

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) | Banner Grabbing | Reconocimiento de máquinas | Servidor/Cliente TCP | Packet Sniffing con Scapy | Paramiko (Cliente SSH) | Nmap con Python | MacChanger



WebServer2.py

```
WebServer2.py x
1 import socketserver
2 import http.server
3
4 class HttpRequestHandler(http.server.SimpleHTTPRequestHandler):
5
6     1 def do_GET(self):
7         2 if self.path == '/admin':
8             self.wfile.write(b'Esta pagina es solo para administradores!\n')
9             self.wfile.write(self.headers.as_bytes())
10        else:
11            3 http.server.SimpleHTTPRequestHandler.do_GET(self)
12
13
14 httpServer = socketserver.TCPServer(("", 20002), HttpRequestHandler)
15 httpServer.serve_forever()
```

1

Se sobrescribe el método do_GET()

2

Si el path contiene el directorio /admin enviará como respuesta un mensaje y las cabeceras HTTP del cliente.

3

De lo contrario llama al método do_GET() original.

Web Scrapping

Web Scraping



Técnica que automatiza la **extracción de información de sitios web**.

Se enfoca en la transformación de datos sin estructura en la web en datos estructurados.

Web Scraping

BeautifulSoup

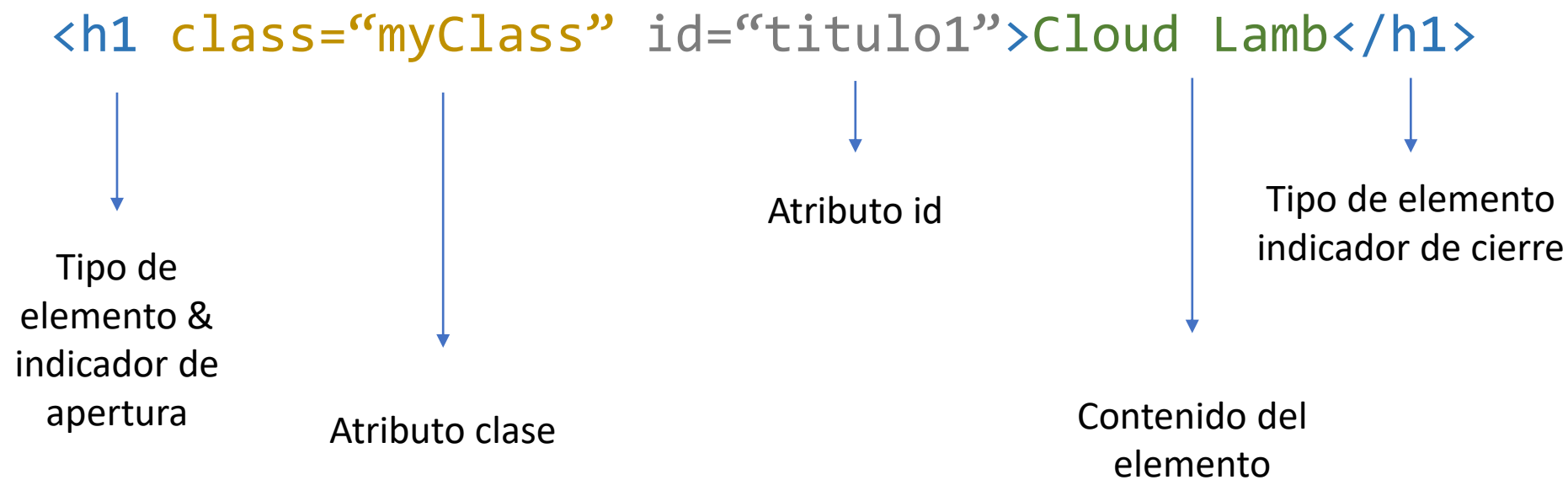


Librería de Python que **facilita el web scraping**.

- Extrae información de **archivos HTML**.
- Crea un árbol con todos los elementos del documento que permite **buscar de forma sencilla** elementos o etiquetas HTML como enlaces o formularios.
- Soporta **múltiples parsers** para el tratamiento de los archivos.

