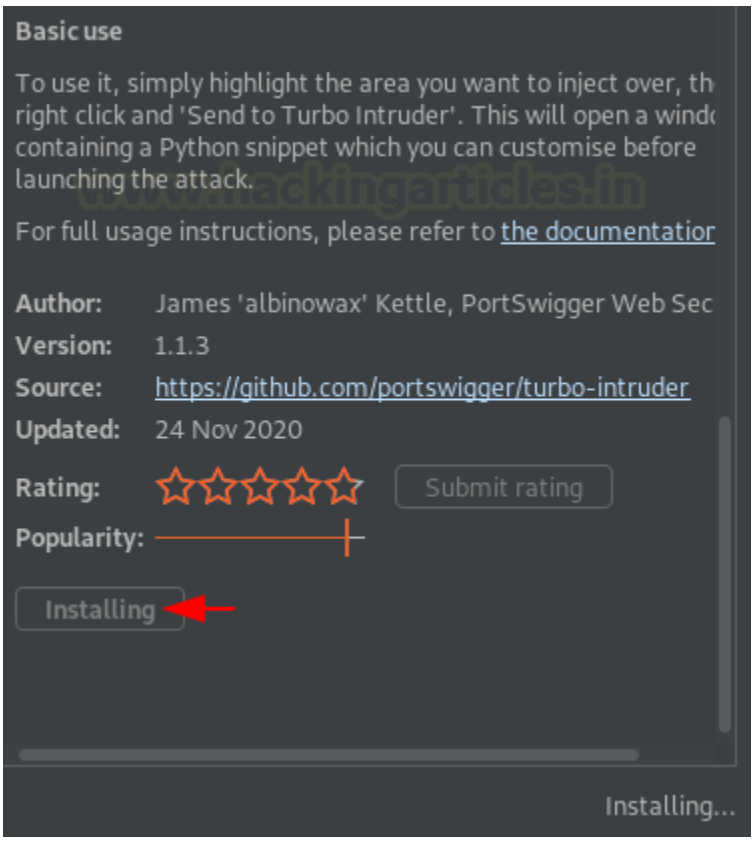
But the turbo intruder is different and as its speed is. It works on an old technology i.e., HTTP pipelining which establishes the connection first and shares as many possible requests in one go without worrying about the received responses to minimize the latency and server processing time.

Turbo Intruder's Installation

It's not difficult to find this plugin nor to install it, simply navigate to the **Extender tab** and then further select the **app Store** option within it and once you scroll your mouse down, you'll find it right in front with a rating reaching almost **5 stars.**

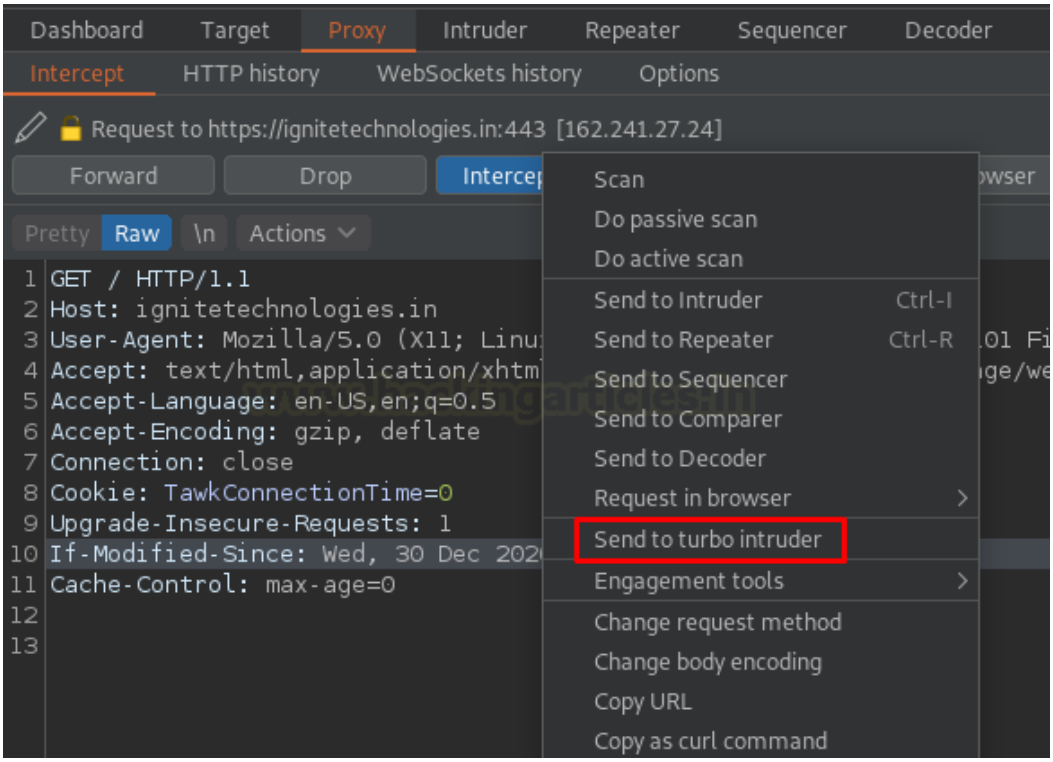


Now simply **hit the install button** over in the description and there we're ready to go.



But where it is, it's not aligned in any of the tabs?

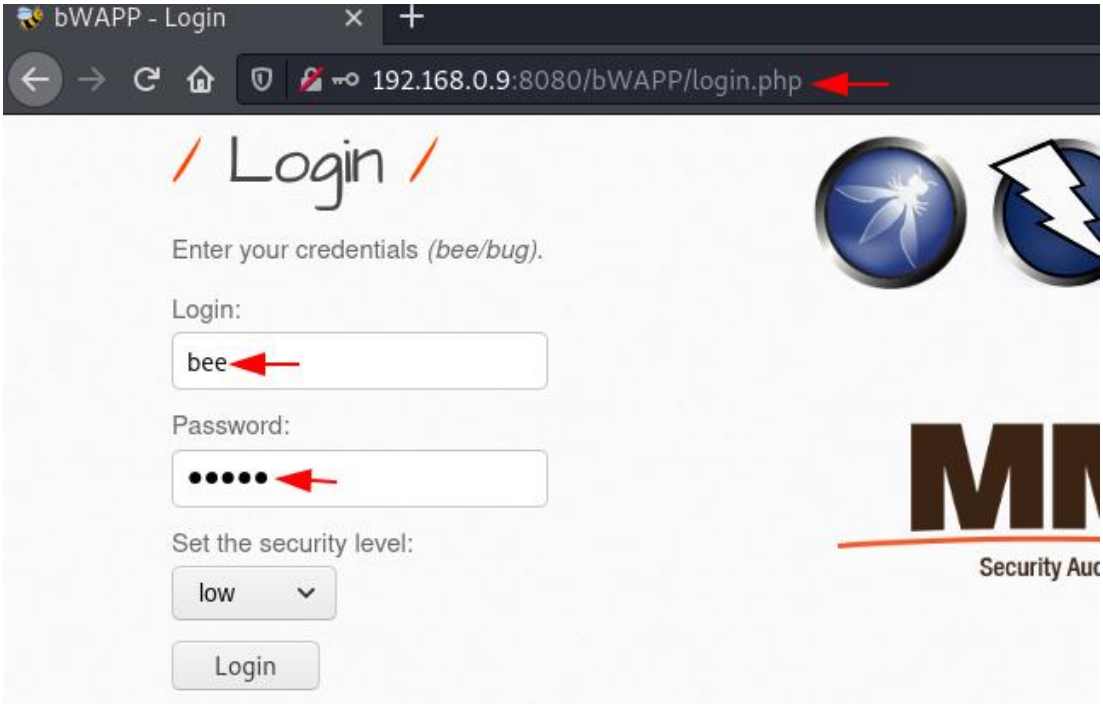
Let's check out within the **Action section,** a simple right-click can give us the **Send to Turbo Intruder** option.



Brute Forcing the application's Passwords

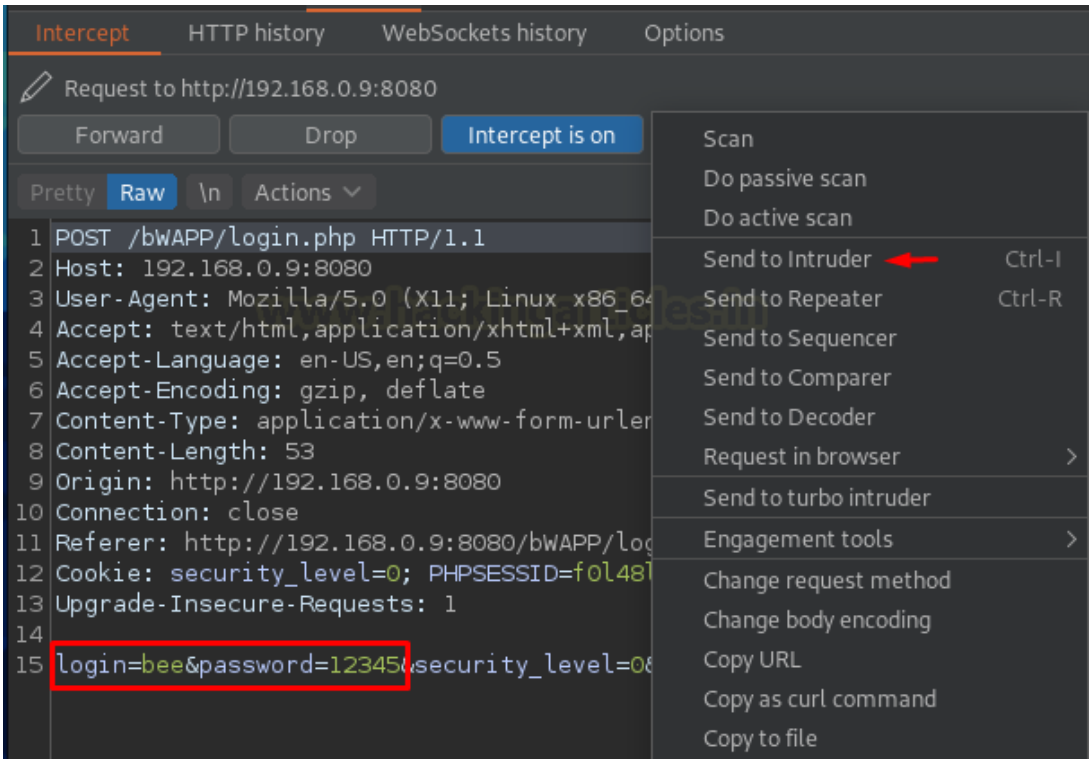
You might be wondering like, all of these concepts are documented either at the portswigger's web page or the extension's GitHub repository, so what's new. And what about the practical exposure, how we could confirm that, yes turbo intruder fuzz the application in a much faster way than the basic burp's intruder does.

