LONS-C2: Desarrollo de una herramienta de Command and Control para pentesting

Mario González Castaño

Resumen– Con la creciente importancia de la ciberseguridad tanto para individuos como empresas, el trabajo de verificar la seguridad de diferentes sistemas por parte de los pentesters es indispensable. Los Command and Control (C2) son herramientas que facilitan la gestión de terminales y automatización de procesos en auditorías internas, por ello, se ha creado LONS-C2, una herramienta C2 multiplataforma desarrollada en Python para la automatización de acciones en pentesting, permitiendo centralizar sesiones en una única terminal. La herramienta ofrece funcionalidades de ejecución de comandos, transferencia de archivos entre servidor y clientes, extracción de capturas de pantalla e información de red. Se ha testeado LONS-C2 en un entorno que simularía el real, obteniendo los resultados esperados de cada una de las funcionalidades implementadas. Con el fin de ofrecer el proyecto en formato open-source a la comunidad, este ha sido publicado y documentado a través de GitHub, permitiendo a la comunidad el uso libre de la herramienta e incentivando su mejora a través de la funcionalidades que ofrece esta plataforma.

Palabras clave– Ciberseguridad, pentesting, command and control, sockets, open-source.

Abstract– With the growing importance of cybersecurity for both individuals and companies, the task of verifying the security of different systems by pentesters is essential. Command and Control (C2) tools are designed to facilitate terminal management and process automation in internal audits. That's why LONS-C2, a cross-platform C2 Python tool, has been created for automating actions in pentesting, allowing for the centralization of sessions in a single terminal. The tool offers functionalities such as command execution, file transfer between server and clients, capturing screenshots and network information. LONS-C2 has been tested in an environment that simulates the real one, achieving the expected results for each of the implemented functionalities. In order to offer the project in an open-source format to the community, it has been published and documented through GitHub, enabling the community to freely use the tool and encouraging its improvement through the functionalities provided by this platform.

Keywords– Cybersecurity, pentesting, command and control, sockets, open-source

1 Introducción

Hoy en día tanto empresas como individuos disponen de gran cantidad de datos de información personal expuesta en Internet, como pueden ser cuentas de banco, documentos de identificación, contactos, mensajes, entre otras. Es crucial preservar y proteger estos datos, debido a la gran cantidad de ciberamenazas que existen actualmente, y que van creciendo de forma drástica cada año[1].

Las empresas hace tiempo que se han dado cuenta de que invertir en ciberseguridad es una necesidad. Los riesgos a sufrir una brecha de datos, un ransomware u otro tipo de ciberataque son cada vez mayores y las posibles pérdidas asociadas aún más. Para cubrir esta demanda, un perfil profesional que cada vez está cogiendo más importancia es el de "pentester", acrónimo proveniente de la combinación de las palabras "penetration" y "tester", que es un profesional que se dedica a realizar auditorías de seguridad a todos aquellos activos como aplicaciones, servicios, dispositivos que sean susceptibles a ser atacados, para identificar posibles vulnerabilidades, malas prácticas, configuraciones erróneas y en resumen, dar soluciones para mejorar su seguridad.

Una situación cotidiana para un pentester al realizar una auditoría de seguridad, en el caso de que sea a una red interna, es el manejo de diferentes sesiones en distintos dispositivos para automatizar el lanzamiento de exploits, ejecución de comandos, traspaso de ficheros y otro tipo de acciones. En este tipo de situaciones, el pentester se tiene que en-

• E-mail de contacto: mario.gonzalezca@autonoma.cat
• Mención realizada: Tecnologias de la Información
• Treball tutoritzat per: Ramon Marti Escale
• Curs 2022/23

frentar a una red interna, con un conjunto de ordenadores y otros dispositivos como impresoras, switches, entre otros. En el caso de que la red sea pequeña, gestionar las distintas sesiones en diferentes terminales puede ser posible, pero claramente es una solución poco escalable.

La herramienta que se pretende crear en este proyecto es un Command and Control (C2)[2], una infraestructura que permite controlar un conjunto de máquinas desde un servidor central, con el objetivo de ofrecer una alternativa simple para automatizar y facilitar la realización este tipo de tareas. La herramienta se divide en dos partes: cliente y servidor. El servidor se ejecuta en aquel dispositivo desde donde el auditor quiera centralizar las sesiones a las diferentes máquinas, y el cliente, donde el auditor ejecuta el programa para crear la conexión hasta el servidor central, dando "acceso" a que la máquina sea controlada.

Este proyecto muestra el proceso de creación de la herramienta Command and Control (C2) LONS-C2[3], basada en sockets TCP, desarrollada en Python, para automatización y gestión de sesiones en auditorías de seguridad.

2 Objetivos

El objetivo principal y prioritario de este proyecto será el desarrollo de una herramienta C2 en Python, que facilite al pentester la gestión de sesiones en una auditoría.

Para cumplir este objetivo, se han planteado los siguientes objetivos, ordenados por prioridad:

- Análisis de requerimientos y diseños necesarios para poder llevar a cabo la herramienta. Entre ellos, los diagramas de clase y secuencia.
- Implementación del servidor centralizador, que disponga de una interfaz de usuario mediante por la cual el pentester utilizará para comunicarse con los clientes.
- Implementación del cliente, que permitirá al pentester conectar hacia el servidor aquellas máquinas que requieran del control del servidor.
- Creación del repositorio público que alojará la herramienta y documentación, en formato open-source, para que pueda ser usada por todo aquel que la necesite.

3 Estado del arte

Un Command and Control (C2)[4] es una infraestructura que permite el control de un conjunto de máquinas desde un servidor central. Existen un gran número de frameworks de utilizados en auditorías de Red Team. Estos se dividen entre frameworks open-source y frameworks comerciales.

En cuanto a los open-source, estos son creados por empresas o grupos de desarrolladores con el objetivo de ofrecer una herramienta útil para la comunidad. Generalmente, son proyectos sin ánimo de lucro[5]. Algunos ejemplos de este tipo de frameworks son:

- Sliver[6]: Sliver es un framework de Command and Control, capaz de generar ejecutables para conectar a las máquinas "cliente" para cualquier tipo de arquitectura existente, y controlar las sesiones de manera segura desde un servidor central. Permite la colaboración entre múltiples pentesters de forma simultánea.
- Meterpreter[7]: Meterpreter es un framework C2 conocido por estar integrado en el framework de seguridad ofensiva Metasploit, incluido en la distribución Kali Linux[8]. Meterpreter permite realizar una gestión ordenada de múltiples sesiones en diferentes máquinas, y dispone de una gran cantidad de funciones implementadas para la recolección de información del host.
- Gcat[9]: Gcat es una implementación de C2 que utiliza Gmail como medio para la comunicación entre el servidor y los clientes. Existen bastantes proyectos de este tipo que utilizan servicios conocidos para intentar evitar ser detectados por servicios de monitorización que detecten comportamiento inusual por tráfico, enmascarando la comunicación mediante estos servicios, se evade la seguridad por monitorización.