Web Scrapping

Elementos HTML



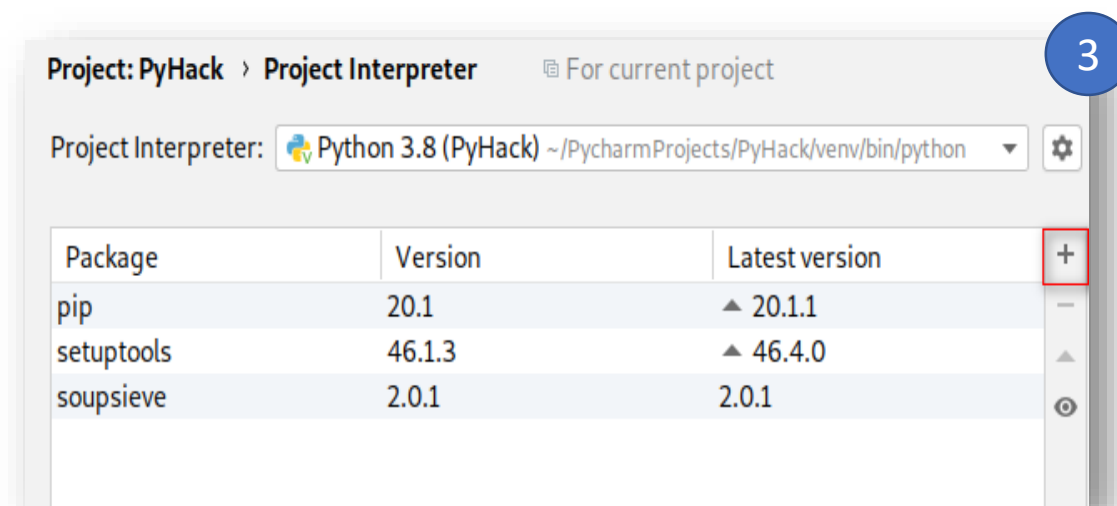
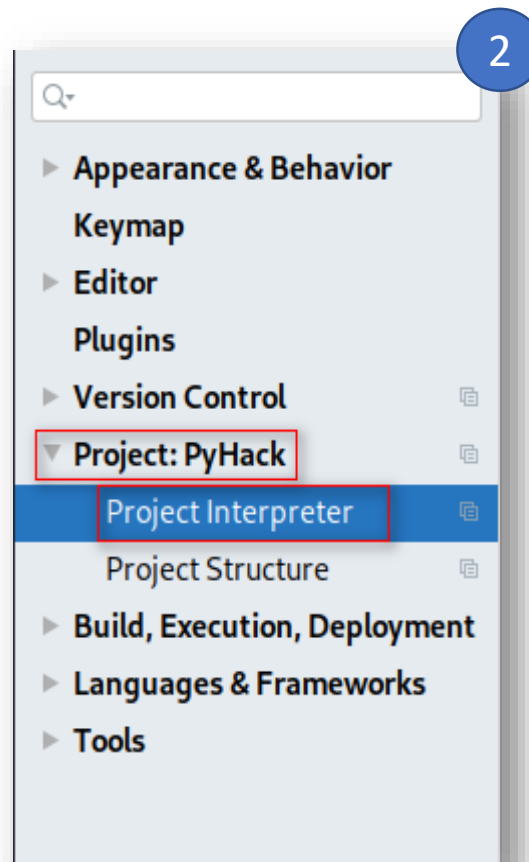
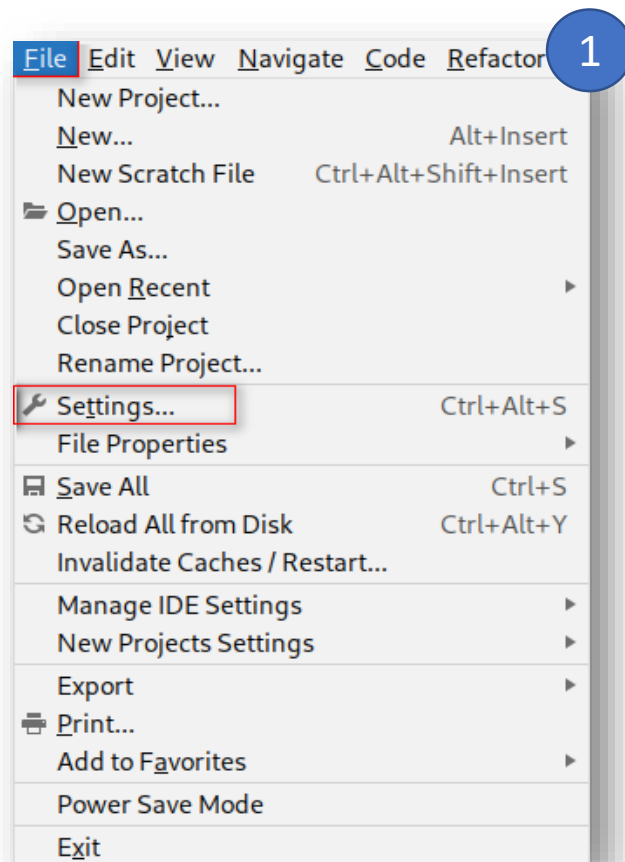


