Research Document

Student: Sean O'Connor – C00224424

Supervisor: Richard Butler

Cybercrime & I.T. Security – CW_KCCYB_B

Institute of Technology Carlow

Table of Contents

Introduction

For my year 4 project, I have decided to test the extendibility of the Burp Suite via Python, Java, and Ruby. The goal of this project is to firstly analyse the tools and functions which are available to the user in the Suite before any extending is performed. Next, I will be testing and comparing the three languages, Python Ruby and Java and seeing which is the best language for creating extensions for the Suite before finally analysing, creating, and discussing extra functions I believe should be added to the Suite which are currently not available to the user.

In this research document I will be doing a comparison between the three programming languages, what they have to offer, their styles and syntax, their pros, and cons, etc. Once this comparison is done, I will then be doing an analysis into each of the current functions available to the user before any extending is performed by the user. This analysis will describe what the function does and what tools are available to the user as well as the general layout of the function. After this is done, I will be going on to compare the languages again but this time I will comparing how well they perform in extending the Suite itself. From this comparison I will make a choice as to what language I will be using going forward for the creation and extension of functions I believe should be added to the suite. I will be doing a brief discussion of these new functions before finally concluding on why I believe it is worth taking the time to extend the functionality of the Burp Suite.

Comparing Languages for Burp Extensions

For adding extensions to the Burp Suite, three language options are available within the community version of the Suite: Python, Java, and Ruby. In this section I will go over each of the languages, talking about what they are able to do and comparing them to each other. The first language I will discuss is Python.

Python

In modern day computing there are many different coding languages than can be used by users to write, compile, and run programs. Among these many languages is Python. Python is an object-orientated language which is highly desirable to server-side web and application developers. When compared to languages, Python is generally seen as being like Ruby in that it is easier to use and understand and unlike the other languages and, like Ruby, Python runs on an interpreter system meaning that it does not have a compilation step when code is written, unlike Java. Programmers can write the code and run it immediately, which allows for a very rapid edit-test-debug cycle. Its syntax is very similar to the English language which allows software developers to create programmes with less lines that programmes written in Java.

Python has many positive aspects to it that make it desirable to web application developers such as being very efficient at visualising data for the user or the fact that it supports asynchronous coding meaning that it is easier to solve code problems because it runs multiple units of code separately from the main application allowing the coder to spot issues in the different units easily. However, not everything about Python makes it the best language out there as there are certain aspects which do not make it a desirable language to may programmers such as Python mainly focusing on web application development and running on server-side applications or its speed limitations from being an interpreted programming language. When it comes to programming languages it comes down to the user to decide on what language best suites there needs at a given time, they do not have to be confined to a single language for their whole lives, other languages can be used in different scenarios.

In our scenario, extending the functionality of the Burp Suite, we may use two different languages, which I have already mentioned earlier on if Python does not suite our needs. If I do use Python going forward to create extensions, we must run the Python extensions through an interpreter called Jython. What Jython allows us to do is to run our python language on the Burp Suite as the Suite is Java based. So, to run our code we need the interpreter to read our Python code and make it runnable on the suite itself. In our extension settings, under the Python Environment heading we must configure Burp to look for the Jython file in the right folder. (W3Schools, 2020) (Paul Osborne, Jan 28, 2020) (Paul Krill, Feb 24, 2015) (codeinstitute, n.d.) (portswigger, 2020)

```
Def main():
      print("Hello World")

If __name__ ==  '__main__'
      main()
```

Java

Another programming language, and quite possibly the most widely used language in modern day computing is Java. Java is "a general-purpose, concurrent, strongly typed, class-based object orientated language". Java was originally developed by Sun Microsystems in 1995 and runs an estimated 3 BILLION devices worldwide. When people go to college to start a degree in IT they will, at some stage in their education come across and use the Java language. Whether they are creating web/phone applications, setting up web servers or even creating games, Java is used at least ONCE by most people in the IT sector. An example of it being used in everyday use would be the popular video game Minecraft, which when originally released on the PC was built and run entirely on Java which was considered very strange at the time.