Thereby to confirm and check that all, let's tune into our vulnerable application bWAPP and we'll capture the login portal's HTTP Request by setting the username to **bee** and a random password as **12345.**

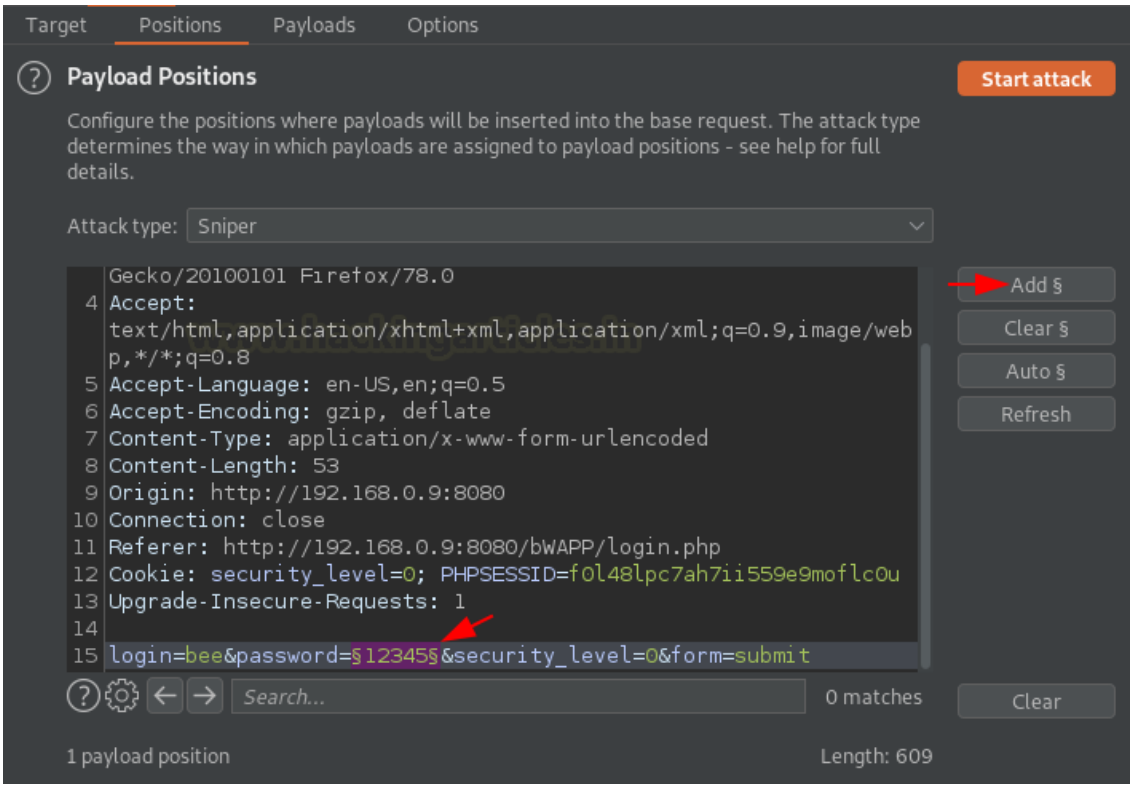


Via Burp's Intruder

Once the **Login** button got fired up, we got the Request intercepted at our Burp monitor, so let's first share it with **"Intruder".**

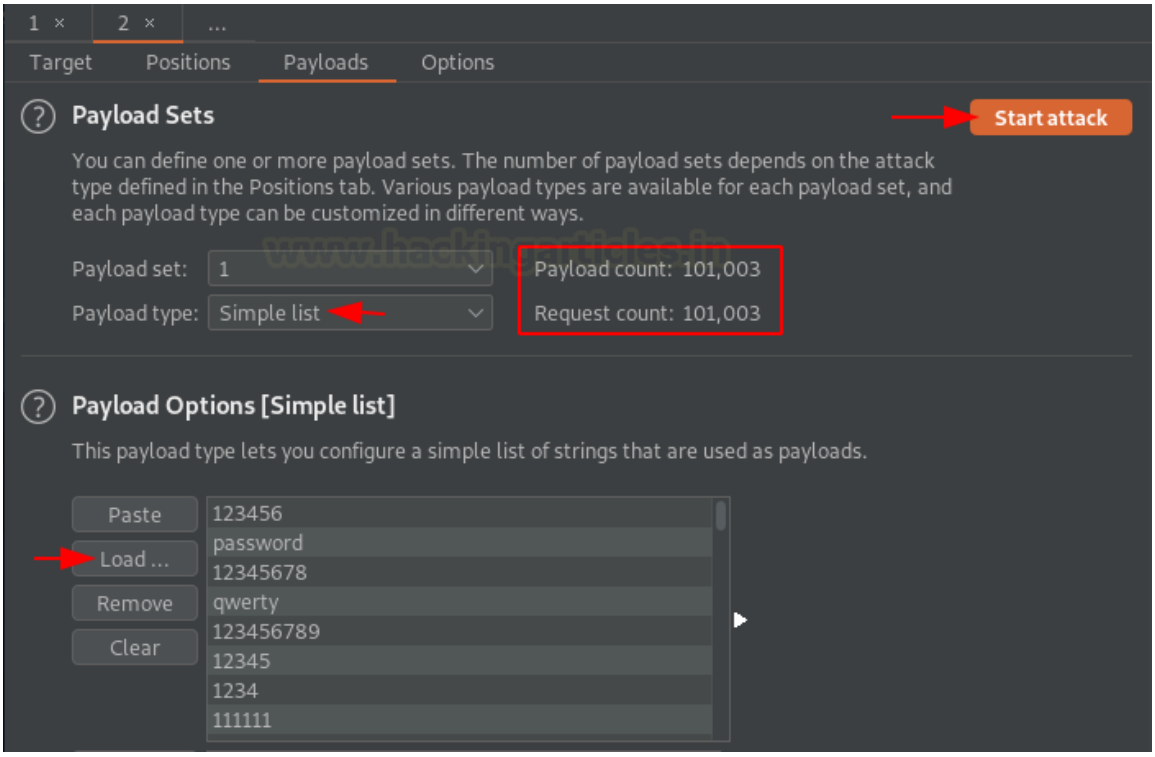


However, the Intruder steps are on our tips, so let's recall them once again. As soon as the Intruder receives the request, our first work is to set the clear out the things and set the **injection points** over at the **position tab**.

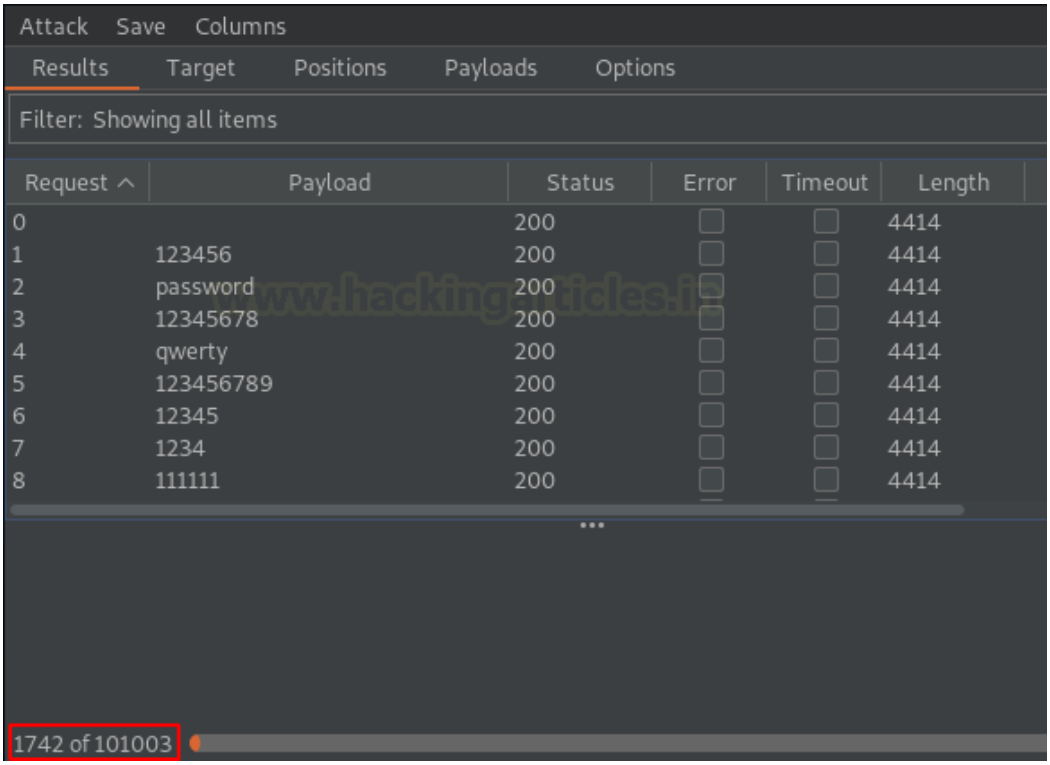


Now, time to inject our fuzzing lists. Switch to the Payloads tab, right next to Position one, and click the Load button to select the desired list. For this section, we've modified the 10-million-password-list of Daniel Miessler SecLists Github's repository and have injected bee & bug within it.