Por otra parte, tenemos los frameworks comerciales, desarrollados por empresas del sector y que son soluciones muy avanzadas y que ofrecen una mayor facilidad de uso en entornos profesionales, con funcionalidades mucho más complejas. Entre estos, algunos de los más conocidos son:

- Voodoo[10]: Voodoo es una plataforma colaborativa para operaciones de seguridad ofensiva para equipos. Se caracteriza por ser sigiloso, evitando la detección de soluciones antivirus, multiplataforma, para poder ser ejecutado tanto en Linux, Windows, MacOS e incluso en Android y dispositivos IoT, y dispone de una gran cantidad de módulos de explotación y que permiten recolectar todo tipo de información.
- Cobalt Strike[11]: Cobalt Strike es un agente de post-explotación, que permite la emulación de adversarios en una red interna, que hace uso de tecnologías propias como Malleable C2, que permite personalizar todo tipo de ataques para que parezcan diferentes en cada ataque. Dispone de gran cantidad de ataques, desde ataques de ingeniería social, hasta explotación de servicios. Además, se caracteriza por la generación de reportes personalizados con la información recolectada.

4 Metodologia

Se decidió optar por una metodología Agile, debido a la flexibilidad y adaptabilidad que esta proporciona respecto a otras metodologías como pueden ser la Waterfall. La metodología Agile se basa en las iteraciones y entregas continuas del software, de manera que este sea funcional a la vez que se añaden nuevas funcionalidades. Para el tipo de software del proyecto, esta metodología ha sido la idónea.

En el software desarrollado, se ha empezado con la implementación de un "core", formado por una pseudo-consola, que muestra la información como clientes conectados, panel de ayuda, etc. A este "core", se le han ido añadiendo diferentes funcionalidades mediante módulos, de manera que, de forma iterativa, la herramienta ha ido tomando forma.

5 Planificación

Para la planificación de este proyecto, se ha llevado a cabo un diagrama de *Gantt* que se puede encontrar en la Figura 20, dividiendo el trabajo en las etapas de Búsqueda de

información, Diseño, Implementación del servidor, Implementación del cliente y Desarrollo del repositorio GitHub.

6 Diseño

En esta primera etapa del proyecto, se comenzó con la definición del funcionamiento de la herramienta, mediante un diseño *abstracto* de la herramienta, que permitió identificar la estructura del programa, los *patrones de diseño*[12], que son soluciones habituales a problemas que se tienen que resolver con frecuencia a la hora de realizar el diseño de un software y finalmente para poder visualizar una primera idea de la herramienta. Tras la creación del diagrama abstracto, se identificaron los diferentes módulos que utilizaría la herramienta.

El diagrama de secuencia, que representa la lógica de la herramienta, lo podemos ver en la Figura 19.

6.1. Diagrama abstracto de la herramienta

Para entender a grandes rasgos el funcionamiento que tendrá la herramienta, y además poder identificar los patrones de diseños que serán útiles para la herramienta, se planteó un diseño *abstracto*. Mediante este diseño, podemos entender a grandes rasgos el funcionamiento de la herramienta. Esto permitió poder identificar patrones de comportamiento para posteriormente hacer los diagramas de clase y secuencia. El diagrama abstracto que representa la herramienta lo podemos encontrar en la Figura 1.

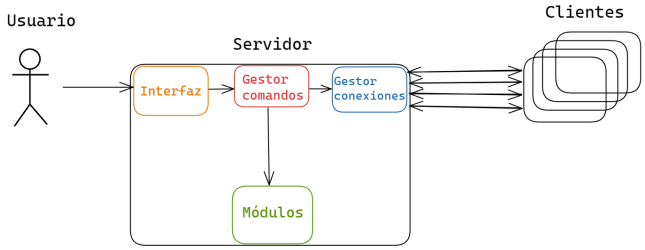

Fig. 1: Diagrama abstracto de la herramienta

En este diagrama, podemos identificar diferentes elementos:

- La *Interfaz* se encarga de interactuar con el usuario: recibir el input por parte del usuario y lo enviará al *Gestor de comandos*.
- El *Gestor de comandos* recibe el input desde la *Interfaz* y realizará las acciones oportunas según este input.
- Los *Módulos* son funcionalidades independientes que implementa la herramienta y que el *Gestor de comandos* utiliza cuando el input del usuario lo requiera.
- El *Gestor de conexiones*, que gestiona el envío y recibo de información entre servidor y clientes.

6.2. Patrones de diseño identificados

Una vez desarrollado un diagrama abstracto de la herramienta, se identificaron posibles patrones de diseño que podrían ser utilizados en la implementación con el objetivo de crear un código entendible, flexible y reutilizable.

Tras hacer una búsqueda entre los diferentes patrones de diseño, se pudieron identificar varios que funcionarían correctamente con el diseño planteado. Para el servidor, se utilizó el patrón *Singleton*[13]. Este es un patrón de diseño creacional que asegura que una clase tenga una única instancia. La clase *Server* debe ser una instancia única, para evitar problemas de red al intentar iniciar varios servidores.

Para la creación de los módulos, se utilizó el patrón *Template Method*[14]. Este patrón de diseño define el esqueleto de un algoritmo global en una superclase, permitiendo que las subclases sobreescriban la lógica del algoritmo sin modificar su estructura. Este patrón de diseño se utilizó para la implementación de los módulos. La superclase *Module*, es el esqueleto de los diferentes módulos de la herramienta. Gracias a este patrón de diseño, se crea una separación lógica mucho mayor entre módulos, logrando reutilizar aquella parte del código común entre los distintos módulos.

6.3. Diagramas de clase y secuencia

Tras la identificación de los patrones de diseño, se hizo una definición más exacta de la herramienta, mediante el uso de diagramas de clase y diagramas de secuencia. En el diagrama de clase del servidor, podemos ver las 3 clases principales (*Server*, *Client*, *CommandLineInterface*) y las implementaciones de los módulos, partiendo de la superclase *Module*, el diagrama se puede encontrar en la Figura 18. El diagrama de clase del cliente, está formado por la clase *Client* que gestiona toda la funcionalidad del cliente. Se encuentra en la Figura 17.