Script

Web Scraping

Librerías

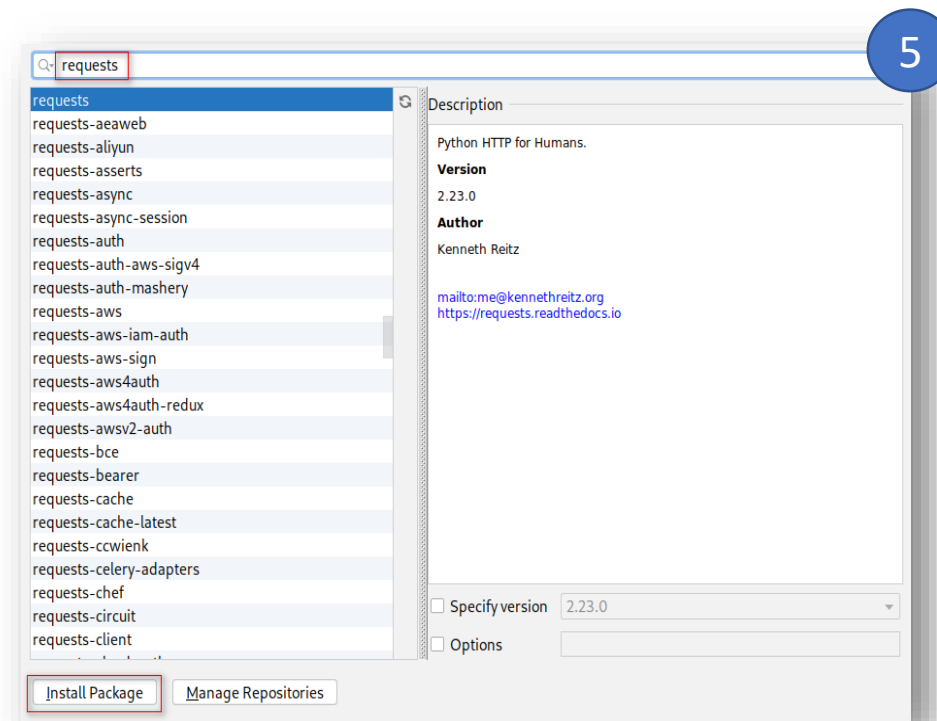
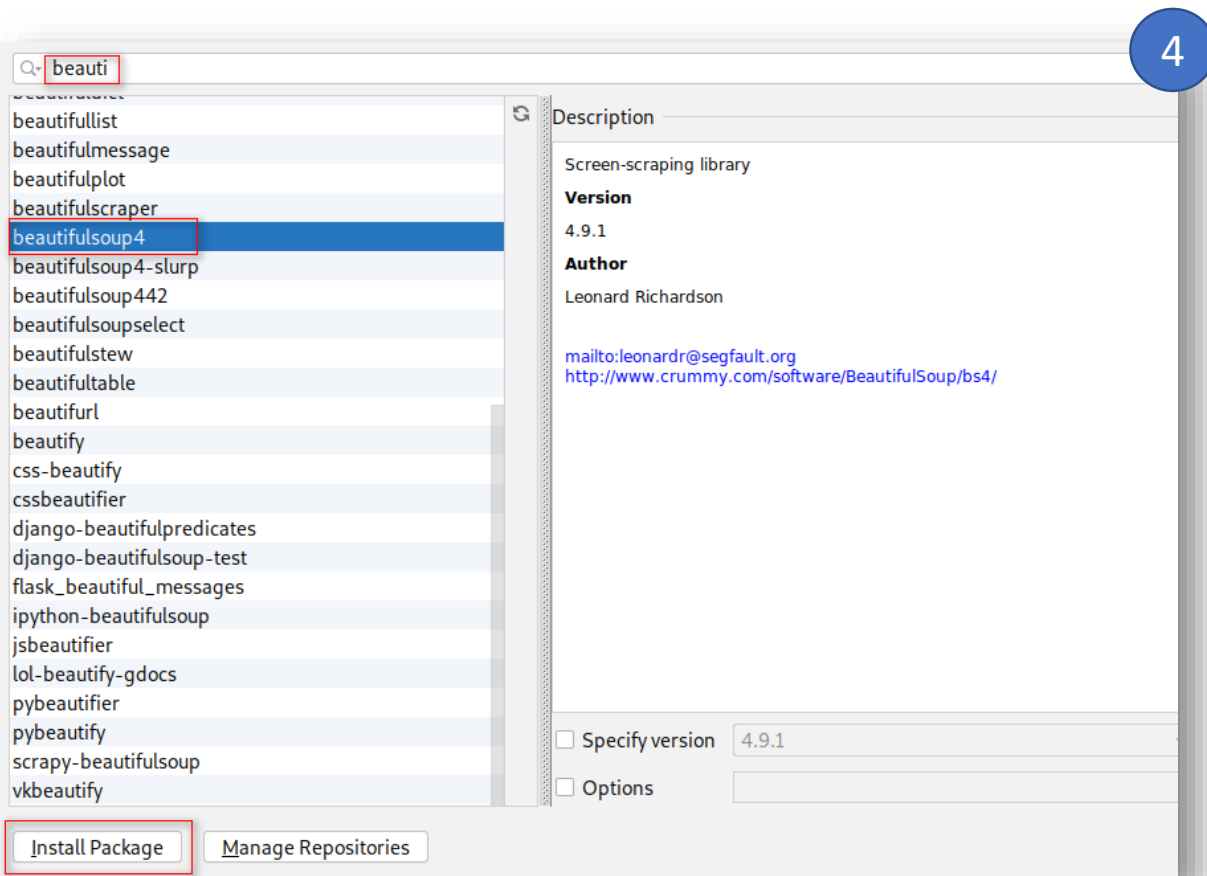
Instalar las librerías *beautifulsoup4* y *requests* en PyCharm.