The reason Java is so popular is for a multitude of reasons. First is that it is a **Write Once, Run Anywhere** language, meaning that a programme can be created in Java and run across multiple different platforms unlike languages like Python which can't be run properly on Android devices and need to be packaged differently. Java programmes can run on any platform if they have a Java Virtual Environment to interpret the java byte code. Java is also believed to be very easy to learn how to use compile and debug. It is often used as an introductory language for people new to software development. Java is also considered to be very secure as it has a security manager which defines the access of classes. Not everything with the language is good, however. Java devours computer memory and runs significantly slower than other languages such as C++ or Python.

Another thing that can be seen as both a pro AND a con for Java is the fact that it is Object Orientated which allows you to form normal, standard programs and re-use code but getting used to Object Orientated Software Development can take time for user new to programming. Out of the 3 languages I will be testing, Java is the only one I cannot use in the Brackets Text Editor. The reason for this is because Java code must be properly compiled before being executed and Brackets does not come with a dedicated compiler. So, for this section of the project I will be using the Eclipse IDE. (W3Schools, 2020) (TutorialsPoint, 2020) (CodeInstitute, 2020) (data-flair, 2020)

```
Class A
{
      Public static void main(String args[])
      {
            System.out.println("Hello World");
      }
}
```

Ruby

The final language we can use to implement extensions within the Burp Suite is Ruby. Ruby is pure object-orientated language developed in 1993. Made by Yukihiro "Matz" Matsumoto, Ruby was created with the intention of being a language that would balance both imperative and functional programming. To do this Matsumoto took inspiration from his favourite programming languages, Perl, Smalltalk, Eiffel, Ada and Lisp and mixed up all of what he considered to be their best parts It is considered to be similar to Python in the sense that it is a very similar server-side scripting language but also is similar to the likes of C or Java as they have similar syntax, which can make it easier for programmers new to Ruby to learn the language. One huge benefit is that the language is supported across nearly all platforms, such as Windows, Linux and Mac OS. Another benefit is that ruby also has a large library of built-in functions to which the user can use in their Ruby scripts, which can increase the workflow while using this language. Like Python, Ruby must also have an interpreter available for its coded extensions to be able to run properly in the Burp Suite again, because Burp is Java based. To do this we must go into the extensions Options in the Suite and look under the heading Ruby Environment. Under there we must select the right directory where the Ruby interpreter for Java, JRuby is available. (Ruby, 2020) (Cynixit, n.d.) (TutorialsPoint, 2020) (GeeksforGeeks, n.d.)

```
Class HelloWorld
      Puts "Hello World!"
End
```

Comparison Table:

	Python	Java	Ruby
Server-Side Programming	Yes	Yes	Yes
Object Orientated	No	Yes	Yes
Multi- Platform Support	Yes	Yes	Yes
Built-In Functions	Yes	Yes	Yes
Code Must Be Compiled	No	Yes	No
Code is Interpreted	Yes	No	Yes

Analysis of the Burp Suite and It's Functions

In this section I will be giving an overview of the Burp Suite itself, talking about its origins and functionality before moving onto the functions that are currently available for use within the Suite before adding any extensions.

Burp Suite

The Burp Suite is an open-source Java based framework which offers a range of tools to a user for identifying vulnerabilities and attack vectors in a web application. The Suite was developed by the company Portswigger and is known to be the favoured tool among professional IT Security testers and hackers because of its simple and easy-to-use interface. Below I will discuss each of the functions available in the community version of Burp in detail, discussing each features functionality, layout, etc.

Tabs & Features

Embedded Browser:

The burp Suite's first main feature you will utilise is the embedded browser. The embedded browser supplied by the suite allows users to create HTTP traffic on their network so that they can intercept and analyse this traffic without interfering with their actual browser. If users do not wish to use this feature, they can also configure their browser settings to allow the Burp Suite Certificate so that Burp can intercept real traffic. This function is found under the Proxy tab within the suite. When you first open this tab, you will be greeted with 2 options: to open the Burp embedded browser Chromium or open a different browser. Using the embedded browser is better when you don't want to go through the configuration of t your browser settings to allow for the Burp Suite Certificate, it means you can begin testing almost immediately. When you open the embedded browser, a separate window will open with a blank page and a new tab will open upon within the suite itself called Intercept. What this allows you to do is intercept HTTP traffic that is sent within the embedded browser. I will discuss what is displayed to the user in the next section, Proxy. (Portswigger, 2020)