6.4. Definición de los módulos

En cuanto a los módulos que se identificaron como necesarios para la herramienta C2, se consultaron otros frameworks C2 conocidos y muy utilizados, como pueden ser Meterpreter[7] o Sliver[6], escogiendo aquellas funcionalidades más importantes para llevar a cabo un pentesting, teniendo en cuenta los requerimientos y el tiempo que se dispone para implementar los módulos. Los módulos seleccionados e implementados fueron los siguientes:

- *Run:* Este módulo implementa la funcionalidad de ejecución de comandos de sistema en los clientes. El pentester indica comando de sistema que quiera ejecutar sobre la máquina cliente, el cliente ejecuta estos comandos en su sistema, los envía al servidor, y este recibe el resultado de esos comandos, el cual muestra directamente sobre la interfaz del servidor C2.
- *UploadFile:* Mediante este módulo, el pentester puede subir desde el servidor C2 a un cliente, los archivos necesarios para realizar diferentes tareas, como extracción de información, ejecución de herramientas, entre otras acciones necesarias en el cliente.
- *DownloadFile:* Este módulo implementa la transferencia de archivos de un cliente al servidor C2. Mediante este módulo, el pentester es capaz de transferir cualquier archivo que requiera desde el cliente, como evidencias de información, resultado de ejecuciones de herramientas, ficheros propios de la máquina, etc.

- *Screenshot:* Este módulo implementa la funcionalidad de capturar y enviar el contenido de pantalla de los clientes. El pentester es capaz de recibir como imagen en el servidor C2, una captura de pantalla del estado actual de la interfaz de usuario en el cliente.
- *NetInfo:* Mediante este módulo, el pentester puede consultar la información de red de la máquina cliente más relevante, como pueden ser las interfaces de red, puertos abiertos y/o a la escucha, etc.

7 IMPLEMENTACIÓN

Tras el diseño de la herramienta, se comenzó la implementación de las clases para el desarrollo del C2.

Debido a que es uno de los objetivos del proyecto, se procedió a crear el repositorio GitHub para alojar la herramienta y poder mantener un control de versiones del código. En este repositorio[3] se han ido actualizando los archivos del código fuente durante la implementación de la herramienta C2, tanto el servidor, como el cliente. Tras la implementación de cliente y servidor, se creó la documentación de la herramienta en el repositorio.

Como nombre de la herramienta y del repositorio, se ha decidido utilizar "LONS-C2", Listener-Oriented Network Server (LONS), Command and Control (C2).

7.1. Implementación del servidor

El servidor es la parte encargada de interactuar con el usuario, el pentester, que indicará las acciones a realizar en cada uno de los clientes a través de la interfaz. El servidor gestiona las conexiones con cada uno de los clientes, los datos recibidos por cada uno de ellos, y muestra la información por pantalla. La implementación del servidor se ha dividido en la implementación del core del servidor, la gestión de las conexiones, la lógica del servidor y los módulos.

7.1.1. Implementación del core del servidor

El core de la herramienta consta de la interfaz de usuario con la estructura de clases y los paneles de ayuda. En esta fase, la herramienta aún no era funcional. El usuario podía interactuar con la interfaz de usuario, ver los paneles de ayudas, pero sin disponer de las acciones necesarias para la comunicación entre servidor y clientes. Una vez se implementó esta primera versión básica de la herramienta, se procedió a implementar el resto de funcionalidades.

7.1.2. Implementación de la gestión de las conexiones

Tras definir el core de la herramienta, se procedió a gestionar las conexiones servidor-clientes. El servidor dispone los sockets de los clientes, que es la interfaz que permite al servidor enviar y recibir información con cada uno de ellos. La clase encargada de esta función es la clase **Server**. Esta clase se encarga de gestionar las conexiones hacia los clientes, poder enviar información hacia los clientes y recibirla.

Para poder integrar la clase Server con las estructuras de datos necesarias para la gestión de los clientes, se creó la clase **Client**. Esta clase es la estructura de datos de un cliente. Es decir, esta clase instancia objetos con toda la información necesaria de un cliente para poder comunicarse con él.

7.1.3. Implementación de la lógica del servidor

Tras disponer del core del C2 y la gestión de las conexiones, se creó la lógica del servidor. Esta parte de la implementación es la encargada de gestionar que las acciones que pide el usuario ejecuten los procesos internos necesarios para llevar a cabo el resultado esperado. La clase encargada de esta función es *CommandLineInterface*. La lógica de esta clase se resume en la Figura 2.

La clase *CommandLineInterface* se gestiona el input del usuario para interactuar con el servidor, ejecutar las acciones necesarias según el input recibido, y mostrar los resultados por pantalla. Además de esto, *CommandLineInterface* se encarga de crear la instancia del servidor, inicializarla y llamar al método propio *commandHandler*, que a su vez llama al método *sessionHandler*. Estos métodos se encargan de gestionar el input del usuario para interactuar con la herramienta. Los métodos se pueden encontrar en el repositorio Github [15].

El método *commandHandler* se utiliza al iniciar la herramienta C2, cuando no hemos escogido todavía ninguna sesión interactiva, y cuando escogemos una sesión interactiva para ejecutar comandos o módulos en un cliente remoto, se ejecuta *sessionHandler*. Para escoger una sesión en un cliente, se usa el comando *sessions*. Al ejecutar *sessions*, nos aparece tanto las sesiones, como la opción de especificar la palabra clave *back*, para volver al menú de inicio sin escoger una sesión.

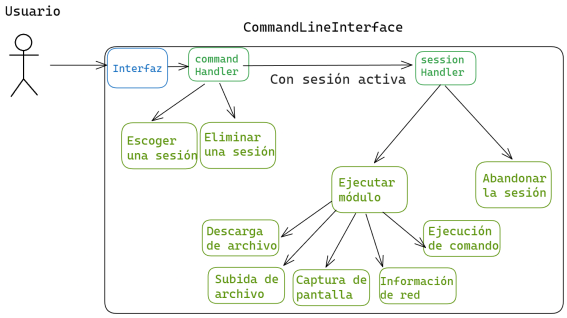

Fig. 2: Lógica del servidor - CommandLineInterface

Estas funciones crean una pseudo-consola, que recibirá el input del usuario de forma continua y según el input, se ejecutarán las acciones pertinentes. Al ejecutar la herramienta [3], se muestra por pantalla el endpoint en el que el servidor recibirá a nuevos clientes (0.0.0.0:1337), y la manera de ejecutar el panel de ayuda, con el comando *help*.

```
Server listening in 0.0.0.0:1337...
Type help to see options
mario@lons>
```

Fig. 3: Inicio de la ejecución de la herramienta C2

Para facilitar la experiencia de uso, la herramienta dispone de varios paneles de ayuda en cada uno de sus estados. Cada panel de ayuda muestra las acciones que el pentester puede realizar.

7.1.4. Implementación de los módulos

Finalmente, se implementaron los módulos que se encargan de añadir funcionalidades al core de la herramienta. Todos los módulos siguen una plantilla, en este caso una superclase, que proporciona el esqueleto a seguir para crear un módulo nuevo. Los módulos deberán adecuarse a la estructura de esta superclase para poder integrar la funcionalidad con la herramienta.

