

# **Final Report**

**Student: Sean O'Connor – C00224424**

**Supervisor: Richard Butler**

**Cybercrime & I.T. Security – CW\_KCCYB\_B**

**Institute of Technology Carlow**

## Contents

Introduction .....	3
Project Description.....	3
Project Achievements .....	3
Module Descriptions .....	3
Ping and Ping Sweep.....	3
Threaded Port Scanner .....	3
Subdomain Crawler .....	3
Problems and Resolutions .....	4
Language Choice.....	4
GUI creation and Ruby.....	4
Burp and Jython Import issues.....	5
Failed Objectives and Main Differences .....	6
Log Function.....	6
HTML Scraper.....	6
NSlookup functionality.....	6
Ruby & Switch to Python .....	6
GUI Layouts.....	6
What I have Learned.....	7
Advanced python knowledge .....	7
Java Swing.....	7
Burp Suite knowledge .....	7
Basics of Ruby .....	7
Different Start.....	7
Personal Lessons .....	7
Acknowledgements .....	8
Bibliography.....	9
Appendix.....	10
Appendix A.....	10
Appendix B .....	12
Appendix C.....	13
Appendix D .....	14

## Introduction

In this report, I will be giving a description and summary of my final Year Project. I will discuss my project achievements, failures, problems and resolutions and I will also discuss how I feel the project went for myself and what changes, if any, I would make if I had to start over again.

## Project Description

For my final year project, I was testing the Extendibility of the Burp Suite via Java Python and Ruby. The Burp Suite is a suite of web application pen testing tools that allows users to load custom extensions onto the suite for different uses. My project was to firstly test and compare three languages that the Burp Suite allowed, and then after this comparison was completed, create an extension of functions in the language I had chosen from my comparison.

## Project Achievements

By the end of the project, I had:

- Created a Java Swing GUI in the suite via Python.
- Created a working Ping and Ping Sweep function.
- Implemented a Port Scanner with Threading.
- Implemented a Subdomain Crawler.

## Module Descriptions

### Ping and Ping Sweep

The Ping function allows a user to enter a target host or IP and ping them, receiving a reply that tells if there was a response and the time taken. The Ping sweep function allows a user to enter the first 3 segments of an IP address, and then enter a range and do a sweep of that range, returning a count of active and inactive Ips.

(See **Appendix A**)

### Threaded Port Scanner

The Threaded Port Scanner allows a user to enter in a target host or IP and perform a scan which will check what ports are open on said target. With a threader, it operates very quickly and returns a full list of what port numbers are open.

(See **Appendix B**)

### Subdomain Crawler

The Subdomain Crawler allows a user to enter in a target domain, then select a file to be used for a scan. Once a file is selected, the scan will append a potential subdomain onto the entered domain and attempt to get a request from the potential subdomain. If a request is acknowledged, that subdomain is added to a list, which is returned to the user once the end of the scanning file is reached.

(See **Appendix C**)

## Problems and Resolutions

During this project, I have encountered many problems, ranging from choosing a language to do the project in to implementing the GUI and functions into the Suite itself. The first problem I encountered was trying to decide between Java Ruby and Python for which language I would be using going forward.

### Language Choice

When creating an extension for the Burp suite the suite offers you a choice of three languages, Java, the default language, Python and Ruby. So, in the early stages of this project, I had to test each language and try to create a basic extension tab with each. When I tried with Java, I kept encountering error after error because I am not as proficient with Java as I am with other coding languages. This led me to decide not to choose this language going forward as I felt it would slow down the overall progress of the project if I was already encountering issues at this basic level. So, this led me to test and compare Python and Ruby. Both languages were not as difficult to implement as Java and they both seemed to operate on the same level of programming. So, because of their similarity I decided to choose Ruby as the language for the project as I already had experience with Python from a previous project and wanted to expand my knowledge and horizons when it came to programming languages. Now that a language had been chosen, I needed to move on to creating my GUI for the suite extension with Ruby, but this proved to be a greater challenge than I anticipated.