Proxy:

This intercepting proxy is provided by the Burp Suite and will allow users to intercept HTTP traffic within the browser and even modify the contents of the HTTP traffic, essentially making the user the man-in-the-middle. As was discussed in the embedded browser section previously, we can open an embedded browser from this Proxy tab. Opening the embedded browser will open a Chromium tab and this will open 4 new sub tabs called **Intercept**, **HTTP history, WebSocket History and Options.**

In the **Intercept** tab, you will see 4 sub-tabs: **Raw, Headers and Hex**. The **Raw** tab will display the HTTP information such as the Host and the User agent in plain text. The user can edit this plain text to whatever they want and have the option to perform multiple actions on this raw data from a drop-down menu, allowing the user to send the data to the different functions within the suite. The **Headers** tab will display to the user the HTTP Request Header information, telling them if it was a GET or POSR, what the host was etc. The user can add, remove, or change the position of information within this tab. Finally, in the **Hex** tab, as the name suggest, the user will see the information displayed in hexadecimal.

Next, we have the **HTTP History** tab which will display to the user a list of requests that have been detected within the embedded browser, showing their hosts, their method and their URLS. IF you click on one of these items on the list a new menu system will open below the list which will allow us to view the same elements that we could see in the Intercept tab; Raw, Headers and Hex. Above the item list is a filter tab, which when the user clicks on it, the user will have a range of filters they can apply to the list so that they ca narrow down what they may be searching for.

The next tab we have available to us in the Intercepts tab is the **WebSocket History** tab which will display a table that tells us a history of WebSocket connections, displaying their URLS, the direction of the messages, response length etc. Like the HTTP History tab, we can click on any item of the List and view their contents on the lower pane. (Portswigger, 2020)

Intruder:

The Intruder tool is used to test the success/failure rate in an input point. This is done by running a set of values through said access point and noting the output generated. This tool may also be used for dictionary attacks or Brute Force attacks. When you open the Intruder tab you will have access to 4 tabs, **Target, Positions, Payloads and Options.** The **Target** tab is for configuring two details needed for starting an intruder attack and has a simple layout, consisting of two input fields, Host and Port, a checkbox for deciding whether HTTPS is used and the button to start the attack.

The next tab **Positions** contains a large field in which the user can position their payloads to be inserted into the base request. It also has a drop-down menu for the type of attack the user wishes to perform, of which four are available, Sniper, Battering Ram, Pitchfork and Cluster Bomb. Once the user has selected what attack they wish to use and where they wish for it to be inserted, they will move onto the Payloads tab.

The **Payloads** tab will allow the user to configure the payloads you wish to use for your attack. The tab is divided up into 4 segments the first of which being the Payloads Sets. Here the user can define one or more sets of payloads and which type of payload is needed for each payload set. The next section, Payload Options allows you to configure a list of strings which can be used as payloads, allowing you to paste or load in strings, remove a string or clear the entire section. You can also add another item underneath. The 3rd section, Payload processing allows the user to add, edit or remove rules that have been defined by the user that are performed on each payload before use. Finally, we have the Payload Encoding section, which allows the user to URL-encode certain characters in the final payload for safe HTTP Request transmission.

The final tab, like the Proxy tab is for general options for the Intruder. This tab is divided up into 7 segments; Request Headers which allows user to control whether the intruder will update the configured request headers during the attacks, Request Engine which will control the engine that will be used to create the HTTP requests when performing the attacks, Attack Results which will control what information is recorded in the attack results, Grep – Match which allows the user to flag result items containing certain specified expressions in the box, Extract which will extract any useful information specified in the box from the responses into the results table, Payloads which will flag result items containing reflections of the submitted payload and finally Redirections which control how Burp handles web redirections when attacks are being performed, as in whether or not the redirections are followed. (Portswigger, 2020)

Repeater:

When using the other functions, you may see the option to "Send to Repeater" in the context menu. Is to send the output of the other functions to this tab and then manipulate any part of the HTTP request headers and see what each of these responses looks like. On this tab you can also send a request directly to your target, by pressing the send button. This will display a prompt window asking the user to input a target host, a target port and check a checkbox if they wish to use HTTPS. Once this is configured and sent, the user will see the requests being displayed in the left-hand box either in raw format or in hexadecimal, and they will also have a dropdown menu allowing them to perform a range of actions with the data. The box on the right-hand side will display the Reponses but in raw format only and no extra actions can be taken with this. (Portswigger, 2020)

Sequencer:

The burp Sequencer tool is used for taking a sample of data that has been acquired by the user and testing the randomness of this data. This is used to test a web applications session tokens, CSRF tokens or any other important data items that are unpredictable. When the user clicks onto the Sequencer Tab, they will have access to three sub-tabs, **Live Capture**, **Manual Load** and **Analysis Options.** The **Live Capture** tab is divided into 3 parts the first of which is Selecting Live Capture Request, which allows the user to send request from the sequencer tool to the other tools in the suite to create and configure a live capture. The user selects a request they wish to use, configure the Options below and then start the Live Capture. The next section is the Token Location within response which allows the user to select the location in which the token appears in the response, with the user being able to choose from a cookie, a form field, and a custom location. Finally, we have the Live Capture Options which allows the user to configure the settings for the HTTP Request Engine when performing a Live Capture.

Next, we have the **Manual Load** tab which allows a user to load the sequencer with a sample of tokens that the user has already gotten and then perform a statistical analysis on that sample. The page allows you to either paste in or load a sample and then analyse that sample.

Finally, we have the **Analysis Options** tab which allows the user to configure two options: Token Handling and Token Analysis. The Token handling options allows users to control how tokens are handled during the analysis, whether tokens are padded and if so, what are they padded with. The Token Analysis Option controls the types of analysis that is performed at either the character level or at the bit level. (Portswigger, 2020)

Decoder:

The Decoder Tab unlike the others, has no sub-tabs or options, consisting of a simple interface containing a large text box and options on the side to Decode, Encode, Has or Smart Decode. To use the decode the user may simply copy and paste the data they wish to decode or select the option to send to the decoder tab from a tab where the original data has been output to the user. When they user selects the Decode, Encode or the Hash functions, they will see a drop-down menu offering a range of functions associated with the menu they have selected, with their choices being, Plaintext, URL, HTML, Base64, ASCII Hex, Hex, Binary, Octal or Gzip. The Hash function will list a large range of different hashes a user may wish to use. The smart Decode option is a tool offered by Burp which will try to intelligently decode the data it reads in and output the correct information, an option for people who do not know what they need to decode the data as. (Portswigger, 2020)

Comparer:

The Comparer, like the Decoder, has no sub-tab or special options and again has a very simple layout, interface, and functionality. As the name suggests, this tool's function is to compare to items of data at a world level and/or a byte level. Like the decoder, the user can either copy and paste data into either one of the boxes or send the data directly from wherever the data was output to the user. Once the data has been selected, the user can then select if they want the data to be compared at the word level or at the Byte level and once the comparison is started a new window will pop up containing the results of the comparison. (Portswigger, 2020)

Extender:

The final main function of the Burp Suite which user have access to, and the one I am most interested in, is the extender. The extender, as the name suggests allows the user to extend the current functionalities of the Burp Suite by loading 3rd party code to the suite. The user may either write their own code or get extensions through the dedicated BApp Store, where Burp users can upload their own functions for the users to install for themselves. When the user opens the extender tab, they will have access to four sub-tabs; **Extensions**, **BApp Store**, **APIs,** and **Options.** The **Extensions** tab as the name suggests is where users can load up the extensions they have install from the store or created themselves. The user will see a large window with options to add or remove functions and change their load order. Once a file is loaded successfully, they will see it in the input box with a check box checked off beside it, showing that it is set to be loaded into the suite. When the user selects the checkbox to load the extension a popup window showing the progress of the loading will be showed along with and Output tab and errors tab. Once the extension is loaded the user may close this pop-up window and continue to use the Suite. They will also see, below this box, another tab for details, output, and errors. These three fields are there for the user to see details about the extensions they have loaded to the suite such as they extension type, the filename/directory etc. If there are any errors with loading the extension, these will be displayed in the output and errors tab for the user's convenience.