From the below image, you can see that the number of requests is more than 100,000, which is not large but still it could help us to determine the speed.



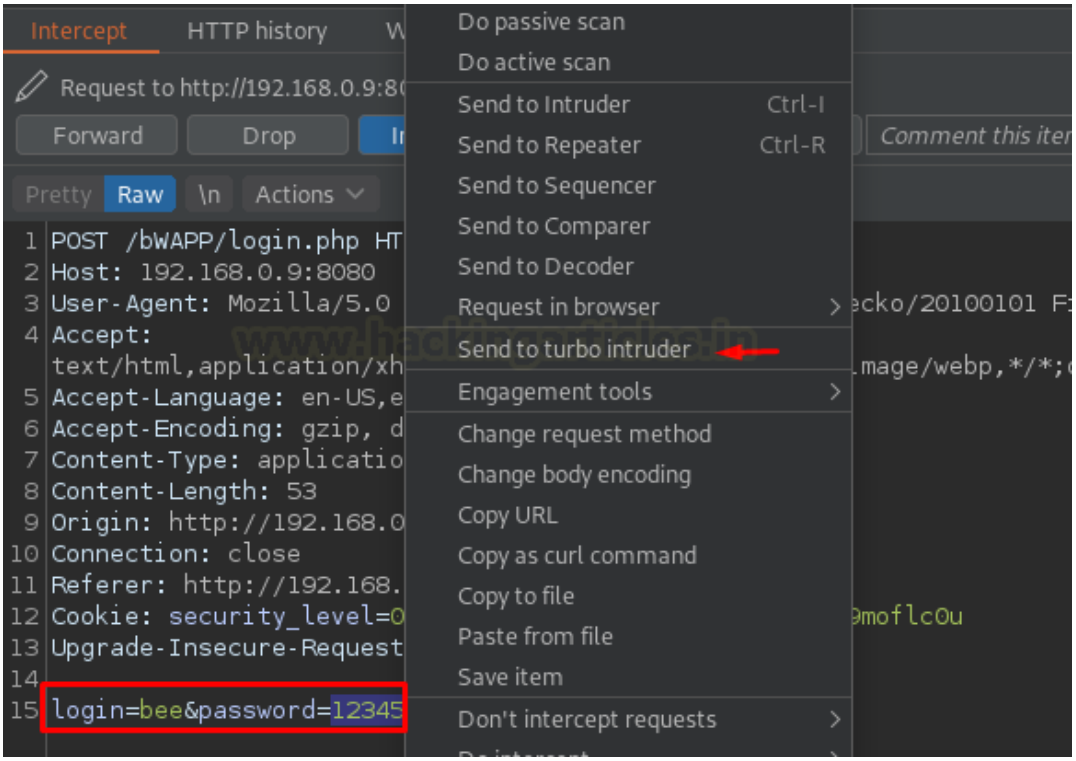
Now as soon as we hit the Attack button, the fuzzer starts up and it took about **60+ seconds** to fuzz about **1700 requests** which makes it about **23 requests per second (RPS).**



Via Turbo Intruder

Let's first **pause** the intruder here and will check what Turbo Intruder dumps out as its speed.

Back into the intercept tab, we'll **select the payload position** and will then hit right-click in order to share the request to the Turbo Intruder.

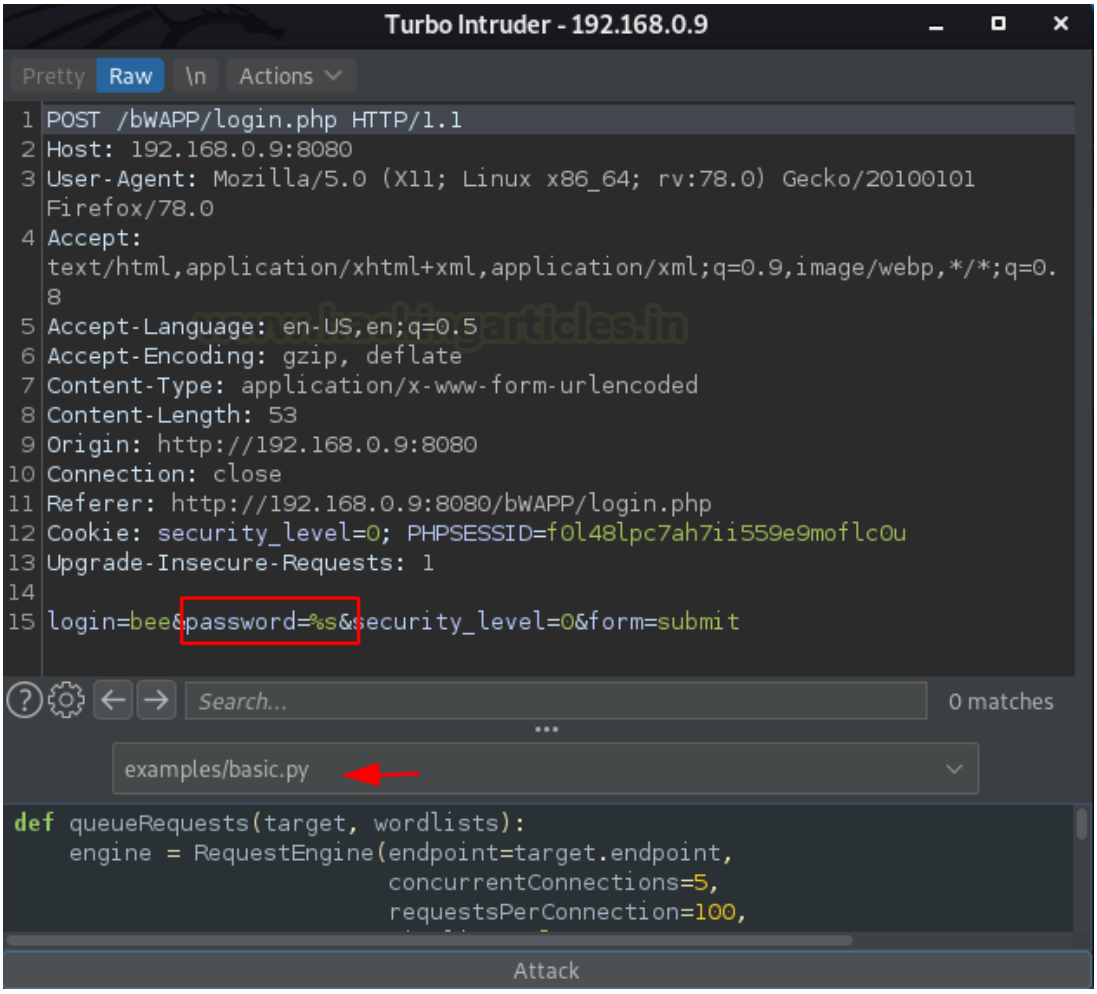


As soon as the **"Send to turbo intruder"** option got fired up, we got a new window popped in front of us. Let's explore what it contains.

The window was segregated into two sections the upper part where our **"shared request"** is embedded into and just below that in the other part we got a **"snippet of python code"** aligned.

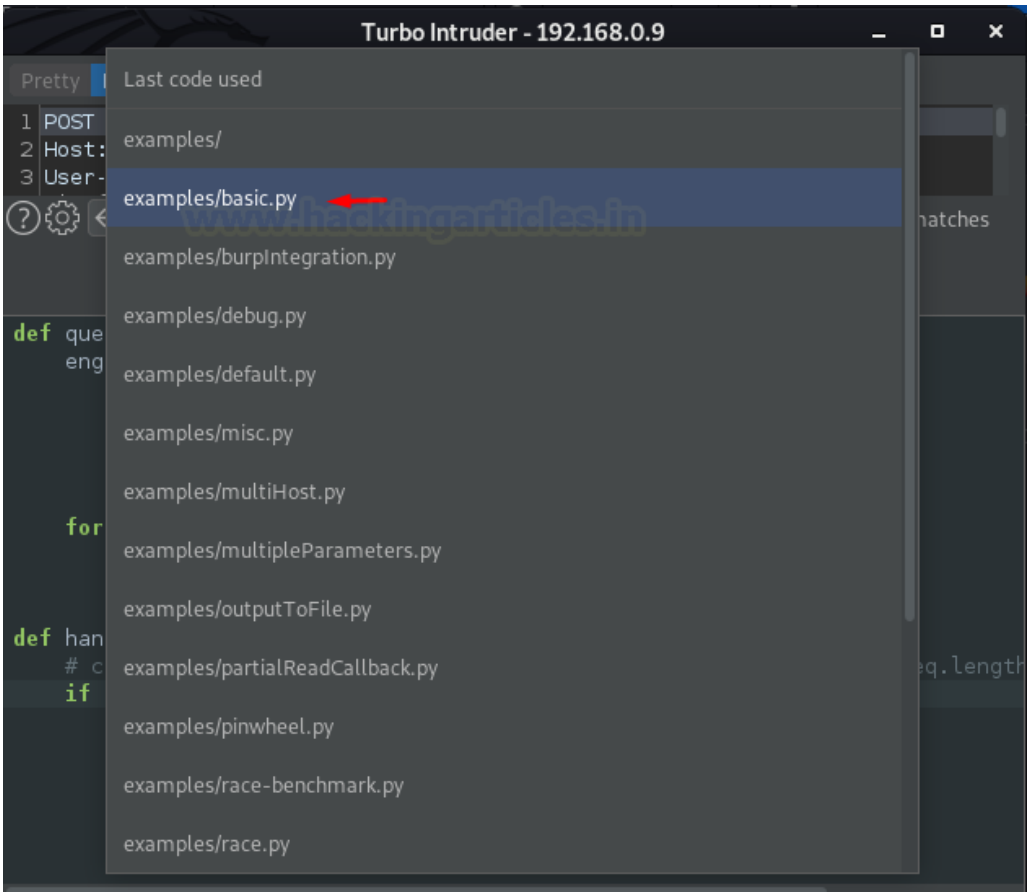
However, over into the **Request section**, we can see that our injection point was converted into **"%s",** representing where the payload will be going to hit.