Web Scraping

Librerías

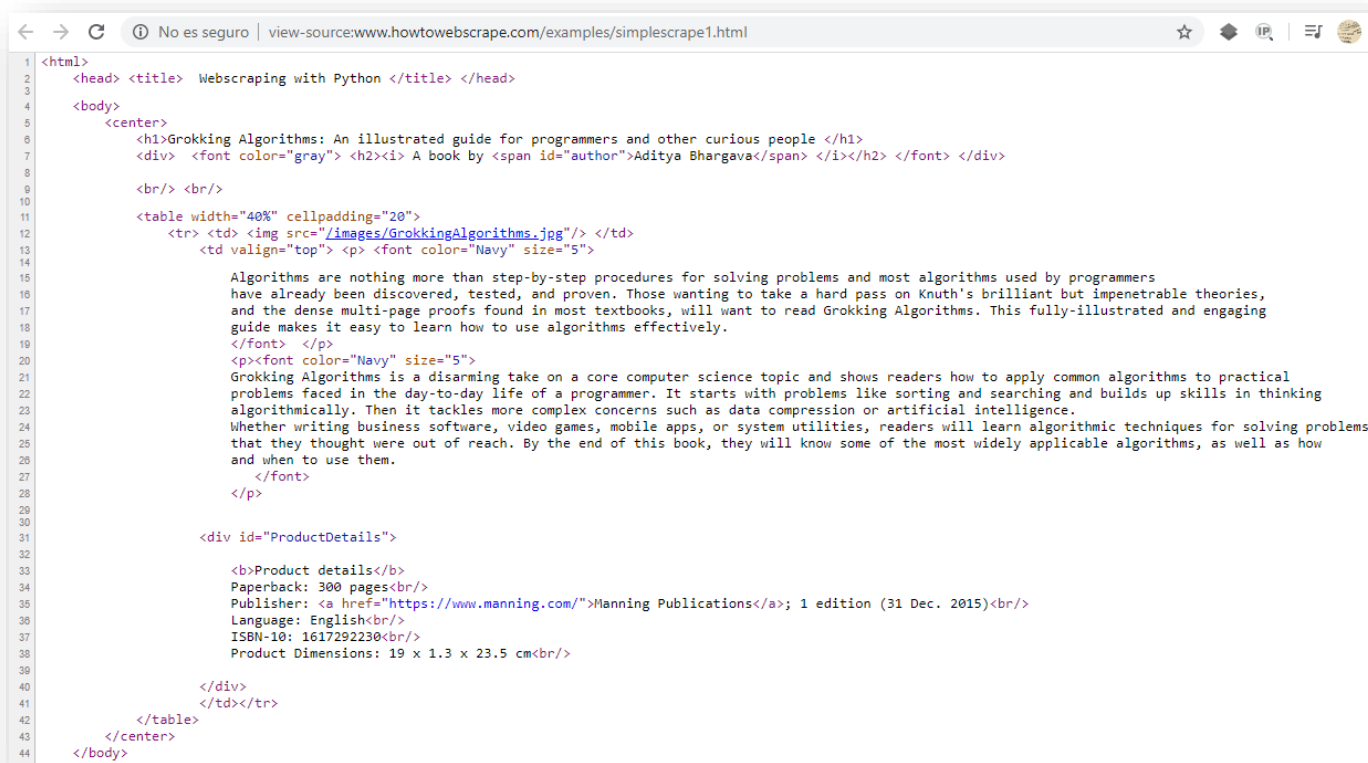
Instalar las librerías *beautifulsoup4* y *requests* en PyCharm.