The next tab, **BApp Store,** is where the user can select and install a huge range of various functions created and uploaded by users of the Burp Suite. From this page the user can sort the list of functions by name, whether they have a function installed, the functions rating from 1 to 5 stars, the functions popularity, when it was last updated and finally any other details such as what Burp Suite Version is needed for the function. When a user selects the function that they want, a side menu will be open which will show the function in detail along with the option to install the function to their suite. Below this list is two buttons which will allow a user to either refresh the list of functions or choose to manually install the user's own functions which will bring up a new page for installing.

The next tab, **APIs**, is a full listing which shows the available APIs that a user might need for created Burp Extensions. The tab consists of the full list which will only show APIs that are available with the current version of Burp a user has running on their machine. Once a user selects an API a second window on the side will show the contents of that file which Is available for users to observe and even copy from. The second window also has a search bar so users can quickly navigate an API to find something specific. Once the user has found what they are looking for, they can use the 'Save Interface Files' or 'Save Javadoc Files' to save copies of these files locally on their machine for when they are developing API's.

The final tab in the extenders tab is the **Options** tab which like other options tabs in other functions, is split up into 4 sections, Settings, which control how the Burp Suite handles extensions on start-up, Java Environment, which allows the user to select an environment for executing any extensions that are written in Java, Python Environment, which like the Java setting, allows you to configure an environment for executing Python files in the Burp Suite and finally the Ruby Environment which like the previous two allows us to configure an environment for executing Ruby extensions. (Portswigger, 2020)

What Features will be added to the Suite?

In this section I will discuss the range of functions I will be adding to the Burp Suite and what languages they are going to be added with. First, I will try adding 1 common function to the Burp Suite across 3 different languages, Python, Java, and Ruby. After this is done and a comparison between the three is performed, I will choose a language to go forward with and then finally choose and analyse more functions that are either available in the BApp store or functions I wish to add.

Python

So, to start testing on what language I would be using, I decided to implement a simple Hello World extension to the Burp Suite. This extension would not have its own tab, and neither would it have any real functionality. Its only purpose is to test the syntax and ease of use of the languages when it comes to adding extensions to the Burp Suite. To start off I decided to try Python, as I am familiar with the Python language and was interested in seeing how Python works with Burp. The first step for setting up the Python extension was to go to the Extender tab and go to the options sub-tab. Here we had to go to the Python environment section and choose the location of the Python standalone Jar file so that our Python code be interpreted by the Suite. Once this was selected, we then had to create our Python file. I decided to use the Brackets text editor to create this as I had used this of a previous project and enjoy its easy-to-use interface. Before a new file is created, we should make a folder on our computer specifically for our extensions. It is in this folder where we have our extension we have created and our jar files. Now that we have our extensions folder created and our Jar file located, we crate our new .py file. To start the file properly we use the section of code provided to us by Portswigger, the group who created the Burp Suite. For Python, this consists of 5 lines of code which are needed to allow our file to connect to the suite. Below is the snippet of code form the Portswigger site.

```
from burp import IBurpExtender
      class BurpExtender(IBurpExtender):
            def registerExtenderCallbacks( self, callbacks):
            # your extension code here
            return
```

Once this code has been copied into your file you can then add in your own code into the section marked. For Python I was able to create an actual tab in the Proxy tab of the suite as I had access to a sample of code online from laconic Wolf's python burp extension tutorial. Since using this code, the domain where this tutorial and its code is no longer available, but I still had the code from previous testing. So now that we have our code, we save our file and go back to the Extender tab within the Suite. In this tab we stay in the Extensions sub-tab where we can see where we can add or remove extensions. To add our Hello World Extension, we clock add which brings up a new window for configuring the extension file. We select the desired language from the drop-down menu (the default being Java) and once we have Python selected, we select the file we need. Once our file is chosen, we select next which will bring us to a new window where we see the progress of the extension being loaded. If loaded successfully, the loader will display a message saying the loading was successful. If our file has any output, such as printed lines it will be printed in the Output tab on this loader window AND in the Extender tab. If the load was unsuccessful then we will receive an error message and the Errors tab will be populated with the necessary errors. But because ours was successful, we can go to the Proxy tab and open an embedded browser. If we minimise this browser and go back to the suite, we will see a Hello World tab has been created with no functionality, proving it to be successful. (Portswigger, 2020) (Jython, 2020) (laconicwolf, n.d.)