La clase *Module* es el esqueleto a seguir por los diferentes módulos. Cada clase o módulo dispone de dos métodos fijos. El método **init**, que representa la inicialización de la clase, establece los atributos *name* y *description*, que indican el nombre del módulo y una pequeña descripción de la funcionalidad que cumple. El método **execute** ejecuta las acciones definidas en la implementación del módulo, para llevar a cabo el objetivo del módulo. Podemos encontrar la clase *Module* en la Figura 4

```python
class Module:
    """
    Abstract class representing a module.
    """

    def __init__(self, name: str, description: str):
        """
        Initializes the Module with a name and description.
        """
        self.name = name
        self.description = description

    def execute(self):
        """
        Executes the module.
        """
        pass
```

Fig. 4: Clase Module, esqueleto de los módulos

A continuación veremos uno a uno cada uno de los módulos implementados siguiendo la clase vista anteriormente.

Primeramente, encontramos el módulo *Run*. Este módulo es el más útil para la mayoría de casos en una auditoría interna, donde se necesita ejecutar comandos de sistema en cada uno de los clientes. Este módulo permite al pentester interactuar con la máquina remota, a través del intérprete de comando del sistema, simulando una terminal en la máquina remota mediante un intercambio de información a través de sockets. El módulo se encarga de enviar el comando a ejecutar a través del socket, espera a su ejecución por parte del cliente, recibe el resultado de la ejecución del comando, y lo muestra por pantalla.

El módulo *UploadFile* tiene la función de subir un archivo local del servidor hacia uno de los clientes, en este caso el cliente que se haya escogido como sesión activa. Para implementar este módulo igual que el resto de módulos, se implementó la función execute, la cual realiza las acciones necesarias para el correcto funcionamiento del módulo. La función execute a su vez llama a otras funciones auxiliares creadas en cada uno de los módulos, que igual que el resto del código fuente, se pueden encontrar en [3].

En este caso, el módulo se encarga de leer el archivo local del sistema que el usuario ha indicado que quiere enviar al cliente, mediante el parámetro local_file_path, y mediante el método auxiliar sendFile, se envía a nivel de bytes por el socket, para que el cliente lo pueda recibir y guardar en el path que el usuario especifica como remote_file_path.

En el caso del módulo *DownloadFile*, es el método que se implementó con la idea, al igual que UploadFile, de ofrecer una transferencia bidireccional de archivos entre cliente y servidor. La idea es la misma que en el módulo anterior, pero cambiando el flujo del programa; el servidor se descarga un archivo del cliente, transfiriéndolo al servidor central en el que el pentester está utilizando la herramienta. El método implementado envía información al cliente mediante el socket, básicamente qué archivo se requiere para que el cliente lo pueda enviar. Tras esto, el servidor se queda a la espera del contenido del archivo del socket, el módulo recibirá el contenido y lo escribirá en la máquina local, en el path indicado a en local_file_path.

Seguidamente, encontramos el módulo *Screenshot*. Este módulo implementa la funcionalidad de mostrar una captura de pantalla del estado actual de la máquina remota, el cliente. El cliente implementa la mayor parte de la funcionalidad de esta acción, y en el caso del servidor, se limita a enviar la información necesaria para que el cliente pueda identificar que debe realizar esta acción. El servidor se encarga de crear la imagen con los datos recibidos por el socket, que será el contenido de la imagen en bytes.

Finalmente, tenemos el módulo *NetInfo*. Este método implementa varias funciones encargadas de filtrar información, tanto de las interfaces de red como los puertos abiertos del cliente, y los muestra en un formato entendible para el usuario. El objetivo de este módulo es poder mostrar la información más relevante sobre las conexiones de red de la máquina remota.

7.2. Implementación del Cliente

Tras la implementación del Servidor, lo siguiente fue proceder con la implementación de la parte del Cliente. El cliente se encarga de establecer la conexión hacia el servidor, quedarse a la escucha de posibles comandos, y en caso de recibir comandos o acciones, realizarlos y enviar los datos de vuelta hacia el servidor. Como hemos visto en la implementación del servidor, este, a su vez, gestiona esta información recibida por el cliente, mostrándola por pantalla de una manera amigable y entendible por el usuario.

Lo primero que se implementó en la parte del cliente fue la gestión de las conexiones, para establecer el envío de información bidireccional entre cliente y servidor. Mediante la función *makeConnection* el cliente crea dos sockets. El primero, definido en el atributo server_socket, es el que se usa para enviar toda la información relevante, proveniente de la ejecución de comandos, resultados de datos, ficheros, etc. El segundo, control_socket, es el socket encargado de enviar información de control, permitiendo al servidor gestionar el estado del cliente en todo momento, así como recibir un string inicial para poder identificar información relevante al sistema operativo del cliente, de lo que también se encarga esta misma función.

Tras esto, se implementó la gestión de la información recibida desde el servidor, mediante la función *listenForCommands*. Esta función es la encargada de gestionar el flujo del programa según la información que recibimos del servidor. Mediante un conjunto de condicionales, se procedió a implementar las funcionalidades dentro de cada caso, que representa cada una de las acciones que el servidor espera del cliente.

Por lo tanto, el funcionamiento del cliente es el siguiente;

se ejecuta continuamente la función listenForCommands, si se recibe alguno de los de los comandos relevantes definidos previamente en la implementación del servidor (ejecución de comandos, descarga/subida de archivos, información de red, captura de pantalla), el cliente lo gestiona con las acciones pertinentes.

Una vez hemos visto la estructura que se encarga de ejecutar las acciones por parte del cliente, vamos a ver en detalle cada una de ellas para ver su implementación y funcionamiento.

7.2.1. Función auxiliar executeAndSend

Lo primero que se implementó en el cliente fue la función auxiliar *executeAndSend* que podemos ver en la Figura 5. Esta función es utilizada por varios casos en la estructura del programa.

```
def executeAndSend(self, command):
    """
    Handles the case of running a system command, sending the data back to the server
    """

    output = os.popen(command).read()
    print("Running command "+ Fore.RED + "{}".format(command) + Style.RESET_ALL)
    self.server_socket.send(output.encode('utf-8'))
```

Fig. 5: Función auxiliar executeAndSend

La función *executeAndSend* recibe por parámetro un comando de sistema a ejecutar, y mediante la librería **os**, la cual permite usar funcionalidades del sistema operativo de forma abstracta, ejecuta el comando con la función os.popen(), y extrae el output del comando con la función auxiliar read(). Finalmente, envía el output del comando por el buffer del socket.

7.2.2. Ejecución de comandos

Haciendo uso de la función executeAndSend, la implementación de la ejecución y envío de comandos es tan simple como hacer uso de la función, con el comando que recibamos por parte del servidor, mediante el socket.

Cuando el cliente recibe la cadena command, filtra el resto de la cadena del buffer, que es el comando a ejecutar. Haciendo uso de la función auxiliar que hemos definido, ejecuta el comando y envía el output de vuelta al servidor, que se encargará de mostrar los resultados por pantalla.