### GUI creation and Ruby

One this Ruby and Python shared in this project was the use of Java Swing for creating a GUI interface in the Burp Suite. While both languages had their own GUI software because the suite is Java based, I needed to use Java Swing. So, this alone was a challenge on top of the challenge of learning the new language I had chosen. The first part of the project was slow to make progress I had no idea how to implement Java swing code within a Ruby file. I resorted to researching online to try and find a solution to my problem however It was not as easy as I thought it would be. Ruby itself is still a somewhat new language in the programming world so there was not much help online regarding using Java Swing alongside Ruby, although there was plenty of help with Ruby's own GUI programming called Ruby on Rails. My luck changed when I came across a GitHub page which had example templates for creating tabs and sub tabs. Rewriting these templates to fit my own needs, I was able to create a simple GUI in the Suite through Ruby. (KINGSABRI, 2017)

When it came to creating a basic ping function, there were a few resources online that helped point me in the right direction, but I took time to get a basic ping working in the Command Prompt. The problems arose when I tried to implement this function within the Burp Suite. Even though I had a basic tab created I still had no real GUI, input, and output boxes, etc., and when I tried to create the interface, I was not able to run the extension as error after error kept cropping up. I could not understand how to implement Java Swing within Ruby and could find no help online. After some time, I realised I had made no progress and time was still moving on, so I made the decision to switch to Python. While Ruby is a nice language to work with, for the sake of this project I did not have the time to spare to figure out what I needed, while Python had many more resources online to help. So, Ruby had to take the backseat and Python became my main programming Language.

## Burp and Jython Import issues

After finally getting a GUI up and running, with thanks to a tutorial by Laconic Wolf, my next goal was to implement functions. This was slowed down straight away by the Ping function. The actual function could work in the suite but the problem I was encountering was the output of the function. Due to the way the function works and because of Burps own limitations, the Ping function could not output the operating systems own ping output which is what it was doing within the Command Prompt. The main reason for this is because the output box in the GUI could not have strings being incremented as they were executed, instead the output needed to be displayed in one go. So, to overcome this issue, I had to try and recreate the system response for a ping within the function itself and return the four lines to the user in one go. This means that if there is a response the user will receive an output as shown in the below screenshot. (Jake, 2018)

Even after getting this working, I was still encountering further issues when it came to the other functions I had to implement. The next one on the list was the Threaded Port Scanner. Like the Ping function, the Port Scanner with the threader worked fine as I had it written and running in the command prompt. However, when I tried running it within the Suite, a new error popped up saying the module 'queue', which was a module needed for a threader, was not found. For some reason, the Burp Suite could not find the queue module that had been installed to make the threader run. I tried many different solutions, such as adding the full path to the module in the code and after that did not work, I resorted to removing the threader from the code altogether. While this got rid of the error and allowed the function to run, it ran at an excruciatingly slow rate, too slow for it to be worth the time. So, I attempted a different fix, I went online to find the actual source code for the queue module and created a queue module within the directory of the extension and placed this source code into the file. This, for some reason, worked and the Suite would now recognise the import. This allowed the function to use the threader, making the port scan much faster than before.

Another issue I encountered with this was that the output I was receiving from the function was incomplete. The Open and Closed ports results would only show 1 to 3 results, which was incorrect, and I have no clue why it was doing this. To amend this error, I created a list for the open ports to be placed in once they have been scanned and this list was assigned to the results box, allowing me to receive the full output of open Ports. It was at this stage I decided to remove the closed ports results box as I felt it was an unnecessary result for a port scanner, as a pen tester would only typically be interested in the open ports. The final issue I encountered, printing the time taken, was easy to solve. Again, this feature worked with no problems within the Command Prompt while testing, but the suite would not output the datetime output to the textbox as it was not a string, so to fix this I simply changed the output to say `str(total)`. This allowed the suite to output a string to the textbox, giving us our time taken.

The next big issue I encountered with the HTML Scraper was an issue I believe is associated with the Jython jar file. When I made the HTML scraper and ran it externally to the suite, the function worked fine. However, when it was loaded into the suite, an error was returned to me saying "no module named html" in one of the import files. When I went into the appropriate file, I saw that the line the error appeared on used the import module "html.parser". This led me to believe that Jython could not recognise the ".parser" segment of the import, possibly because once it read the '.' it believes that this is the end of the import name. I tried to get past this issue by changing the name of the import and even resorting to using a different parser for the scraper, but nothing worked as this error persisted. (see **Appendix D**)

## Failed Objectives and Main Differences

### Log Function

One of the first functions I had lined up to run in the extension, the Log Function was quickly scrapped as firstly I could not get the suite to write externally to a separate file and secondly, I realised the need for a separate log function was not there as the suite already had a built-in console where I could simply have all function print their output alongside their result box returns.