Ruby

The next language on my list to test was Ruby. Ruby for me was uncharted territory as I had never encountered or even heard of the language until starting this project. To start the Ruby extension, we must follow similar steps to the Python extension. First, we must find JRuby which is a Java interpreter for Ruby. Once we have downloaded the JRuby Complete Jar file we must again make sure it has either been saved or moved to our burpExtensions folder. Then we must configure Burp to look for this file in the location for when we need a Ruby environment created, following the same steps as we did in the Python segment. Once the JRuby location has been configured we can now create our Ruby file. We will be using Brackets for this again and saving it in the same location as the Python file. Like the Python extension, Portswigger offers a snippet of example code for user to use when they are creating their Ruby extensions.

```
require 'java'
java_import 'burp.IBurpExtender'

class BurpExtender
      include IBurpExtender

      def registerExtenderCallbacks(callbacks)
            # your extension code here
      end
end
```

As we can see from this small snippet of code, Ruby does not stray too far from the syntax and format of Python making it easy to read for a new user like myself. What changes from Python to Ruby is how we import the necessary imports, instead of saying `from burp import x, we` say `java import 'x'`, with x representing the name of the import. Also, instead of needed ':' at the end of each 'def' we can just go to a new line and tab in. The final main change is the use of the 'end' keyword which is needed at the end of each method and class in Ruby whereas Python does not need any keyword to end a class or function. Now that this start sample has been created, we can setup our Ruby file to output Hello World into the Suite's extension output box to prove that we can extend the Suite through Ruby. To do this we add the following single line:

```
puts "Hello World"
```

What this allows us to do is print a line which says, "Hello World". For this example, we will not be setting up a new sub-tab in the Proxy tab, as we are simply doing this for testing purposes and as Ruby is new to me, I will need more time to fully learn and understand how to add a new tab through Ruby. But once we load our Ruby file through the same way as we did for the Python file, we will see that the extension is loaded successfully and "Hello World is outputted to the Suites output box, meaning that Ruby can be used to extend the Suite. (Learn.Co, n.d.) (Portswigger, 2020) (JRuby, 2020)

Java

Finally, we get on to the last language on the list and the one I have been looking forward to the least, Java. The reason Java was tested last was for 2 main reason; Firstly, I have already had previous experience with Java from the first two years of my college course, so I did not need to spend any time learning how to write in the language. The second reason is because I personally do not like using Java as a programming language. It is very easy for a programme to break and can sometimes be hard to fix an error that occurs in the programme. And that is exactly what happened in this testing phase of the languages. According to the same brief of information from Portswigger about creating a burp extension, to create a Java based extension you must save a list of APIs from the API list in the Extender tab in the same package as the file you wish to create. However, when I tried to do this myself, I encountered several errors which I discovered to be due to my IDE creating a second package when I saved the APIs. Because of this the package names in the APIs did not match the main package which caused errors. So, to fix this I had to delete the original package I had created for these APIs and rename the new package to the right name. Even after doing this however some important APIs were still throwing up error messages as they were looking for a package that was not there. If the APIs had worked normally, we would've have followed the same steps as the previous segments, by using the code provided by Portswigger to create a basic hello world extension with Java. Below is the segment of code needed. (Portswigger, 2020)

```
package burp;
public class BurpExtender implements IBurpExtender
{
      public void
registerExtenderCallbacks(IBurpExtenderCallbacks callbacks)
      {// your extension code here}
}
```

So, going forward I will be using the Ruby language to extend any future functions. The reasoning for this is because I found the Ruby language to be interesting in the short time, I've been using it and would be keen to learn how to write and understand the language full. I also feel that as a new language that I will have to teach myself, it would be something of a challenge for myself to see if I can learn a brand-new language from scratch.

Features Planned

I plan on implementing 4 features going forward with this project:

Basic Ping

The first function I would like to add is as simple if a function as you can get, a basic Ping. What this will allow the user to do is input a target host into an input box, choose the number of responses they wish to get and the length of time it will take. The output will be displayed just below the input box for the user to observe.