7.2.3. Descarga de archivos

A continuación se implementó la funcionalidad de descarga de archivos. Esta funcionalidad espera a recibir por el buffer la cadena que le permite identificar la ejecución de la funcionalidad, y en tal caso filtra el resto de la cadena del socket, que en este caso es el archivo que el servidor le ha indicado que quiere descargar. De esta manera, el cliente lee este archivo de su sistema operativo, haciendo uso de la función nativa *open*, en pedazos de 1024 bytes el archivo, y los envía por el socket, de esta manera, el cliente envía el contenido a medida que lo va leyendo, y el servidor irá recibiendo en pedazos este archivo.

Esto se ha implementado así, debido a que se han encontrado problemas de inconsistencia a la hora de leer archivos de gran tamaño e intentar guardar todo su contenido en una variable, por lo que se ha optado por esta manera, que en cuanto a velocidad de transferencia no ha empeorado la implementación anterior debido a la gran velocidad de lectura y escritura tanto de los discos duros como las redes en la actualidad.

7.2.4. Subida de archivos

En el caso de la subida de archivos, se implementó el caso inverso a la descarga de archivos. La implementación es un poco más compleja que la descarga debido a que el servidor debe enviar más información al cliente.

Para empezar, el cliente debe conocer el formato de la información que envía el servidor. En este caso, el cliente necesita recibir la cadena identificativa de subida de archivos, el archivo donde se guardará la información, ya que es un envío de archivo del servidor hacia el cliente, y finalmente el contenido del archivo en bytes.

Debido a esto, se ha definido el siguiente formato de mensaje.

upload**<pathArchivo>**FILE_START**<contenido>**

Lo primero que encontramos es la cadena upload, que permite al cliente saber que la información que reciba después de esta cadena, es referente a una subida de archivos desde el servidor. Después, se encuentra el path del archivo, que es la ruta donde el cliente guardará el archivo. Tras esto, encontramos la cadena identificativa FILE_START, que nos servirá tanto para poder filtrar el path del archivo, que se encontrará entre las cadenas upload y FILE_START, como para poder saber que después de esta cadena, el resto del contenido será el contenido del archivo en bytes a guardar.

A partir de aquí, y de la misma manera que la implementación de la descarga de archivos, el cliente va leyendo contínuamente el contenido del buffer del socket y va escribiendo este contenido en bytes en la ruta de archivo proporcionada por el servidor, de manera que cuando escribamos el contenido al completo, tendremos el archivo que hemos enviado desde el servidor en la ruta que hemos especificado, en el sistema operativo del cliente.

7.2.5. Información de red

En cuanto al módulo encargado de mostrar información relevante sobre la red, en la parte del cliente, tan solo se envían los comandos de sistema necesarios para que el servidor gestione esta información, la filtre, y muestre aquella información más importante por pantalla. Debido a esto, se compone de dos casos. El primero, cuando recibamos en el cliente la cadena netstat, el cliente en ese momento envía al servidor el resultado del comando *netstat -nat*. De esta manera, se envían los puertos abiertos del cliente al servidor. Seguidamente, al recibir la cadena interfaces. En este caso, se tiene en cuenta el sistema operativo del cliente. Si el cliente es un sistema Windows, se ejecuta el comando ipconfig, en caso contrario, el comando ifconfig. Básicamente, los dos muestran información sobre las interfaces de red.

Con toda esta información, el servidor es capaz de filtrar y mostrar de una manera simple los puertos e interfaces del cliente, mediante el módulo NetInfo.

7.2.6. Captura de pantalla

Finalmente, tenemos el caso en el que el cliente envía una captura de pantalla al servidor. En este caso, mediante el uso de la librería **PIL** (Python Imaging Library), el cliente toma una captura de pantalla del sistema, que será lo que ve por pantalla en ese instante, y mediante la librería **io** guarda el contenido en bytes de la imagen en un buffer. Tras esto, el cliente envía al servidor el contenido de la imagen para mostrarla al pentester.

8 Desarrollo del repositorio GitHub

Tras finalizar la implementación de servidor, cliente y la integración de estos, el último paso y cumpliendo con uno de los objetivos principales del proyecto, será publicar la herramienta para la comunidad, mediante la creación del repositorio público en GitHub[3], para alojar el código fuente de la herramienta, en conjunto a una pequeña guía de uso.

Esto permite no solo que la comunidad pueda usar libremente la herramienta, sino que también puede aportar sus ideas y colaborar para mejorarla. Esto se puede hacer de varias maneras: escribiendo en el propio repositorio una "Issue", un hilo donde se reportan problemas encontrados, nuevas ideas, donde el creador y otras personas comentan estas ideas y proponen mejoras. La segunda opción es aportar una idea directamente en forma de código, haciendo un "fork"[16] o bifurcación del código inicial, mejorándolo y enviando la nueva propuesta al creador, el cual verifica los cambios y si todo está correcto, acepta la propuesta y esta pasaría a ser el nuevo código fuente del repositorio. Se creó el repositorio, para después crear el markdown como guía de uso y finalmente la estructura de directorios con los archivos de código fuente.

8.1. Creación del repositorio público

En cuanto a la creación del repositorio público, en la página de GitHub, accediendo a la sección inicial, veremos un botón "New" que nos permite crear un nuevo repositorio.

Seguidamente, indicaremos los detalles del repositorio, como el nombre, descripción y el tipo de proyecto. En nuestro caso, es un repositorio público.

8.2. Creación del markdown

Al crear un repositorio en GitHub, nos da la opción de añadir un README file. El propósito de este archivo es proporcionar información de uso, una guía para el correcto uso del proyecto, con el objetivo de que los usuarios y desarrolladores que visitan el repositorio conozcan el propósito de la herramienta, y puedan utilizarla de forma sencilla, sin conocimiento previo.

El formato de este archivo es *markdown*[17], un lenguaje que permite dar formato al texto de manera sencilla: crear índices, secciones, imágenes y dar una estructura clara al texto, para facilitar el acceso al repositorio a los usuarios.

El archivo de markdown del repositorio se identifica por README.md, entre los archivos del repositorio. Al navegar a la página web del repositorio, por defecto GitHub carga el contenido de este archivo con el formato indicado, de manera que actúa como portada del repositorio.

8.3. Creación de estructura de directorios

Finalmente, se subió el código fuente de la herramienta, servidor y cliente, y otros archivos necesarios, como las librerías y recursos utilizados, y otra información en los archivos del repositorio. Como podemos ver en la Figura 6, la estructura se divide entre el código fuente del servidor, el cliente y otros recursos utilizados para la creación del repositorio.

