User Manual

Student: Sean O'Connor – C00224424

Supervisor: Richard Butler

Cybercrime & I.T. Security – CW_KCCYB_B

Institute of Technology Carlow

Contents

Requirements

Operating System:
Tested and operational on Windows 10

Programming Language:
Tested and operational on Python 3.9

Python Modules Needed:

Beautifulsoup4	Queue (queue.py file will be in same directory)
Threading	Requests
Socket	Datetime
Sys	OS
Datetime	Subprocess

Other Requirements: (keep in same directory as main file)
jython-standalone-2.7.2.jar

exceptions_fix.py

Installation

Download all necessary Python Modules referenced above as well as the Standalone Jython.jar file from the Jython website and the exceptions_fix.py from the following link:
https://github.com/securityMB/burp-exceptions

This will help increase the readability of exceptions raised by the Burp Suite extensions written in python. (securityMB, 2015)

Create a folder for the extension file to be stored in alongside your Jython Jar file.

Open Burp Suite as an Admin user. Admin rights are needed for multiple functions.

Once in the suite, go to extender tab, and enter the options menu. Go to the Python Environment and select the file location for the standalone Jython file. You should also select the location of the folder where all python modules have been saved in.

After this go back to the extensions tab and click the add button. This will allow you to select the extension file we want to load up. This will bring up a pop-up screen showing the module being loaded into the suite. If any errors occur, they will be displayed back to the user. If the module is loaded successfully, the window will notify the user and they can close it.

Operation

Now that we have successfully loaded our extension, we will see the tab labelled Project Tab at the top. When you click into this you should see the 4 sub tabs, Ping, Threaded Port Scanner, Sub Domain and Scraper.

Ping & Ping Sweep

The Ping tab contains to operating functions, the first being the basic ping. This one uses the first text field on the left and the button located directly under it. Put in the correct IP address of the target or the targets valid host name and press the button and the ping operation will be performed, with the output being displayed in the Result text Box.

The Ping sweep function uses the remain three text boxes on the top and the button labelled sweep. To use this function, enter the first three segments of a valid IP address (E.G. 192.168.56) and in the other boxes enter in a start number for the range and an end number. When you press the sweep button, the function will append the numbers within the range you entered in and ping each IP address. If a response is received, the Active IP count is incremented, else the inactive IP count is incremented. Please note it is advised that ranges are kept small as receiving no response from an IP may take longer than a successful response. Once the sweep is complete, the Sweep Result box will show the number of Active and Inactive Ips. (Appendix A)

Threaded Port Scanner

The threaded port scanner has a simpler interface to use compared to the ping tab. It consists of a text field where you enter in a valid hostname or IP to scan. When you press the scan button, the function will execute in the background, scanning each port on the target and checking if they are open. If they are open, the Ports number is appended to a list. In the Suites built in console, the port that has been scanned will be printed out. When the port scanner reaches the end of the range of 500 ports, the list of open ports will be displayed in the Open Result box and the time taken will be displayed. Please note while this function is executing, you cannot access any other functions and must wait for it to finish executing. (Appendix B)

Sub Domain

The Sub Domain scanner tab has two input fields for the user to use before executing the function. The first asks for a target host name, for demonstration purposes you may use public-firing-range.appspot.com. The second field asks the user to select a file, 1, 2 or 3. Each option selects a subdomains file which the program will read from to find the potential subdomains. 1 will access the smallest file of 100 subdomains, 2 accesses the 1000 subdomains file and 3 accesses the 10000 subdomains file. The first option is the default and the quickest. If the user leaves this field empty, the program will read the 100 subdomains file. When the user presses the central Lookup button, the program will execute, reading the appropriate file and sending out requests for potential subdomains. Whenever a subdomain is found that subdomain is appended to a list. Once the end of the file is reached, the list of discovered subdomains is returned to the user in the Result Box as well as the time taken. (Appendix C)

Scraper

This function does not operate in the suite, only in command prompt and has no functionality in the suite but if it did, the user would enter in a domain name including the http(s):// in the domain and then pressing the scrape button. The executing function will find this domain and scrape the HTML contents from the page and return it back to the user in the Result box. (Appendix D)