Web Scraping

Página a utilizar

Se utilizará la página www.howtowscape.com/examples/simplescape1.html como ejemplo para el primer script de web scraping.



```
1 <html>
2   <head> <title> Webscraping with Python </title> </head>
3
4   <body>
5     <center>
6       <h1>Grokking Algorithms: An illustrated guide for programmers and other curious people </h1>
7       <div> <font color="gray"> <h2><i> A book by <span id="author">Aditya Bhargava</span> </i></h2> </font> </div>
8
9       <br> <br>
10
11      <table width="40%" cellpadding="20">
12        <tr> <td>  </td>
13        <td valign="top"> <p> <font color="Navy" size="5">
14
15          Algorithms are nothing more than step-by-step procedures for solving problems and most algorithms used by programmers
16          have already been discovered, tested, and proven. Those wanting to take a hard pass on Knuth's brilliant but impenetrable theories,
17          and the dense multi-page proofs found in most textbooks, will want to read Grokking Algorithms. This fully-illustrated and engaging
18          guide makes it easy to learn how to use algorithms effectively.
19          </font> </p>
20          <p><font color="Navy" size="5">
21            Grokking Algorithms is a disarming take on a core computer science topic and shows readers how to apply common algorithms to practical
22            problems faced in the day-to-day life of a programmer. It starts with problems like sorting and searching and builds up skills in thinking
23            algorithmically. Then it tackles more complex concerns such as data compression or artificial intelligence.
24            Whether writing business software, video games, mobile apps, or system utilities, readers will learn algorithmic techniques for solving problems
25            that they thought were out of reach. By the end of this book, they will know some of the most widely applicable algorithms, as well as how
26            and when to use them.
27            </font>
28          </p>
29
30          <div id="ProductDetails">
31
32            <b>Product details</b>
33            Paperback: 300 pages<br/>
34            Publisher: <a href="https://www.manning.com/">Manning Publications</a>; 1 edition (31 Dec. 2015)<br/>
35            Language: English<br/>
36            ISBN-10: 1617292230<br/>
37            Product Dimensions: 19 x 1.3 x 23.5 cm<br/>
38
39          </div>
40        </td></tr>
41      </table>
42    </center>
43  </body>
```

WebScraper.py

```
01. from bs4 import BeautifulSoup
02. 1 import requests
03.
04. 2 url = "http://www.howtowscape.com/examples/simplescape1.html"
05. 2 webpage = requests.get(url)
06.
07. 3 soup = BeautifulSoup(webpage.content, "html.parser")
08.
09.     print(soup.head) 4
10.     print(soup.title) 5
11.     print(soup.h1) 6
12.     print(soup.div.i) 7
13.
14.     centercon = soup.center.contents
15.     print(centercon)
16.     print(centercon[1])
17.     print(centercon[3])
18.
19.     for s in soup.stripped_strings:
20.         print("String: " + s)
21.
22.     r = soup.find(id='ProductDetails')
23.     print(r)
24.
25.     print(r.text)
26.
27.     r = soup.find_all('div')
28.     for div_tag in r:
29.         print(div_tag)
30.
31.     print("\nValor del atributo IMG:")
32.     r = soup.find('img')
33.     print(r['src'])
34.     print(r.get('src'))
```

- 1 Librería que permite enviar peticiones HTTP/1.1 al servidor especificado
- 2 Se guarda la respuesta del servidor en la variable webpage (que viene siendo el código HTML de la página)
- 3 Se parsea el código HTML con BeautifulSoup.
- 4 Se obtiene la etiqueta *head*.
- 5 Se obtiene la primera etiqueta *title*.
- 6 Se obtiene la primera etiqueta *h1*.
- 7 Se obtiene la primera etiqueta *i* contenida en la primera etiqueta *div*.¹²

WebScraper.py

```
01. from bs4 import BeautifulSoup
02. 1 import requests
03.
04. 2 url = "http://www.howtowscape.com/examples/simplescape1.html"
05. 2 webpage = requests.get(url)
06.
07. 3 soup = BeautifulSoup(webpage.content, "html.parser")
08.
09.     print(soup.head) 4
10.     print(soup.title) 5
11.     print(soup.h1) 6
12.     print(soup.div.i) 7
13.
14.     centercon = soup.center.contents 8
15.     print(centercon)
16.     print(centercon[1]) 9
17.     print(centercon[3]) 10
18.
19.     for s in soup.stripped_strings: 11
20.         print("String: " + s)
21.
22.     r = soup.find(id='ProductDetails') 12
23.     print(r)
24.
25.     print(r.text) 13
26.
27.     r = soup.find_all('div')
28.     for div_tag in r: 14
29.         print(div_tag)
30.
31.     print("\nValor del atributo IMG:")
32.     r = soup.find('img')
33.     print(r['src'])
34.     print(r.get('src'))
```

- 8 Se obtienen las etiquetas hijas de la etiqueta *center* y las almacena en una lista.
- 9 Se obtiene el segundo elemento de la lista.
- 10 Se obtiene el tercer elemento de la lista.
- 11 Obtiene todas las cadenas que contengan las etiquetas.
- 12 Búsqueda del primer elemento que contenga el id especificado.
- 13 Obtener el contenido de un elemento.
- 14 Obtener todas las etiquetas *div*.