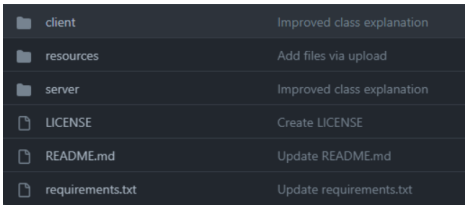

Fig. 6: Estructura de directorios del repositorio

9 Resultados

Finalmente, en este apartado, mediante un pequeño entorno de pruebas que simularía un caso de uso real de la herramienta, veremos los resultados del proyecto; verificar el correcto funcionamiento de la herramienta y mostrar el uso de la herramienta en las diferentes funcionalidades que ofrece.

9.1. Definición del entorno

Primeramente, definiremos el entorno en el que haremos las pruebas. La herramienta es multiplataforma, por lo que tanto servidor como cliente pueden ser ejecutados en sistemas Windows o Linux. En este caso haremos uso de un sistema Windows, y dos sistemas Linux. Los sistemas Linux serán dos máquinas virtuales basadas en Debian 10 y el sistema Windows que se utilizará será un Windows 10 Pro.

Haremos una prueba utilizando el servidor en la máquina Windows y los clientes en las dos máquinas Linux, el esquema lógico del entorno lo podemos ver en la Figura 7.

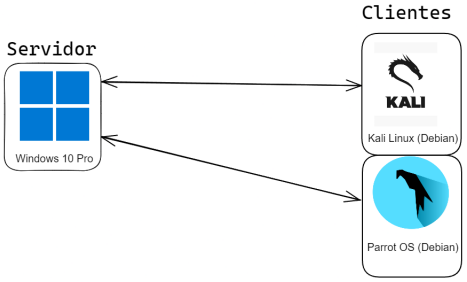

Fig. 7: Entorno de pruebas

9.2. Definición de tests a ejecutar

En este apartado se enumeran los tests a realizar, con el objetivo de comprobar el correcto funcionamiento de la herramienta, ver cómo se comporta la interfaz de usuario y verificar que la herramienta produce el resultado esperado.

Para ello, se han definido un conjunto de tests que se ejecutarán simularán el entorno de un pentester en una auditoría de seguridad real. Los tests son los siguientes:

- Elección de una sesión
- Ejecución de un comando de sistema en esa sesión
- Ejecución del módulo de captura de pantalla
- Abandono y eliminación de la sesión y elección de la otra sesión.
- Envío de un archivo en esa sesión mediante el módulo de subida de archivos.
- Descarga de ese mismo archivo mediante el módulo de descarga de archivos.
- Ejecución del módulo de información de red.

9.3. Ejecución de las pruebas

Primeramente, se inicia el servidor, como podemos ver en la Figura 8. Podremos ver como nos proporciona la información habitual por consola: el comando *help* para mostrar el panel de ayuda, y el endpoint en el que escucha.

Fig. 8: Inicialización del servidor

A continuación, ejecutaremos el programa de cliente en los dos clientes. Podremos ver que los clientes se han conectado correctamente, con la cadena "Connected to the server - IP:PUERTO" en cada una de las terminales de los clientes, y si nos dirigimos a la terminal del servidor, veremos también dos mensajes que nos permitirán identificar que los clientes se han conectado, como vemos en la Figura 9.

Fig. 9: Clientes conectados desde la terminal de servidor

9.3.1. Elección de una sesión

Una vez servidor y clientes están conectados, empezaremos por escoger una sesión. Haciendo uso del comando *sessions*, como se puede ver en la Figura 10, nos mostrará las sesiones activas, que en este caso serán dos. Escogeremos la sesión n.º 2, que corresponde a la máquina "Parrot OS".

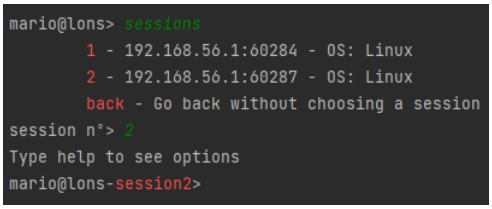

Fig. 10: Test número 1 - Elección de una sesión

9.3.2. Ejecución de un comando de sistema en esa sesión

A continuación, haciendo uso del módulo *Run*, que sirve para ejecutar un comando de sistema, ejecutaremos el comando *hostname*. Este comando permite ver el nombre de la máquina, y nos permite identificar en cuál de las máquinas nos encontramos. Como resultado, obtenemos que el nombre de la máquina es *parrot*, y podemos verificar que efectivamente nos encontramos en la máquina con sistema operativo Parrot OS, como vemos en la Figura 11.

Fig. 11: Test número 2 - Ejecución de un comando de sistema

9.3.3. Ejecución del módulo de captura de pantalla

A continuación, ejecutaremos el módulo para capturar la pantalla del cliente, mediante el comando *screenshot*, como se ve en la Figura 12. Como resultado, nos abrirá la captura de pantalla con el editor de imágenes por defecto del sistema y nos guardará la captura de pantalla en la ruta que le indiquemos.

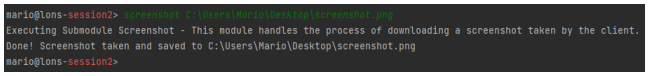

Fig. 12: Test número 3 - Ejecución del módulo de captura de pantalla

9.3.4. Abandono y eliminación de la sesión y elección de la otra sesión

Haciendo uso del comando *back*, abandonaremos la sesión actual, volviendo al panel principal de la herramienta. Para eliminar la sesión, ejecutaremos el comando *delete*, el cual nos permitirá escoger que sesión queremos borrar, borraremos la sesión en la que nos encontrábamos (2). Finalmente, escogeremos la otra sesión haciendo uso del comando *sessions* y indicando la sesión, como podemos ver en la Figura 13.

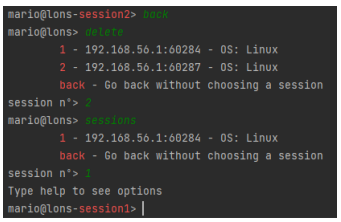

Fig. 13: Test número 4 - Terminal

9.3.5. Envío de un archivo en esa sesión mediante el módulo de subida de archivos

En este test, se enviará un archivo de prueba para comprobar que la transferencia es correcta, manteniendo la integridad del archivo. Haciendo uso del comando *upload_file*, indicando las rutas de origen y destino, conseguimos enviar el archivo, como vemos en la Figura 14. Al calcular el hash

del archivo original y el recibido, obtenemos el mismo valor, por lo que la integridad no se ha visto afectada.

Fig. 14: Test número 5 - Ejecución del módulo de subida de archivos

9.3.6. Descarga de ese mismo archivo mediante el módulo de descarga de archivos

En este test, se ha comprobado el funcionamiento de la descarga, descargando el archivo del test anterior y verificando que la integridad sea correcta. La descarga del archivo se ha realizado con el comando *download_file*, como vemos en la Figura 15. Tras ejecutar el módulo y comprobar el hash del archivo recibido, se ha podido verificar que los archivos mantienen su integridad.