Ping Sweep

This next feature I wish to add them to the Burp Suite is the Ping Sweep. When we use a normal ping, we ping a single host or address and get only that response back. If we have a range of IP's that we wish to see are up or down, a ping sweep will help do this. A ping sweep will take in a range scope of addresses and ping each of them to receive a response from all of them in rapid succession. This means that we do not have to ping multiple separate IPs with separate pings, thus increasing the speed of testing. How it works is, the user will input a target scope of addresses. The ping sweep will then take in this range a ping each of them one after the other, receiving a response from each as it goes along. One good thing about this method is that it saves the user time, however it also means that results may be not as detailed as to speed things up we would be only receiving one response before moving onto the next host, whereas a normal ping uses 4 responses to get an accurate result.

Threaded Port Scan

A port scan is a useful function to have. It lets us know whether a target port is open or closed, up or down. The issue with port scanning is that it takes a long time depending on your target. That's where Threaded Port Scanning comes into play. Where a regular Port Scan will just scan one port and wait for a response before moving onto the next, a threaded port scan scans one port and while waiting for the response will carry on to the next and so on. The output of this scan will be displayed below the user's input. Using a threaded port scan over a normal one greatly increases the speed and efficiency of scanning, but one downside is that it is a lot more demanding on hardware to perform these multiple scans at the same time.

Log HTTP Requests

This next feature is a simple one but a necessary feature I Believe. Each function currently available in the suite has some sort of output available for the user to edit and copy/paste somewhere else. However, if someone wishes to perform multiple functions and get multiple results, they must go to each function output one by one to get their results. This HTTP Log function will allow all results from every tab to be output to a separate tab so that they are all available in the one output box, allowing the user to copy and paste all the info that they need into a text file or research document once they are finished all of their scans quickly and easily. This function was inspired by a function that is available on the BApp store, however the App Store version Writes the data to an SQLite database in the home directory only. I wish to change this to write to an output box within the suite and to a text file of the user's choice so that the information is easier to access.

When it came around to creating the extensions for the suite, I decided to move away from this function and focus on two others. The reason for this change is one, because I realised that this type of tool is not needed for the project I was creating as there was a built-in console in the Burp Suite where output could be printed and taken from each of the functions. I also found out that the Suite allows writing to external files but only in the Professional Edition of the Burp Suite.

Why should we extend the Burp Suite?

The Burp Suite offers a wide range of functions available to the user, even in the Community version. These functions work extremely well, will no bugs, a good help section for new users, an easy-to-use interface as well as many other things that make the Suite a good tool to use. But while these functions have all these pros, there are still many functions a user may wish to use that are not available in the base version.

While other suites may not offer the ability to add functions, Burp offers a very easy way of implementing any custom functions the user may want or need, in 3 very popular languages. With access to an entire App Store, users already have a huge range of choices they can add should they desire to do so. And if a function they wish to have is not available on this app store, the user can simply create the function themselves with help from the Portswigger site which offers sample code and other tips for creating your own extensions. It is a very welcoming and friendly experience allowing users of a wide range of levels of experience to create the Suite that they wish to use rather than being stuck with the basic version. Having Burp being customisable makes it extremely desirable to pen testers and other users as they can fine tune it to suit their own needs.

Bibliography

CodeInstitute, 2020. *What is Java and why is it important?.* [Online]
Available at: https://codeinstitute.net/blog/what-is-java/
[Accessed November 2020].

codeinstitute, n.d. *What is Python, and how is it used by today's coders?.* [Online]
Available at: https://codeinstitute.net/blog/python-used-todays-coders/
[Accessed November 2020].

Cynixit, S., n.d. *Ruby Programming Language (Introduction) For Beginners.* [Online]
Available at: https://medium.com/@SravanCynixit/ruby-programming-language-introduction-for-beginners-e9e300fb6654
[Accessed Nov 2020].

data-flair, 2020. *Pros and Cons of Java`| Advantages and Disadvantages of Java.* [Online]
Available at: https://data-flair.training/blogs/pros-and-cons-of-java/
[Accessed November 2020].