Bibliography

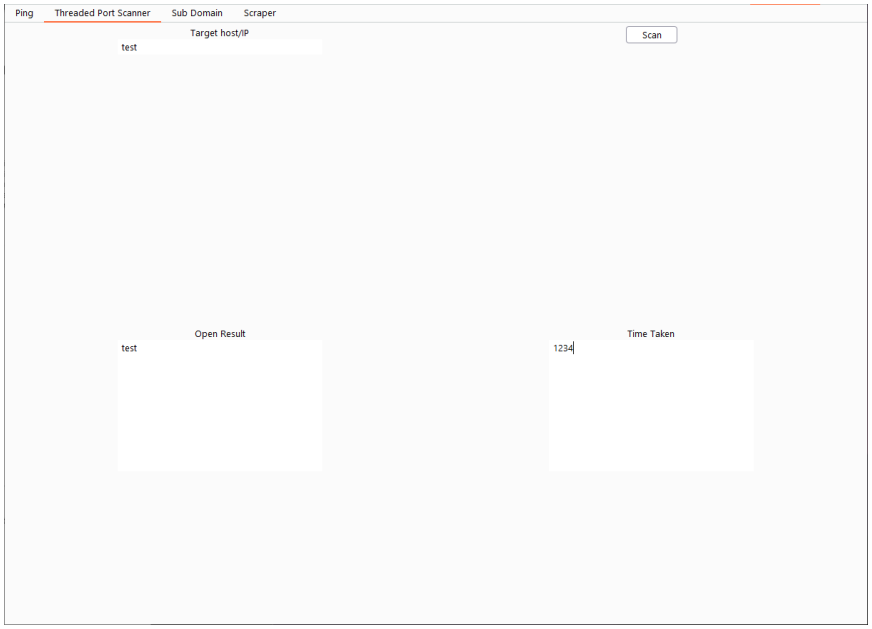
securityMB, 2015. *burp-exceptions.* [Online]
Available at: https://github.com/securityMB/burp-exceptions
[Accessed 4 February 2021].

Appendix

Appendix A

```
def pingFunc(self, event):
    import os
    from datetime import datetime
    from datetime import timedelta
    import socket
    import subprocess

    #function start, timestamp taken
    t1 = datetime.now()

    #initialise variables
    packRec = 0
    packLoss = 0
    #get user input and convert to an IP
    targetIP = self.inputArea.text
    ip = socket.gethostbyname(targetIP)
    #issue ping command
    response = os.system("ping -c 1 " + ip)

    if response == 0:
        #if reply is recieved
        packRec += 1
        str(ip)
        upRes = ' Reply/Replies from target ip: ' + ip
        self.resultArea.text = upRes + "\n" + upRes + "\n" + upRes+ "\n" + upRes
        print ("test")
    else:
        #if no reply from target
        packLoss += 1
        str(ip)
        downRes = 'No reply from target ip: ' + ip
        self.resultArea.text = downRes +"\n" + downRes +"\n" + downRes +"\n" + downRes
    #timestamp at end
    t2 = datetime.now()
    total = t2 - t1
    print(str(total))

    return
```

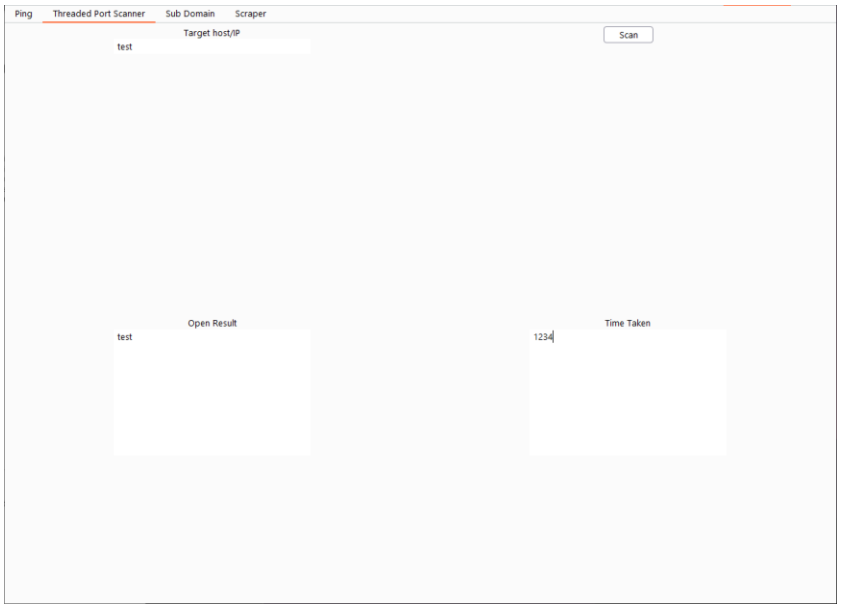
```
def pingsweep(self, event):
    import subprocess
    import os
    #takes in user input
    target = self.inputAreaB.text
    minR = self.inputAreaB1.text
    maxR = self.inputAreaB2.text
    inactive = 0
    active = 0
    #sets for loop for sweep range
    for n in range(int(minR), int(maxR)):
        #appends loop number onto IP
        sweep = target + ".{0}".format(n)
        #opens subprocess to ping using OS
        response = subprocess.Popen(["ping", "-c", "1", "-n", "1", sweep]).wait()
        if response == 0:
                print (sweep, "active")
                active += 1
        else:
                print (sweep, "inactive")
                inactive += 1

    #converts ints to strings and sets the to text area
    inactiveRes = "Inactive IP's: " + str(inactive)
    activeRes = "Active IP's: " + str(active)
    self.resultAreaB.text = inactiveRes + "\n" + activeRes

    print("Inactive IP's: " + str(inactive))
    print("Active IP's: " + str(active))
    return
```