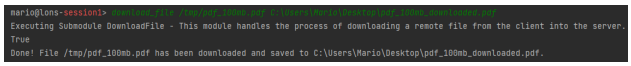

Fig. 15: Test número 6 - Ejecución del módulo de descarga de archivos

9.3.7. Ejecución del módulo de información de red

Finalmente, con el comando *netinfo*, mostraremos la información de red más relevante de la máquina cliente. Tras ejecutar el módulo, podemos identificar varios puertos abiertos de la máquina, así como todas las interfaces de red que dispone, como vemos en la Figura 16.

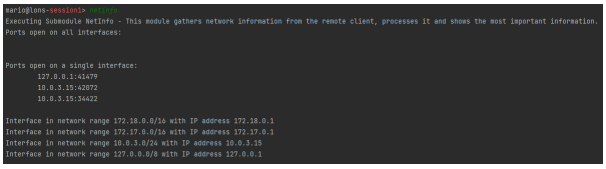

Fig. 16: Test número 7 - Ejecución del módulo de información de red

9.4. Discusión de los resultados obtenidos

Para finalizar, tras realizar todos los tests que se habían planteado, con el objetivo de poder verificar las funcionalidades implementadas por la herramienta, podemos extraer las siguientes conclusiones sobre los resultados obtenidos.

LONS-C2 permite gestionar de forma centralizada un conjunto de clientes, mediante un servidor centralizador, que permite escoger un cliente entre el conjunto de clientes conectados, cumpliendo uno de los objetivos establecidos para el proyecto. Por otro lado, el cliente permite al pentester conectar cualquier máquina que requiera ser gestionada por el servidor, alineado con su objetivo respectivamente. Tanto cliente como servidor, al estar desarrollados en Python, son multiplataforma. Es decir, funcionan en plataformas Windows y Linux, ofreciendo funcionalidades básicas que cualquier pentester en una auditoría a una red interna podría necesitar: transferencia de archivos, ejecución de comandos, extracción de capturas de pantalla, entre otras. Describe las funcionalidades mediante un panel de ayuda accesible desde la interfaz, la cual es amigable con el usuario que está acostumbrado a utilizar la consola o terminal, que es el usuario objetivo de esta herramienta.

Tras un uso exhaustivo de la herramienta, utilizando todas las funcionalidades implementadas en diferentes entornos, la herramienta cumple con los requisitos planteados en la etapa de diseño. Los resultados obtenidos han sido logrados en parte con los conocimientos obtenidos durante la carrera, en gran parte la motivación del trabajo fue haber utilizado sockets en la asignatura *Xarxes*, cosa que supuso un gran reto, sobre todo en cuanto a la gestión de la información recibida por el socket. Esto, en conjunto a mi interés por la ciberseguridad, dieron lugar a querer realizar este proyecto y adquirir los conocimientos adicionales que hicieron falta para lograr con éxito los objetivos establecidos.

Por lo tanto, se puede afirmar que los objetivos establecidos en el inicio del proyecto, han sido cumplidos con éxito, cada uno en sus respectivas fechas, con cierta variación en alguna de las fases, debido a la dificultad de poder estimar con exactitud el tiempo que llevaba la implementación, testeo, y posterior corrección de errores encontrados en el proceso.

10 Conclusiones

Un Command and Control o C2, es una infraestructura que permite controlar un conjunto de máquinas desde un servidor central. LONS-C2, una herramienta C2, surge con el objetivo de ofrecer una alternativa simple, multiplataforma, basada en sockets TCP, para automatizar y facilitar la realización de tareas de un auditor de seguridad o pentester en una auditoría interna de seguridad, donde se tienen que gestionar sesiones por terminal en distintas máquinas.

Para llevar a cabo dicha herramienta, se ha dividido el trabajo en distintas fases. Primeramente, una fase de diseño, la cual comenzó con un diagrama abstracto de la herramienta, que permitió identificar los patrones de diseño que se utilizarían. Tras esto, se hizo el diagrama de clase del servidor y el cliente, para terminar con el diagrama secuencia que muestra el funcionamiento dinámico de la herramienta en conjunto.

Tras el diseño de la herramienta, se llevó a cabo la implementación; primeramente del servidor, implementando en un principio la interfaz de usuario por terminal que muestra el menú inicial y los paneles de ayuda, se continuó con la implementación de la gestión de las conexiones sobre los clientes, la lógica de servidor con cada una de las funcionalidades que ofrece, para terminar por la implementación de los módulos.

Tras la implementación del servidor, se hizo lo mismo con el cliente, implementando las acciones a realizar tras recibir las peticiones de cada una de las funcionalidades establecidas por el servidor.

Después, se creó el repositorio en GitHub que aloja la herramienta de forma pública, alineado con el objetivo de ofrecer la herramienta en formato open-source. Finalmente, se ejecutaron un conjunto de tests para verificar el correcto funcionamiento de la herramienta, obteniendo el resultado esperado en todos ellos.

La herramienta ofrece funcionalidades básicas para un pentest: transferencia de archivos, ejecución de comandos,

extracción de capturas de pantalla, etc. Es una herramienta multiplataforma, por lo que puede ser utilizada en cualquier entorno independientemente del sistema operativo, cosa que la hace útil al trabajar en entornos empresariales.

En cuanto al aprendizaje personal que extraigo del proyecto, creo que me ha servido para aprender a gestionar el tiempo eficientemente, centrando los esfuerzos de forma ordenada en aquellas tareas más importantes y que requerían más tiempo, lo cual creo que ha sido clave para el resultado del proyecto. También me ha ayudado a reforzar mis habilidades técnicas en cuanto a la creación de un proyecto en el que un conjunto de clases deben interactuar entre sí, teniendo en cuenta el uso de threads, concurrencia e intercambio de información por red a través de sockets.

En resumen, se han conseguido cumplir todos los objetivos establecidos desde un inicio: crear una herramienta que permita automatizar y agilizar el trabajo de un pentester, creando tanto la parte del servidor como el cliente, y alojar la herramienta en un repositorio público en GitHub para su uso libre.

11 Trabajo futuro

En cuanto al trabajo futuro, lo primero sería optimizar la información que se muestra en la interfaz de usuario por terminal, para poder visualizar en todo momento información como clientes conectados, clientes desconectados, etc.

El segundo punto que sería importante es un protocolo de encriptado mutuo entre servidor y clientes, para cifrar toda la información que se envía entre ellos.

Agradecimientos

Me gustaría dar las gracias a mi familia y amigos, por el apoyo durante este proyecto, por estar siempre ahí cuando los he necesitado.

También me gustaría agradecer el apoyo a mi tutor, Ramon Martí, ayudándome en las distintas fases del proyecto, aportando siempre consejos útiles e ideas.