And at the other section, there is a **drop-down list** with a **python code** displayed, which thus could be modified as per the attack scenario.



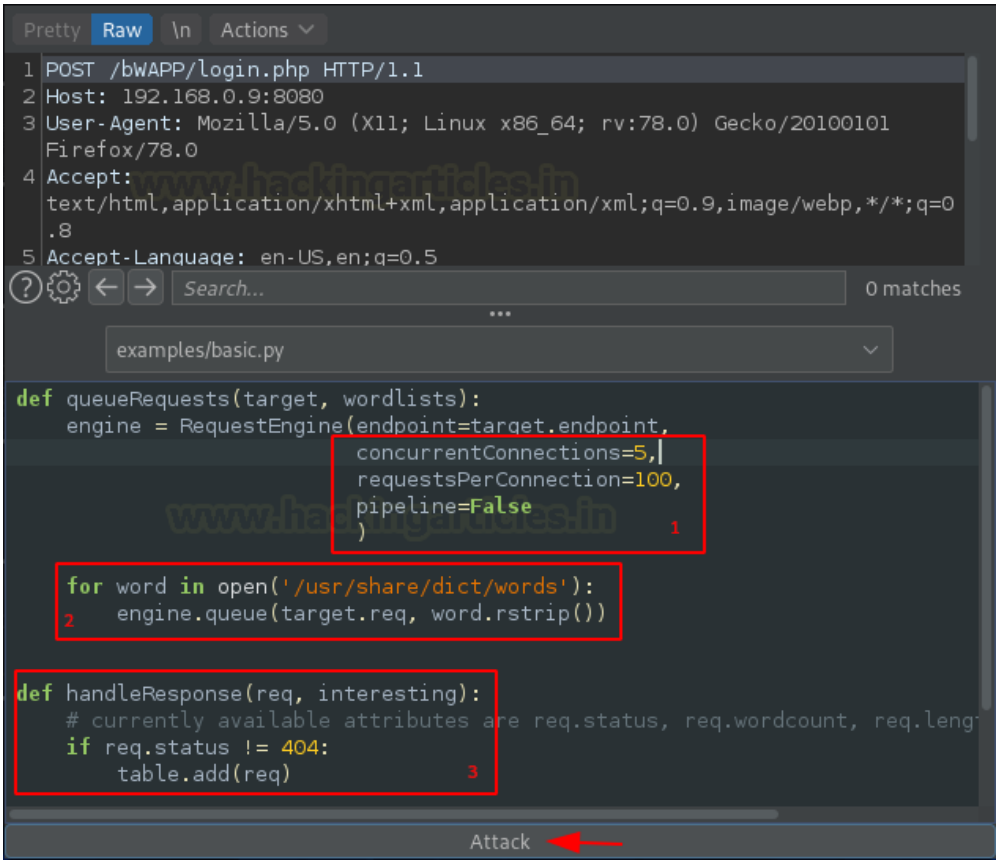
The drop-down list contains several attacking python scripts, that we could use accordingly whenever we need.

So, for the time being, let's use the most common script of the Turbo Intruder i.e. **examples/basic.py**

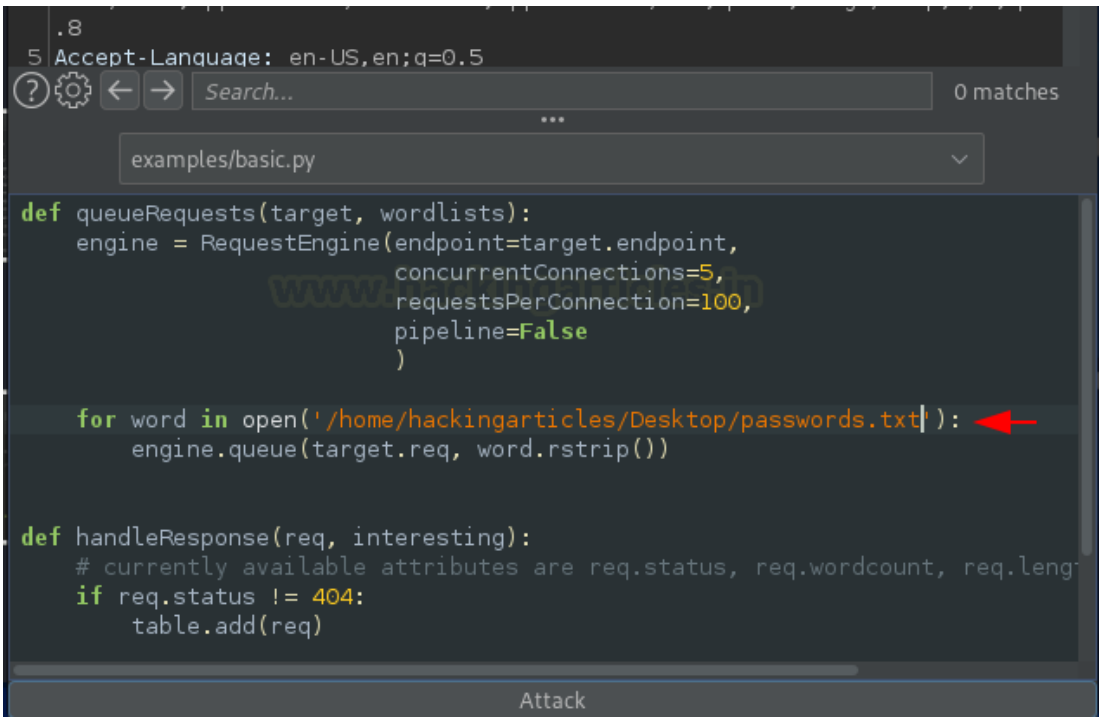


Once we select that, the python code within it got displayed on the screen. Let's see what it says –

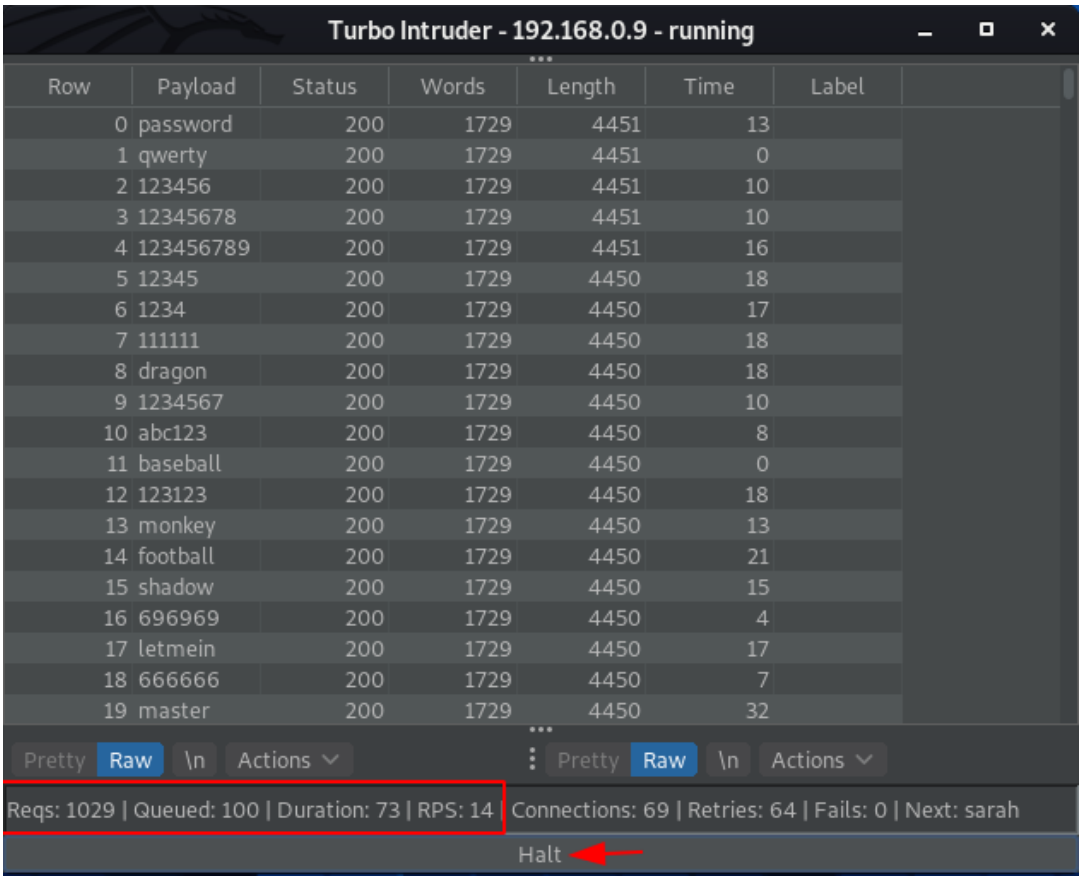
1. The script within the **first highlighted box** is responsible for the **fuzzing speed** and the **number of connections made** by the turbo intruder. However, it even carries up the **requests made per connection.**
2. Over in the **second box** we need to **add the dictionary** manually by specifying its location.
3. The **third code snippet** is the most highlighted section as it defines **which request should be displayed** in the output table. Here the code **"if req. status != 404:"** defines that all the requests will get added to the table leaving the once that are having **status code = 404**



Now, let's **inject our dictionary** by defining its path while manipulating the code.