Appendix B

```
def scan(self, event):
    import socket
    import subprocess
    import sys
    from datetime import datetime
    from queue import Queue
    import threading
    import time

    #prevents double modification of shared variables
    print_lock = threading.Lock()
    #get user input and assign
    target = self.inputArea3.text
    targetIP = socket.gethostbyname(target)
    #create list and get start timestamp
    openRes_list = []
    t1 = datetime.now()

    def portscan(port):
        openPorts = 0
        closePorts = 0
        #connection oriented TCP protocol
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try:
            #gets connection with port
            conn = sock.connect((targetIP, port))
            with print_lock:
                #sets oresult to open port and appends to list
                openPorts += 1
                oresult = "Port {}:     Open".format(port)
                print("Port {}:     Scanned".format(port))
                openRes_list.append(oresult)
                #close connection
                conn.close()
        except:
            closePorts += 1
            print("Port {}:     Scanned".format(port))
```

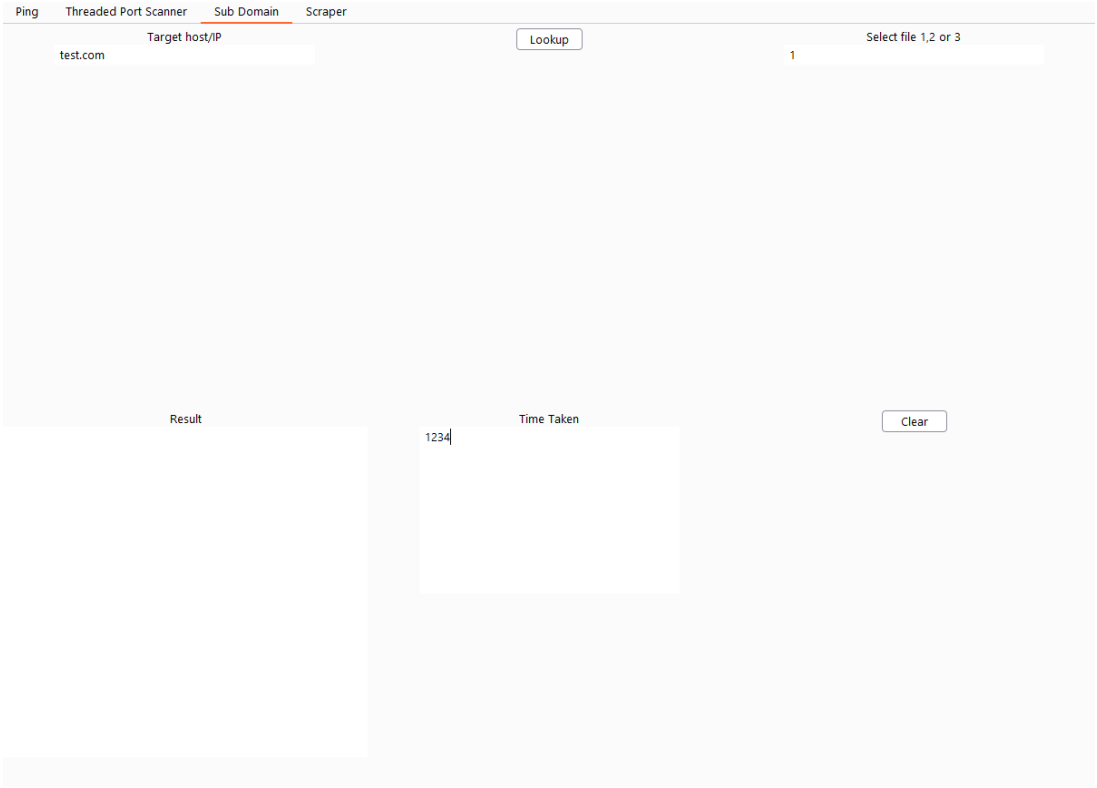
```
#pulls job from que
def threader():
    job = que.get()
    #runs job on portscan
    portscan(job)
    que.task_done()
#create queue
que = Queue()

#number of threads (match with job assign for loop)
for thread in range(500):
    thread = threading.Thread(target=threader)
    #assigned as daemon so it dies when main dies
    thread.daemon = True
    thread.start()

for job in range(1,500):
    que.put(job)
#joing queue till thread terminates.
que.join()
#assign list to result area and get and return time
self.resultArea3A.text = str(openRes_list)
print(openRes_list)
t2 = datetime.now()
total = t2 - t1
timing = "Time taken: " + str(total)
self.resultArea3C.text = timing
return
```