### HTML Scraper

The HTML scraper was what I settled on to replace the Log function after its removal. As mentioned previously, this is the one function I could not get past the errors of. In the end I could not find a resolution and did not have enough time to find an alternative. While this function does not work within the Suite itself, it does however work very well outside of the Burp Suite, emphasising the fact that the issue was not with the code itself but how Burp and Jython handled the necessary imports to make the function work.

### NSlookup functionality

Originally, I had intended on having an entire tab created separately for an NSlookup function to operate on but, like the scraper, the function would run fine externally but return errors once it was running in the Burp Suite. After not being able to resolve this issue, I decided to remove this function. However, after adding the subdomain crawler, I realised I could potentially implement an NSlookup functionality to the subdomain crawler by having it return the IP address of a discovered subdomain. However, in the end I could not get this feature to work but I believe with more time I could have managed to get this working.

### Ruby & Switch to Python

As stated, before I had originally started this project with the intention of coding it entirely in Ruby. This changed after encountering many issues which essentially stopped all progress with the language much to my disappointment. So, because of this the project changed and I resorted to switching to Python instead.

### GUI Layouts

The final main difference from what I had originally planned on having compared to what I had in the end was the layout of the GUI and the tabs themselves. This changed as the Java Swing within Python was very strict to work with and did not allow for as much flexibility as customisation as I would have liked. Because of the complexities of user input as well, some features had to be removed for example the ping and ping sweep functions originally were supposed to have check boxes for user choices for the output, however these were never added as I did not have the time or knowledge of how to implement such a feature. (See **Appendix A**)

## What I have Learned

### Advanced python knowledge

Heading into this project I already had a basic understanding of Python from using it in a previous project from year 3, so this project offered a great opportunity to expand my knowledge of this language even further, making me more proficient in the use of this language and its syntax which will benefit me going forward.

### Java Swing

Throughout my college course, I have had few encounters with GUI creation, mainly in C# and in Java, where we had to create fully functioning calculator in year 2. At the time I was not confident in my abilities to create a GUI through but after this project, I have gained a better understanding of Java Swing and how it works, leaving me feeling more confident with my abilities to work with Java and Java Swing.

### Burp Suite knowledge

Before doing this project, I had only used the Burp Suite briefly as part of my course but had never delved into the suite and its various functions. Thanks to this project, I now understand the suite and its built-in functions more which, should I aim for a career in Pen testing, will benefit me going forward.

### Basics of Ruby

When I started this project, I had initially chosen Ruby for two reasons. The first was that it worked similarly to Python which meant I would potentially find it easier to learn. But the second reason is because I had never encountered the language before, and I was curious to see what it had to offer. Much of my time at the start of the year and at the start of this project was spent learning this new language and its intricacies. While I could not continue with the language, the brief time I had with it has given me a good start to understanding how to use it, meaning I may be able to expand on my knowledge in the future.

### Different Start

The main change I would make to this project if I had to do it all over again would be to do the whole thing through Java, instead of Python or Ruby. Much of my progress for the project was slowed down by the simple fact that I struggled to create a GUI in python with Java Swing, and that the Java Based Burp Suite struggled to take in many of the necessary python imports for the functions being added.

### Personal Lessons

Aside from language and application learning, I have also learned many other lessons, such as time management, work planning, work ethic and motivation. After each weekly meeting I was able to plan out of what tasks I needed to achieve each week which allowed me to set realistic goals that helped keep me motivated. By setting out these goals, I didn't allow myself to get overwhelmed with work, but instead was able to make steady progress throughout the year alongside the main course.

## Acknowledgements

I would firstly like to thank Richard Butler, my supervisor, who throughout the year gave help and direction in the project when I needed it most. I would also like to thank the other Supervisors, Joseph Kehoe, Paul Barry, and James Egan, for constructive questions that helped sculpt the project going forward. I would also like to thank the other lecturers in the course for giving us whatever time they could give to work on the project as well as the other students for being able to help and support whenever it was needed.



## Bibliography

Jake, 2018. *Burp Extension Python Tutorial*. [Online]

Available at: <https://laconicwolf.com/2018/04/13/burp-extension-python-tutorial/>

[Accessed 4 February 2021].