Further, we'll hit the **Attack button** to initiate our fuzzer. As we do so, within **73 seconds** it's just **1029 requests** shared, creating the **RPS** (**R**equest **p**er **S**econd) count to **14.**

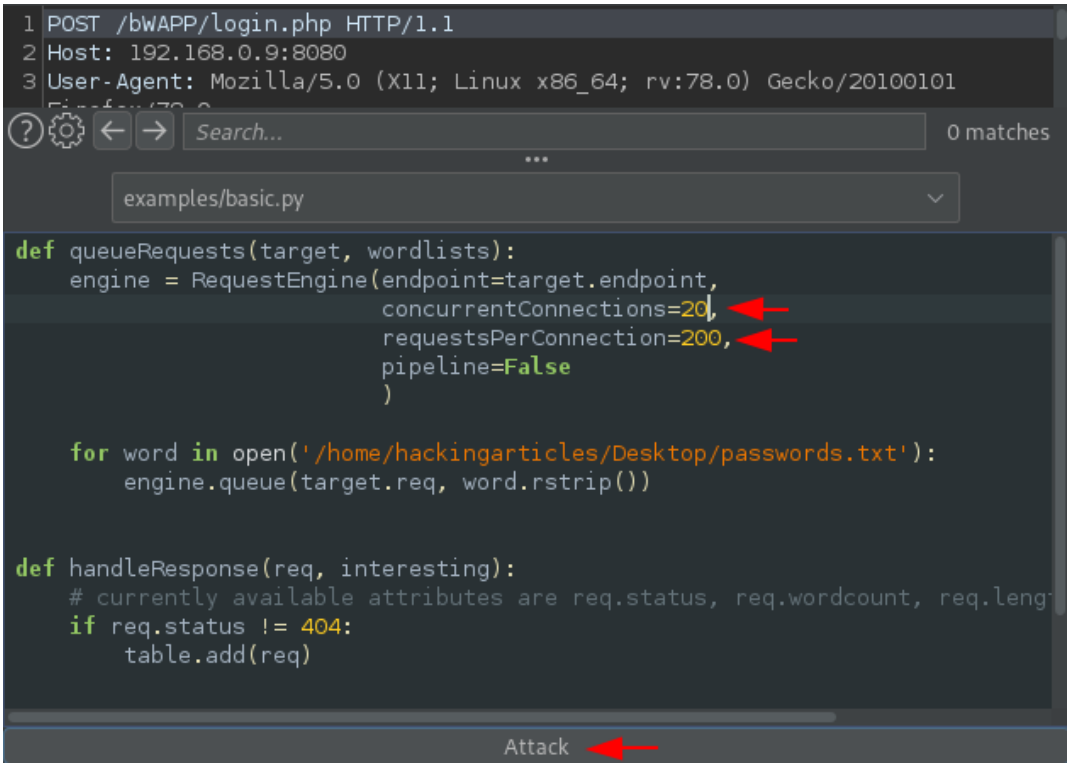


You might be wondering, this needs to be faster than the basic intruder, however, the RPS rate for the Burp Intruder was 23, and it's just 14.

This is all due to the default configuration in the python script as the **concurrent connections** were just **5** and the **Request per Connection** was **100**. Let's modify it all and start the attack again, to do so hit the **halt** button and configure the same with

concurrentConnections=20,

requestperConnetction=200,

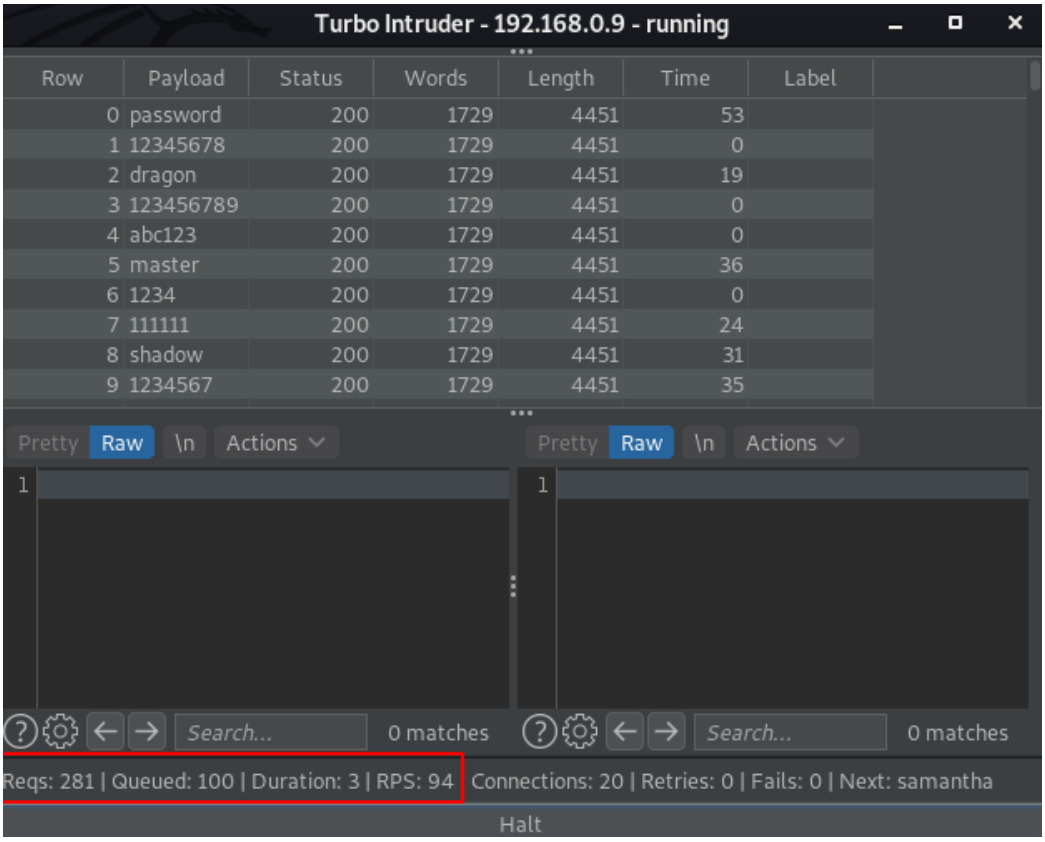


And as the attack executes up, the **RPS rate jumps directly to 94** and within just **3 seconds** the fuzzer hits about **280 requests**.

Thereby for this attack, we can say it's about 4 times faster than the Basic Intruder.

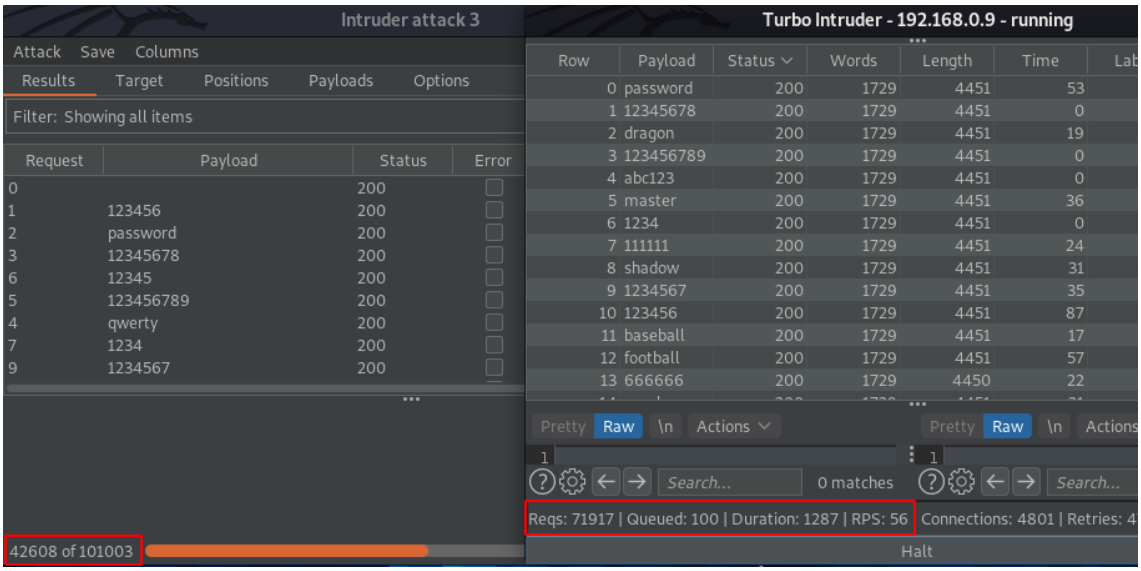
However, during the attack, you could find that the **retries value is going up**, as the fuzzer was initiating. So, is the requests are not hitting?

No, it is an indication that the connection per request value might be too high for the server to handle, making the attack is less optimal, thus we need to cut it down and reduce it to half.



Do you remember, we halted the Intruder's fuzzing? Let's resume it and restart our attack over the turbo intruder, and we'll then wait for a few minutes to analyze the output.