GeeksforGeeks, n.d. *Ruby Programming Language.* [Online]
Available at: https://www.geeksforgeeks.org/ruby-programming-language/
[Accessed November 2020].

https://www.tutorialspoint.com/java/java_overview.htm, 2020. *Java - Overview.* [Online]
Available at: https://www.tutorialspoint.com/java/java_overview.htm
[Accessed November 2020].

JRuby, 2020. *JRuby Downloads.* [Online]
Available at: https://www.jruby.org/download
[Accessed November 2020].

Jython, 2020. *What is Jython?.* [Online]
Available at: jython.org
[Accessed November 2020].

laconicwolf, n.d. *Burp Extension Python Tutorial.* [Online]
Available at: https://laconicwolf.com/2018/04/13/burp-extension-python-tutorial/
[Accessed November 2020].

Learn.Co, n.d. *Hello World Ruby.* [Online]
Available at: https://learn.co/lessons/hello-world-ruby
[Accessed November 2020].

Paul Krill, E. I., Feb 24, 2015. *A developer's guide to the pros and cons of Python.* [Online]
Available at: https://www.infoworld.com/article/2887974/a-developer-s-guide-to-the-pro-s-and-con-s-of-python.html
[Accessed November 2020].

Paul Osborne, C. T. O. C. T. L., Jan 28, 2020. *Pros & Cons of Using Python For Web Development.* [Online]
Available at: https://thepythonguru.com/pros-cons-of-using-python-for-web-development/
[Accessed November 2020].

Portswigger, 2020. *Burp Comparer.* [Online]
Available at: https://portswigger.net/burp/documentation/desktop/tools/comparer
[Accessed November 2020].

Portswigger, 2020. *Burp Decoder.* [Online]
Available at: https://portswigger.net/burp/documentation/desktop/tools/decoder
[Accessed November 2020].

Portswigger, 2020. *Burp Extender.* [Online]
Available at: https://portswigger.net/burp/documentation/desktop/tools/extender
[Accessed November 2020].

portswigger, 2020. *Burp Extender, Python Environment.* [Online]
Available at: https://portswigger.net/burp/documentation/desktop/tools/extender
[Accessed November 2020].

Portswigger, 2020. *Burp Sequencer.* [Online]
Available at: https://portswigger.net/burp/documentation/desktop/tools/sequencer
[Accessed November 2020].

Portswigger, 2020. *Embedded Browser.* [Online]
Available at: https://portswigger.net/burp/documentation/desktop/functions/embedded-browser#:~:text=To%20use%20the%20embedded%20browser,automatically%20be%20proxied%20through%20Burp.
[Accessed November 2020].

Portswigger, 2020. *Using Burp Intruder.* [Online]
Available at: https://portswigger.net/burp/documentation/desktop/tools/intruder/using
[Accessed November 2020].

Portswigger, 2020. *Using Burp Proxy.* [Online]
Available at: https://portswigger.net/burp/documentation/desktop/tools/proxy/using
[Accessed November 2020].

Portswigger, 2020. *Using Burp Repeater.* [Online]
Available at: https://portswigger.net/burp/documentation/desktop/tools/intruder/using
[Accessed November 2020].

Portswigger, 2020. *Writing your first Burp Suite Extension.* [Online]
Available at: https://portswigger.net/burp/extender/writing-your-first-burp-suite-extension
[Accessed November 2020].

Ruby, 2020. *About Ruby.* [Online]
Available at: https://www.ruby-lang.org/en/about/
[Accessed November 2020].

TutorialsPoint, 2020. *Java - Overview.* [Online]
Available at: https://www.tutorialspoint.com/java/java_overview.htm
[Accessed November 2020].

TutorialsPoint, 2020. *Ruby - Overview.* [Online]
Available at: https://www.tutorialspoint.com/ruby/ruby_overview.htm
[Accessed November 2020].

W3Schools, 2020. *Java Introduction.* [Online]
Available at: https://www.w3schools.com/java/java_intro.asp
[Accessed November 2020].

W3Schools, 2020. *Python Introduction.* [Online]
Available at: https://www.w3schools.com/python/python_intro.asp
[Accessed 11 2020].