WebScraper.py

```
01. from bs4 import BeautifulSoup
02. 1 import requests
03.
04. 2 url = "http://www.howtowscape.com/examples/simplescraper1.html"
05. 2 webpage = requests.get(url)
06.
07. 3 soup = BeautifulSoup(webpage.content, "html.parser")
08.
09. print(soup.head) 4
10. print(soup.title) 5
11. print(soup.h1) 6
12. print(soup.div.i) 7
13.
14. centercon = soup.center.contents 8
15. print(centercon)
16. print(centercon[1]) 9
17. print(centercon[3]) 10
18.
19. for s in soup.stripped_strings: 11
20.     print("String: " + s)
21.
22. r = soup.find(id='ProductDetails') 12
23. print(r)
24.
25. print(r.text) 13
26.
27. r = soup.find_all('div')
28. for div_tag in r: 14
29.     print(div_tag)
30.
31. print("\nValor del atributo IMG:")
32. r = soup.find('img')
33. print(r['src'])
34. print(r.get('src')) 15
```

15

Se obtiene la primer etiqueta *img*.

Se obtiene el valor del atributo *src* de la etiqueta *img*

Fuerza Bruta de Directorios Web

Fuerza Bruta a Directorios Web



Es una técnica empleada en web hacking que automatiza la **identificación de directorios y archivos de una aplicación web.**

Fuerza bruta a directorios web

¿Cómo funciona?

Brute Force Tool

URL

`http://ejemplo.com`

DICCIONARIO DE DIRECTORIOS Y
ARCHIVOS

admin

js

img

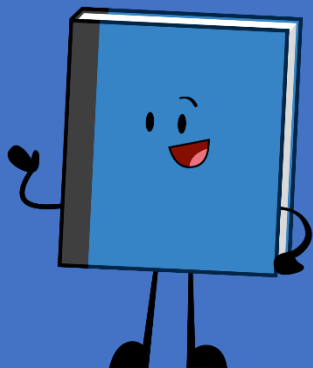
src

log

user

init.php

index.html



Petición HTTP

`http://ejemplo.com/admin`

`http://ejemplo.com/js`

`http://ejemplo.com/img`

...

...

`http://ejemplo.com/init.php`

Código de estado
de respuesta

HTTP

200

200

403

...

...

200

BruteForce.py

```
01. import threading
02. import queue
03. import urllib.parse
04. import urllib.request
05. import urllib.error
06.
07.
08. threads = 50
09. url_objetivo = "http://testphp.vulnweb.com"
10. dict = "/home/kali/all.txt"
11. user_agent = "Mozilla/5.0 (X11; Linux x86_64; rv:19.0) Gecko/20100101 Firefox/19.0"
12.
13. def construir_dict(dict):
14.     print("Procesando diccionario...")
15.
16.     arch = open(dict, "r")
17.     raw_palabras = arch.readlines()
18.     arch.close()
19.
20.     palabras = queue.Queue()
21.
22.     for palabra in raw_palabras:
23.         palabra = palabra.rstrip()
24.         palabras.put(palabra)
25.
26.     return palabras
```

Librerías

Línea 01

Librería que construye hilos.

Línea 02

Librería que permite trabajar con colas tipo FIFO o LIFO.

Línea 03-05

Librería que permite trabajar con URLs.

BruteForce.py

```
01. import threading
02. import queue
03. import urllib.parse
04. import urllib.request
05. import urllib.error
06.
07.
08. threads = 50
09. url_objetivo = "http://testphp.vulnweb.com"
10. dict = "/home/kali/all.txt"
11. user_agent = "Mozilla/5.0 (X11; Linux x86_64; rv:19.0) Gecko/20100101 Firefox/19.0"
12.
13. def construir_dict(dict):
14.     print("Procesando diccionario...")
15.
16.     arch = open(dict, "r")
17.     raw_palabras = arch.readlines()
18.     arch.close()
19.
20.     palabras = queue.Queue()
21.
22.     for palabra in raw_palabras:
23.         palabra = palabra.rstrip()
24.         palabras.put(palabra)
25.
26.     return palabras
```

Variables

Línea 08

Número de hilos a utilizar.

Línea 09

URL objetivo.

Línea 10

Diccionario de directorios y archivos a utilizar*.

Línea 11

Cadena que describe el cliente que hace las peticiones HTTP.

* <https://www.netsparker.com/s/research/SVNDigger.zip>

BruteForce.py

```
01. import threading
02. import queue
03. import urllib.parse
04. import urllib.request
05. import urllib.error
06.
07.
08. threads = 50
09. url_objetivo = "http://testphp.vulnweb.com"
10. dict = "/home/kali/all.txt"
11. user_agent = "Mozilla/5.0 (X11; Linux x86_64; rv:19.0) Gecko/20100101 Firefox/19.0"
12.
13. def construir_dict(dict):
14.     print("Procesando diccionario...")
15.
16.     arch = open(dict, "r")
17.     raw_palabras = arch.readlines()
18.     arch.close()
19.
20.     palabras = queue.Queue()
21.
22.     for palabra in raw_palabras:
23.         palabra = palabra.rstrip()
24.         palabras.put(palabra)
25.
26.     return palabras
```

Función construir_dict()

- Línea 16 Abre el archivo del diccionario.
- Línea 17 Lee todas las líneas del archivo.
- Línea 18 Cierra el archivo
- Línea 20 Instancia de la clase Queue.
- Línea 22-24 Por cada palabra del diccionario se le quitarán espacios en blanco (en caso de existir) y se almacenarán en el objeto *palabras*.
- Línea 26 La función regresa el objeto *palabras*.