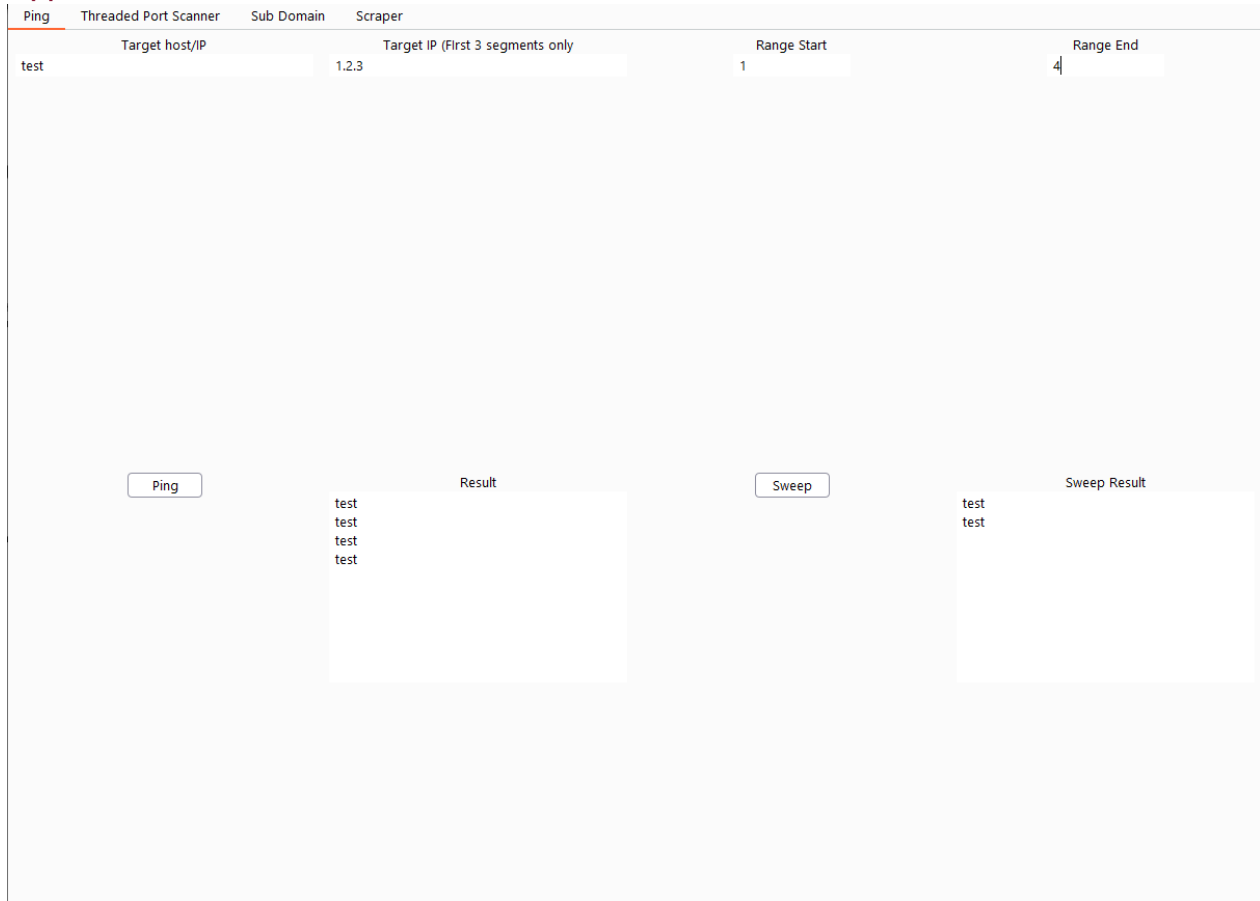
KINGSABRI, 2017. *Burp\_Suite\_Extension\_Ruby*. [Online]

Available at: [https://github.com/KINGSABRI/Burp\\_Suite\\_Extension\\_Ruby](https://github.com/KINGSABRI/Burp_Suite_Extension_Ruby)

[Accessed 11 January 2021].

# Appendix

## Appendix A



Above is the Final GUI and below are the wireframe mock-ups for comparison.