Appendix C

```
def lookup(self, event):
    import requests
    import socket
    from queue import Queue
    import threading
    from datetime import datetime
    #get time start, get inputs
    t1 = datetime.now()
    domain = self.inputArea4.text
    option = self.inputArea4A.text
    #if option 1 chosen, do this
    if option == "1":
        #open, read and assign contents from file
        file = open("subdomains-100.txt")
        content = file.read()
        subdomains = content.splitlines()
        #create list
        discovered_subdomains = []
        for subdomain in subdomains:
            count = 0
            #append http to subdomain and domain
            url = "http://{}.{}".format(subdomain,domain)
            #ip = socket.gethostbyname(url)
            try:#get request from url
                requests.get(url)
                #ip = socket.gethostbyname(url)
            except requests.ConnectionError:
                pass
            else:#if subdomain count +1 and append url to list
                count += 1
                print("Discovered subdomain:", url)
                discovered_subdomains.append(url)
                #print(ip)
        #return list to user and time
        self.resultArea4.text = str(discovered_subdomains)
        t2 = datetime.now()
        total = t2 - t1
        timing = "Time taken: " + str(total)
        self.resultArea4B.text = timing
        print(count)
```

```
##clear button for subdomain crawler
def clear(self, event):
    self.resultArea4.text = ""
    self.resultArea4B.text = ""
    return
```



Appendix D

```
def scraper(self, event):
    import requests
    from bs4 import BeautifulSoup
    #takes user input and gets a request from target
    target = self.inputArea2.text

    ##appends http onto entered domain and gets request from page.
    url = "http://{}".format(target)
    page = requests.get(url)

    #runs page through Beautiful Soup and parses the content
    soup = BeautifulSoup(page.content, 'html.parser')
    print(soup)
    self.resultArea2.text = soup
    return
```

```
public-firing-range.appspot.com
<!DOCTYPE html>

<html>
<head>
<title>Firing Range</title>
</head>
<body>
<h1>Version 0.48</h1>
<h1>What is the Firing Range?</h1>
<p>
     Firing Range is a test bed for automated web application security scanners. <br/>
     Its test cases are not meant
     to be hard to reach or exercise, as the site can be very easily crawlable. <br/>
     The testbed focuses on detection capabilities, presenting many variants of vulnerabilities and
     hard-to-detect edge cases.<br/>
     For more details, see the <a href="https://github.com/google/firing-range">GitHub page</a>.
  </p>
<ul>
<li><a href="/address/index.html">Address DOM XSS</a></li>
<li><a href="/angular/index.html">Angular-based XSSes</a></li>
<li><a href="/badscriptimport/index.html">Bad JavaScript imports</a></li>
<li><a href="/cors/index.html">CORS related vulnerabilities</a></li>
<li><a href="/dom/index.html">DOM XSS</a></li>
<li><a href="/escape/index.html">Escaped XSS</a></li>
<li><a href="/flashinjection/index.html">Flash Injection</a></li>
<li><a href="/mixedcontent/index.html">Mixed content</a></li>
<li><a href="/redirect/index.html">Redirect XSS</a></li>
<li><a href="/reflected/index.html">Reflected XSS</a></li>
<li><a href="/remoteinclude/index.html">Remote inclusion XSS</a></li>
<li><a href="/reverseclickjacking/">Reverse ClickJacking</a></li>
<li><a href="/tags/index.html">Tag based XSS</a></li>
<li><a href="/urldom/index.html">URL-based DOM XSS</a></li>
<li><a href="/vulnerablelibraries/index.html">Vulnerable libraries</a></li>
<li><a href="/leakedcookie/index.html">Leaked httpOnly cookie</a></li>
<li><a href="/invalidframingconfig/index.html">Invalid framing configuration</a></li>
</ul>
```