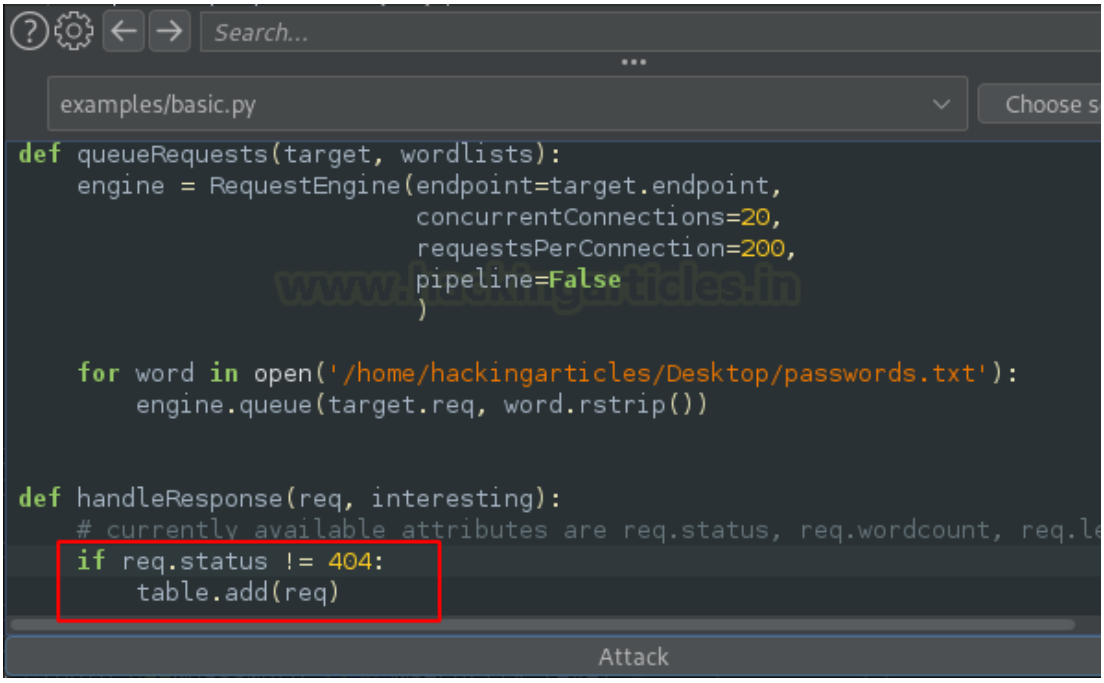
From the below image, we can see that **Turbo Intruder is ahead with 3000+ hits**, displaying the speed it carries within.



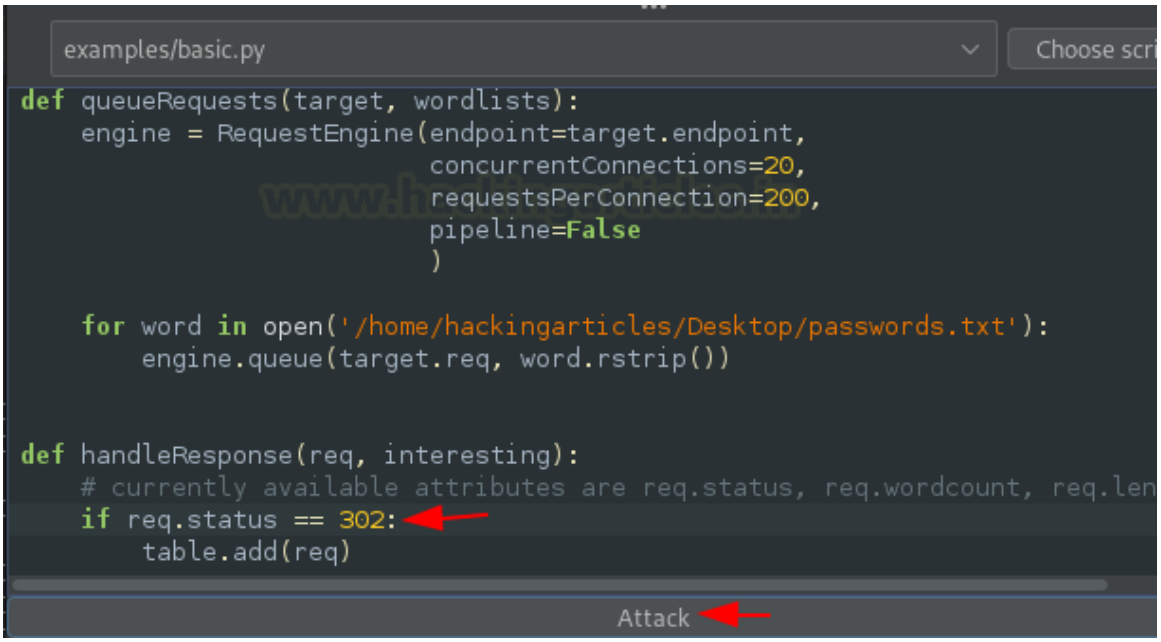
Customizing the Python Scripts

Over in the above section, we've discussed that we can manipulate the python script as we wish so. However, manipulating it doesn't require any advanced scripting skills, it's just that we know what the code wants to say. So, for example –

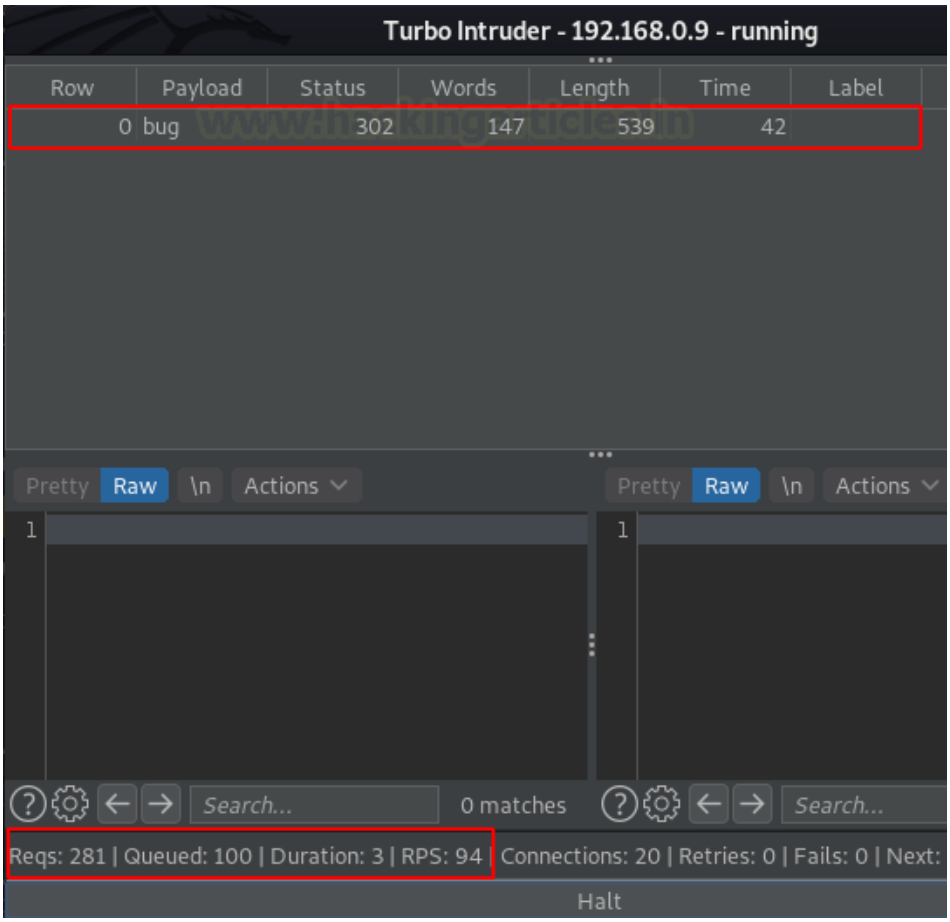
We want the table to dump only the ***302 Redirection***, we don't want any ***200 OK***, nor ***404 Error***, thereby in order to do so, we just need to manipulate the 3rd code snippet.



Now over here, we'll change **not-equal to (!=)** with **equal-equal (==)** and will modify 404 with 302, Such that the output will only have the redirection requests listed.