Ping IP

Enter in a host to Ping:

Select what is to be Returned in response:

Up/Down     Time Taken

Bytes Used     Packets Lost/ Recieved

Ping Sweep

Enter in a range of hosts to Ping:

```

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

```

```

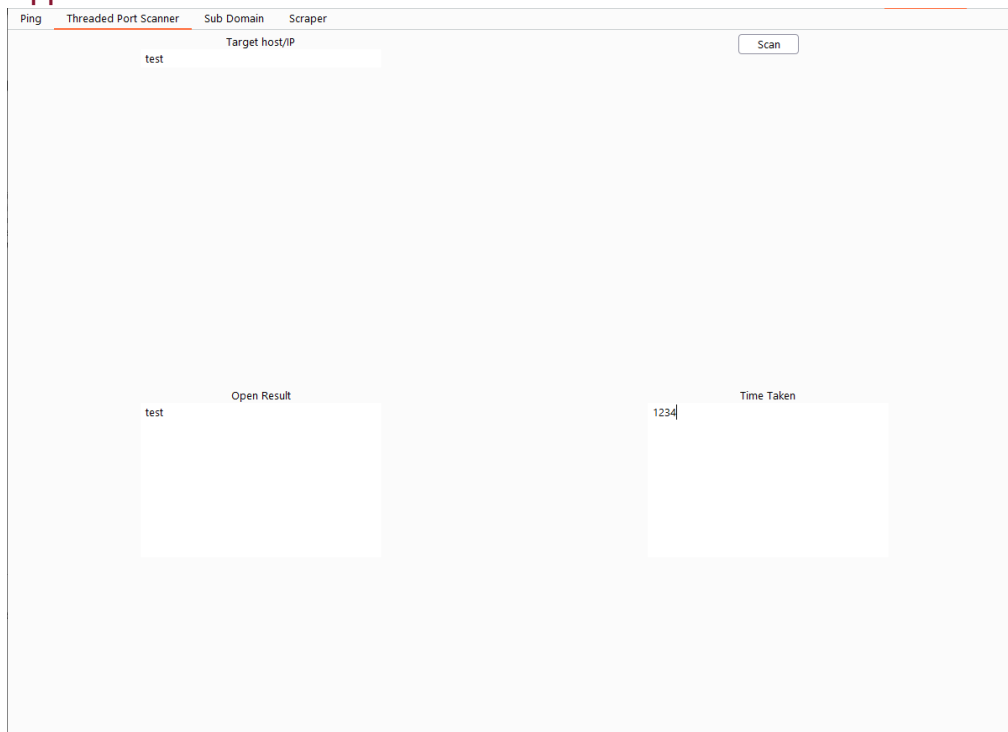
def pingsweep(self, event):
    import subprocess
    import os
    #takes in user input
    target = self.inputAreaB.text
    minR = self.inputAreaB1.text
    maxR = self.inputAreaB2.text
    inactive = 0
    active = 0
    #sets for loop for sweep range
    for n in range(int(minR), int(maxR)):
        #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)

#joining 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

Ping Threaded Port Scanner Sub Domain Scraper

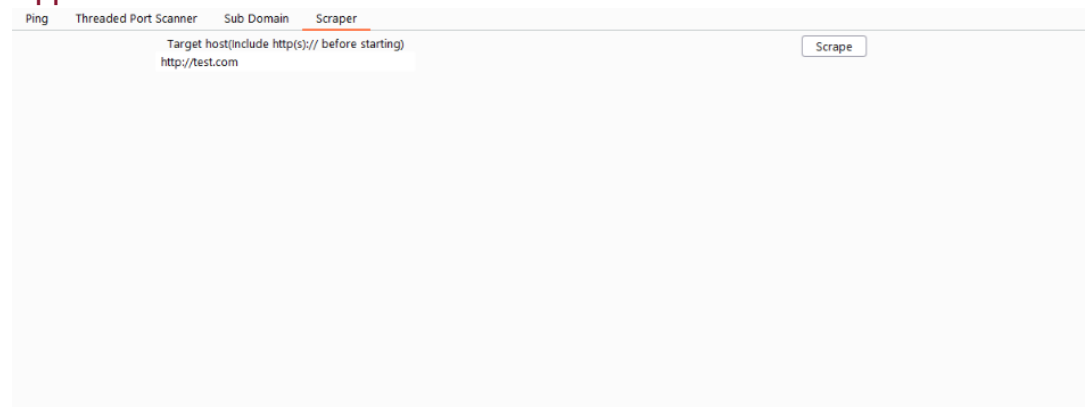
test.com Target host/IP    Select file 1,2 or 3

Result  Time Taken

```
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



Target host(include http(s):// before starting)  
http://test.com

Scrape

Result

test

```
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://{0}".format(target)
    page = requests.get(url)

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

Below is the function output as it appeared in CMD.

```
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>
```