BruteForce.py

```
28. def fuerza_bruta(fila_palabra, extensiones=None):
29.     while not fila_palabra.empty():
30.         intento = fila_palabra.get()
31.
32.         lista_intento = []
33.
34.         if '.' not in intento:
35.             lista_intento.append("/%s/" % intento)
36.         else:
37.             lista_intento.append("/%s" % intento)
38.
39.         if extensiones:
40.             for extension in extensiones:
41.                 lista_intento.append("/%s%s" % (intento,extension))
42.
43.         for i in lista_intento:
44.             url = "%s%s" % (url_objetivo,urllib.parse.quote(i))
45.
46.             try:
47.                 headers = {}
48.                 headers["User-Agent"] = user_agent
49.                 r = urllib.request.Request(url,headers=headers)
50.
51.                 respuesta = urllib.request.urlopen(r)
52.
53.                 if len(respuesta.read()):
54.                     print("[%d] => %s" % (respuesta.code,url))
55.             except urllib.error.URLError as e:
56.                 if hasattr(e, 'code') and e.code != 404:
57.                     print("!!! %d => %s" % (e.code,url))
58.                 pass
```

Función fuerza_bruta()

- Línea 29** Ejecutará el código hasta que la fila de palabras esté vacía.
- Línea 30** Saca una palabra de la fila.
- Línea 32** Crea una lista vacía
- Línea 34-37** Si la palabra es un directorio coloca "/" entre la palabra, sino sólo la coloca al principio.
- Línea 39-41** Si se han especificado extensiones se concatenan una por una con la palabra y se agregan a la lista.
- Línea 43** Por cada palabra en la lista ejecutará el código.

BruteForce.py

```
28. def fuerza_bruta(fila_palabra, extensiones=None):
29.     while not fila_palabra.empty():
30.         intento = fila_palabra.get()
31.
32.         lista_intento = []
33.
34.         if '.' not in intento:
35.             lista_intento.append("/%s/" % intento)
36.         else:
37.             lista_intento.append("/%s" % intento)
38.
39.         if extensiones:
40.             for extension in extensiones:
41.                 lista_intento.append("/%s%s" % (intento,extension))
42.
43.         for i in lista_intento:
44.             url = "%s%s" % (url_objetivo,urllib.parse.quote(i))
45.
46.             try:
47.                 headers = {}
48.                 headers["User-Agent"] = user_agent
49.                 r = urllib.request.Request(url,headers=headers)
50.
51.                 respuesta = urllib.request.urlopen(r)
52.
53.                 if len(respuesta.read()):
54.                     print("[%d] => %s" % (respuesta.code,url))
55.             except urllib.error.URLError as e:
56.                 if hasattr(e, 'code') and e.code != 404:
57.                     print("!!! %d => %s" % (e.code,url))
58.                 pass
```

Función fuerza_bruta()

Línea 44

Se concatena la url con la palabra codificada(URL encoded).

Línea 47-48

Se crea un diccionario con el User-Agent a utilizar.

Línea 49

Genera la petición HTTP con la url y la cabecera.

Línea 51

Envía la petición HTTP al sitio web.

Línea 53-54

Si se ha obtenido una respuesta del sitio web se imprime en pantalla la url utilizada y el código de respuesta HTTP.

Línea 55-58

Cuando se presenta un error se determina si el error tiene un código y es diferente a 404 entonces imprime en pantalla el código y la url.

BruteForce.py

```
60. fila_palabras = construir_dict(dict)
61. extensiones = [".php", ".bak", ".orig", ".inc"]
62.
63. print("Comenzando fuerza bruta...")
64. for i in range(threads):
65.     t = threading.Thread(target=fuerza_bruta, args=(fila_palabras, extensiones,))
66.     t.start()
```