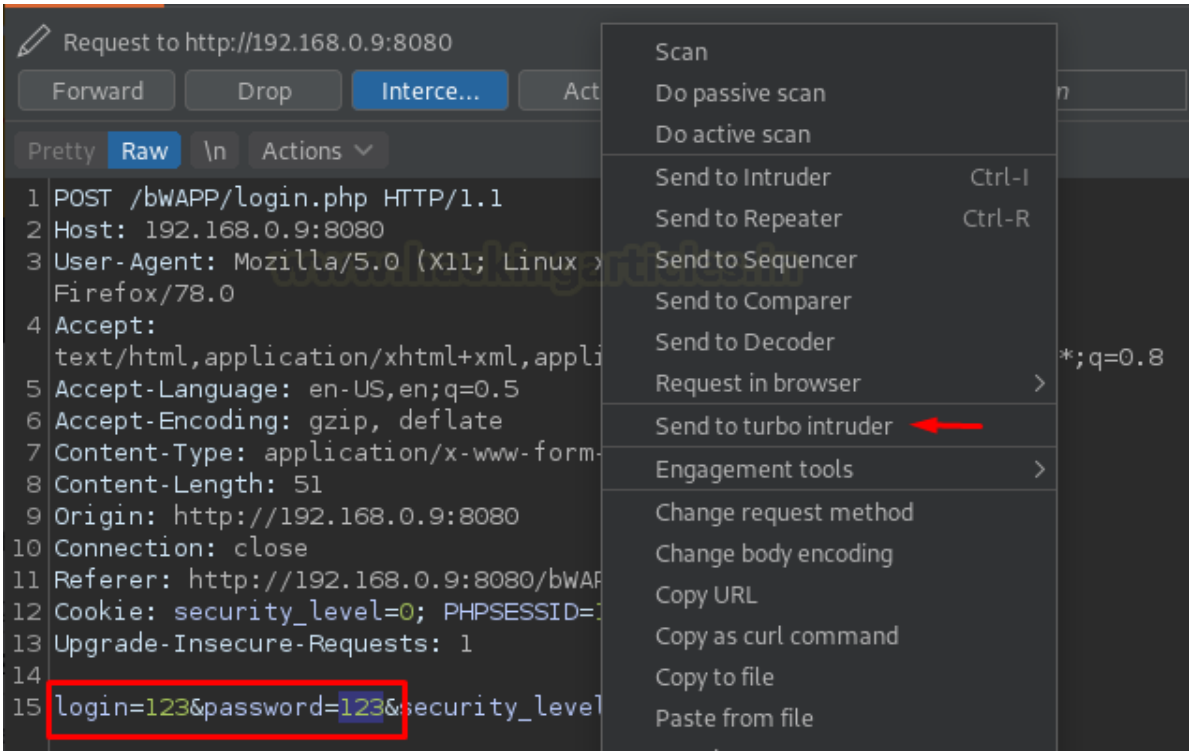
From the below image, we can see that within 3 seconds we got our output listed over at the table segmented.



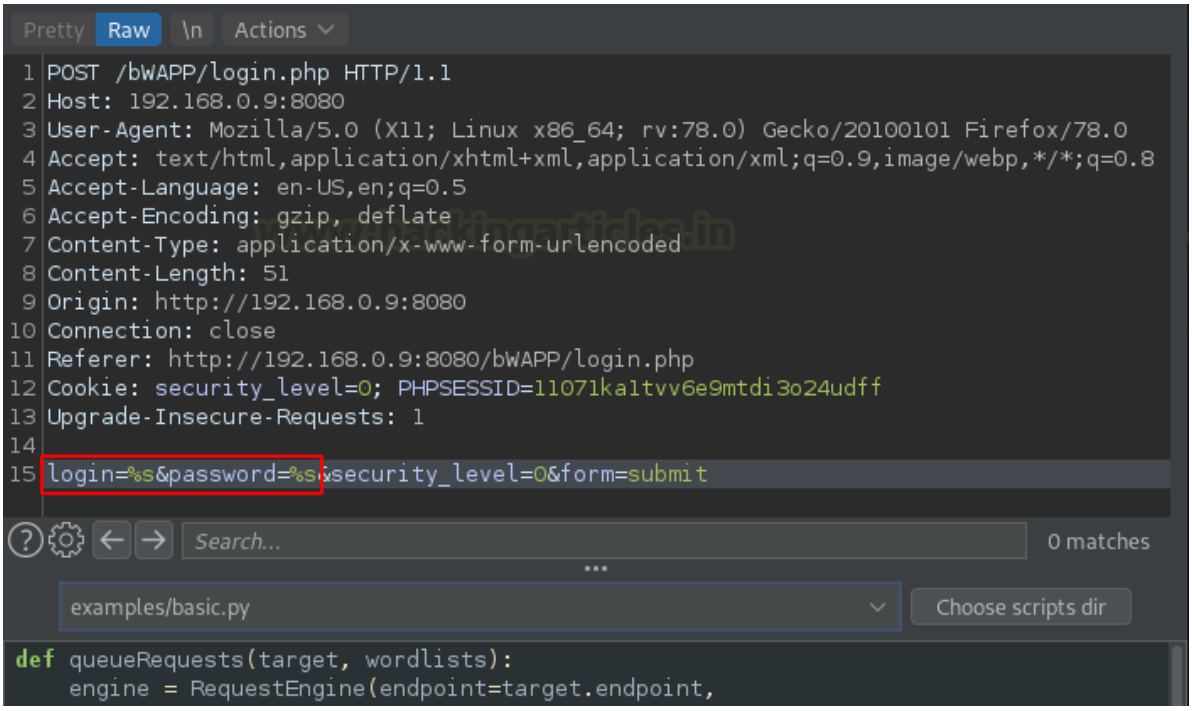
Fuzz for Multiple Parameters

You might have used Cluster Bomb over in the basic intruder which helps us to fuzz the application with multiple parameters. However, in the same way, we're having a python script listed in the drop-down menu too which will thereby provide us with the option to fuzz as similar to the cluster bomb payload type. Let's check that out.

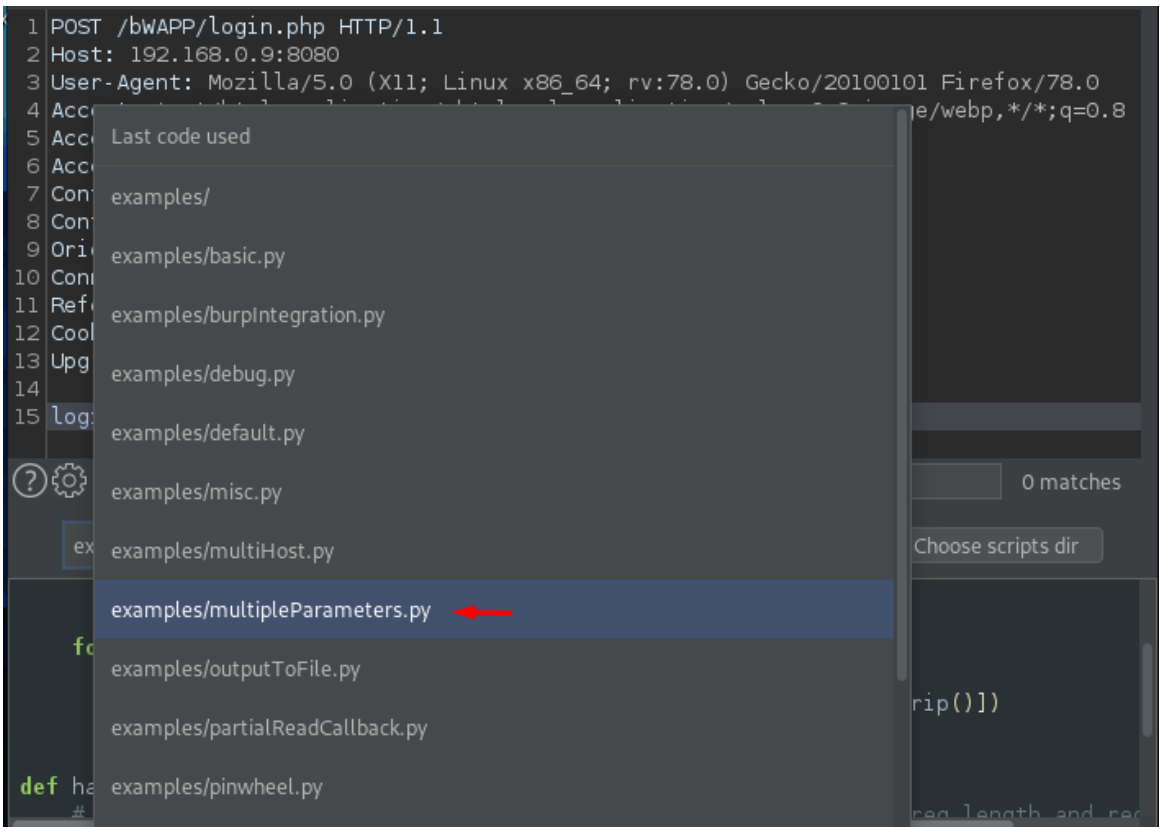
Back into the **Proxy tab** let's select any parameter and hit a right-click in order to navigate to **turbo intruder**



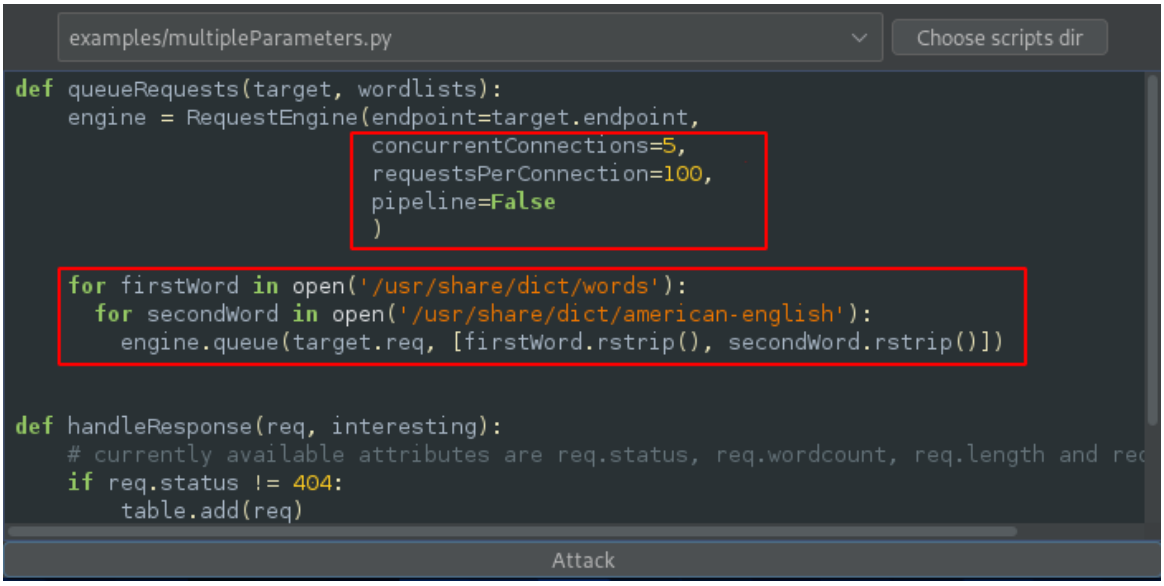
There, over in the request portion, let's set **"%s"** in order to select the injection points.