Código principal

Línea 60 Llamada a la función *construir_dict()*

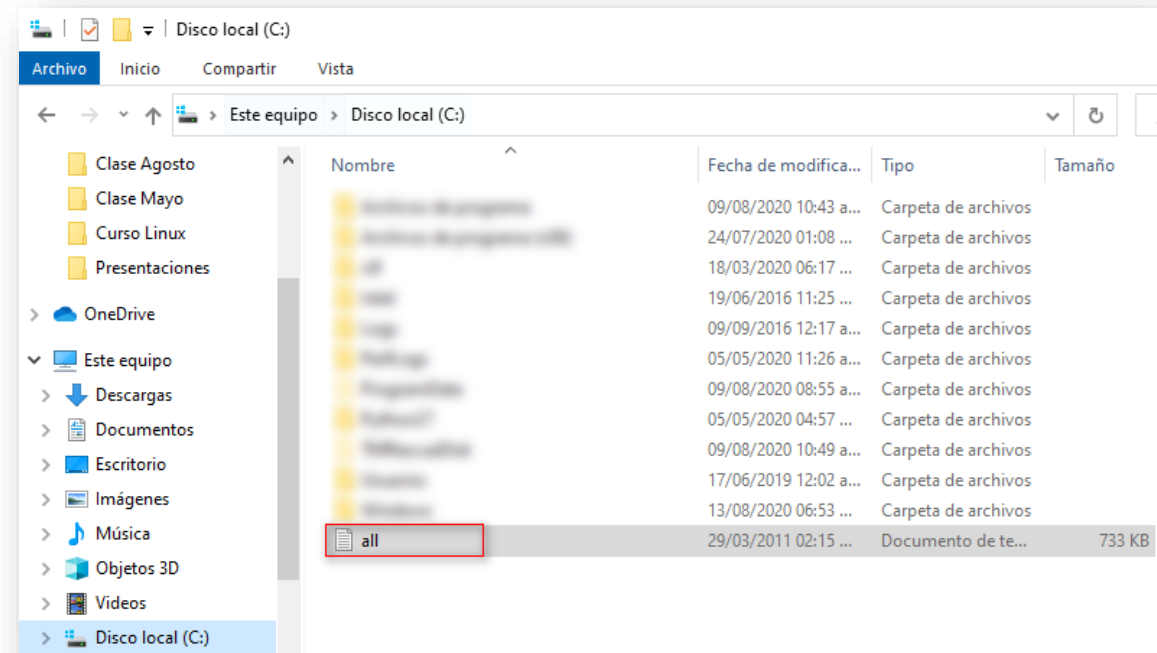
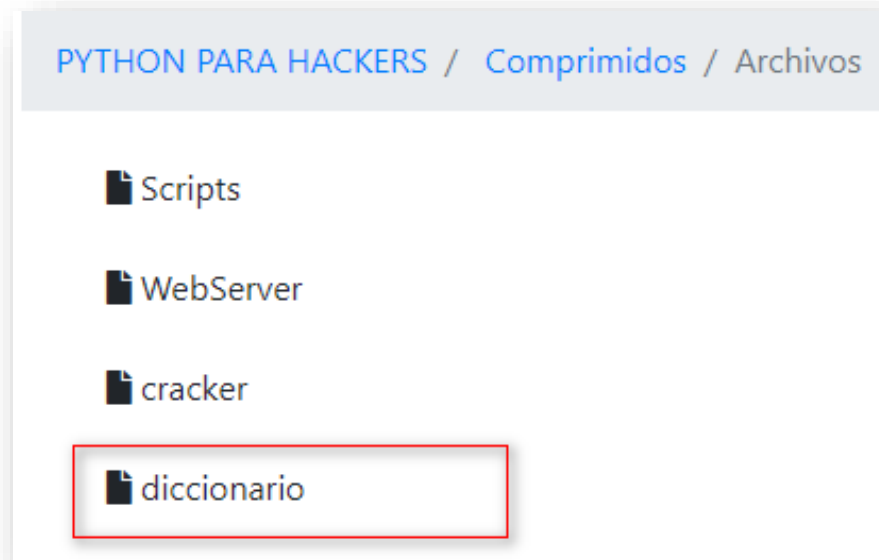
Línea 61 Definición de las extensiones.

Línea 64-66 Se definen y comienzan los hilos los cuales ejecutan la función *fuerza_bruta()* simultáneamente.

BruteForce.py

Diccionario

Descargar el archivo “diccionario.zip” de la biblioteca de **Cloud Lamb** y descomprimirlo. Colocarlo sobre el directorio C:\.



BruteForce.py

Resultado

```
Procesando diccionario...  
Comenzando fuerza bruta...  
[200] => http://testphp.vulnweb.com/CVS/  
[200] => http://testphp.vulnweb.com/admin/  
[200] => http://testphp.vulnweb.com/index.php  
[200] => http://testphp.vulnweb.com/index.bak  
[200] => http://testphp.vulnweb.com/search.php  
[200] => http://testphp.vulnweb.com/login.php  
[200] => http://testphp.vulnweb.com/login.php  
[200] => http://testphp.vulnweb.com/images/
```

Próxima clase...

- **Capítulo 4:** Hands-On (Parte IV)
- **Evaluación** (Capítulo 3)



NEXT >>
Mar, 01 Sept



CloudLamb

**¡Muchas gracias por su
atención!**