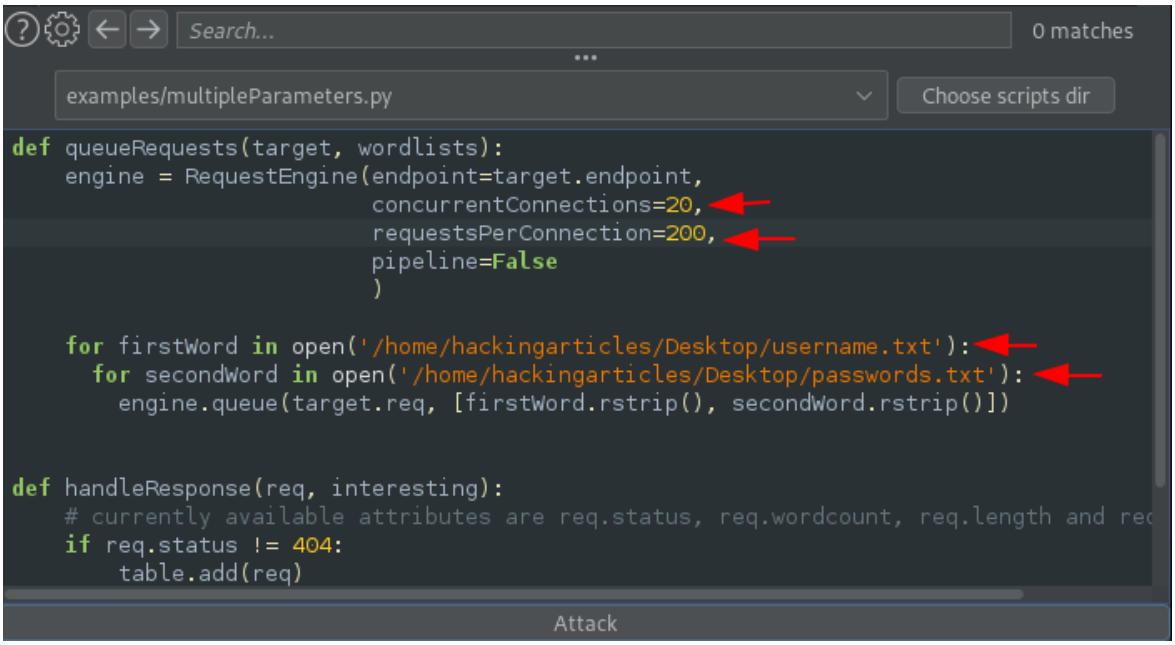
Time to choose the script, click on the bar to check the drop-down menu, and then select **examples/multipleParameters.py**



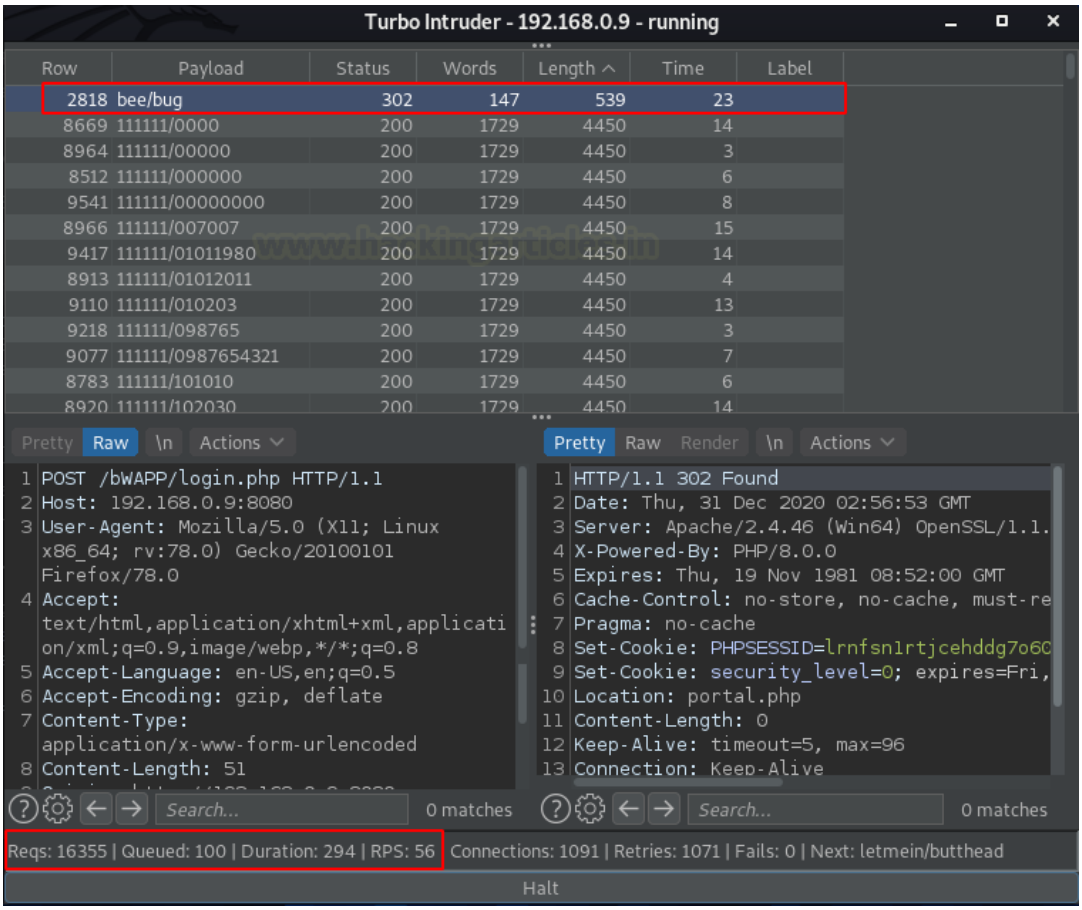
Once clicked-on, the script is there for us to get manipulated.



Let's boot the speed by manipulating
the **concurrentConnections=20** and **requestperConnetction=200,** further we'll set the dictionaries for the first word and the second word.



And there we go, as soon as we hit the **Attack** button the screen got shuffled and we got the output table in-front of us. Within a few minutes, as we double-clicked the Length section, we got the **302 Redirection** with **bee/bug.**


Turbo Intruder - 192.168.0.9 - running
Row Payload Status Words Length Time Label
2818 bee/bug 302 147 539 23
8669 111111/0000 200 1729 4450 14
8964 111111/00000 200 1729 4450 3
8512 111111/000000 200 1729 4450 6
9541 111111/00000000 200 1729 4450 8
8966 111111/007007 200 1729 4450 15
9417 111111/01011980 200 1729 4450 14
8913 111111/01012011 200 1729 4450 4
9110 111111/010203 200 1729 4450 13
9218 111111/098765 200 1729 4450 3
9077 111111/0987654321 200 1729 4450 7
8783 111111/101010 200 1729 4450 6
8920 111111/102030 200 1729 4450 14
Pretty Raw \n Actions
1 POST /bWAPP/login.php HTTP/1.1
2 Host: 192.168.0.9:8080
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 51
Search... 0 matches
Pretty Raw Render \n Actions
1 HTTP/1.1 302 Found
2 Date: Thu, 31 Dec 2020 02:56:53 GMT
3 Server: Apache/2.4.46 (Win64) OpenSSL/1.1.
4 X-Powered-By: PHP/8.0.0
5 Expires: Thu, 19 Nov 1981 08:52:00 GMT
6 Cache-Control: no-store, no-cache, must-re
7 Pragma: no-cache
8 Set-Cookie: PHPSESSID=lrnfsn1rtjcehddg7o6C
9 Set-Cookie: security_level=0; expires=Fri,
10 Location: portal.php
11 Content-Length: 0
12 Keep-Alive: timeout=5, max=96
13 Connection: Keep-Alive
Search... 0 matches
Reqs: 16355 | Queued: 100 | Duration: 294 | RPS: 56 Connections: 1091 | Retries: 1071 | Fails: 0 | Next: letmein/butthead
Halt

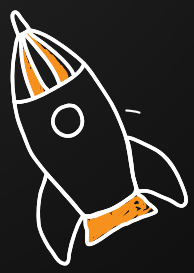
iGNITE Technologies

JOIN OUR TRAINING PROGRAMS

CLICK HERE

BEGINNER

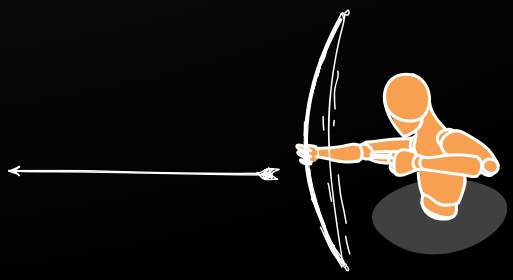
- Ethical Hacking
- Network Pentest
- Bug Bounty
- Wireless Pentest
- Network Security Essentials



ADVANCED

- Burp Suite Pro
- Android Pentest
- Web Services-API
- Advanced Metasploit
- Pro Infrastructure VAPT
- CTF
- Computer Forensics

EXPERT



- Red Team Operation
  - APT's - MITRE Attack Tactics
  - Active Directory Attack
  - MSSQL Security Assessment
- Privilege Escalation
  - Windows
  - Linux