Referencias

[1] I. D. Media. “Los Servicios gestionados supondrán el 50 % de la Inversión en Seguridad TI en 2019.” (2019), dirección: https://www.itdigitalsecurity.es/actualidad/2019/06/los-servicios-gestionados-supondran-el-50-de-la-inversion-en-seguridad-ti-en-2019 (visitado 11-02-2023).

[2] J. Gardiner, M. Cova y S. Nagaraja, “Command & Control: Understanding, Denying and Detecting,” *researchgate.net*, ago. de 2014.

[3] m4riio21. “LONS-C2: Listener-Oriented Network Server (LONS), socket-based command and control server.” (2023), dirección: https://github.com/m4riio21/LONS-C2 (visitado 07-05-2023).

[4] “Red Team: C2 frameworks for Pentesting.” (2021), dirección: https://resources.infosecinstitute.com/topic/red-team-c2-frameworks-for-pentesting/ (visitado 13-02-2023).

[5] Tcostam. “awesome-command-control: A collection of awesome command & control (C2) frameworks, tools and resources for post-exploitation and red teaming assessments.” (2020), dirección: https://github.com/tcostam/awesome-command-control (visitado 13-02-2023).

[6] BishopFox. “Sliver: Adversary Emulation Framework.” (2019), dirección: https://github.com/BishopFox/sliver (visitado 17-03-2023).

[7] OffSec. “Meterpreter basics - metasploit unleashed.” (2019), dirección: https://www.offsec.com/metasploit-unleashed/meterpreter-basics/ (visitado 17-03-2023).

[8] “Kali docs: Kali Linux Documentation.” (), dirección: https://www.kali.org/docs/ (visitado 27-02-2023).

[9] byt3bl33d3r. “gcat: PoC backdoor that uses Gmail as a C2 server.” (2019), dirección: https://github.com/byt3bl33d3r/gcat (visitado 17-03-2023).

[10] S. 2. Security. “Voodoo - Stage 2 Security.” (2022), dirección: https://s2.security/voodoo (visitado 17-03-2023).

[11] Fortra. “Cobalt Strike — Adversary Simulation and Red Team Operations.” (2023), dirección: https://www.cobaltstrike.com/ (visitado 17-03-2023).

[12] R. Guru. “Design Patterns.” (2014), dirección: https://refactoring.guru/es (visitado 15-03-2023).

[13] R. Guru. “Singleton.” (2014), dirección: https://refactoring.guru/es/design-patterns/singleton (visitado 16-04-2023).

[14] R. Guru. “Template Method.” (2014), dirección: https://refactoring.guru/es/design-patterns/template-method (visitado 16-04-2023).

[15] m4riio21. “LONS-C2: CommandLineInterface class and methods.” (2023), dirección: https://github.com/m4riio21/LONS-C2/blob/main/server/CommandLineInterface.py (visitado 26-04-2023).

[16] GitHub. “Fork a repo - GitHub Docs.” (), dirección: https://docs.github.com/en/get-started/quickstart/fork-a-repo (visitado 13-05-2023).

[17] GitLab. “Handbook markdown guide.” (2016), dirección: https://about.gitlab.com/handbook/markdown-guide (visitado 13-05-2023).

APÉNDICE

A.1. Diseño y planificación

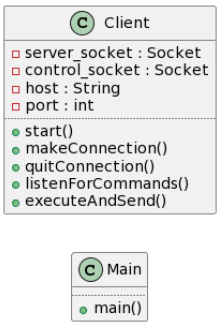

Fig. 17: Diagrama de clase del cliente

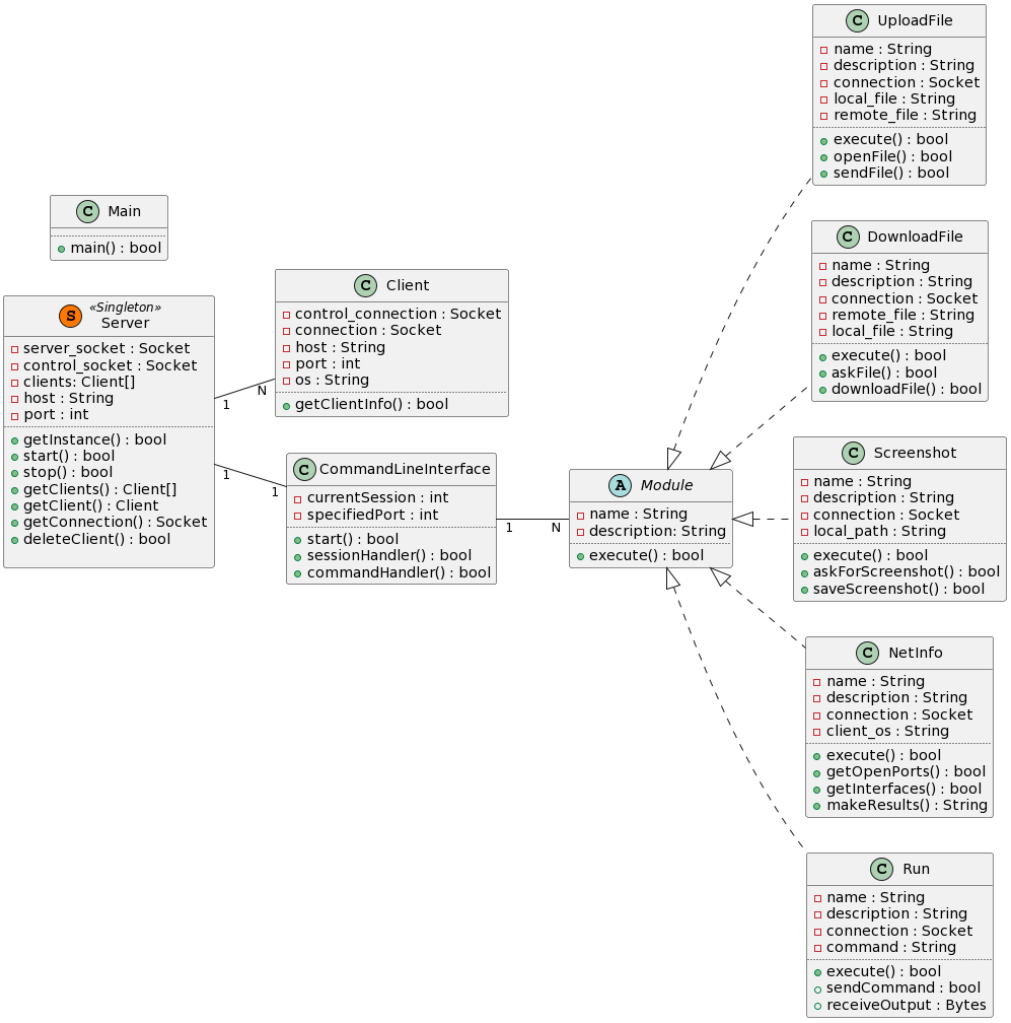

Fig. 18: Diagrama de clase del servidor

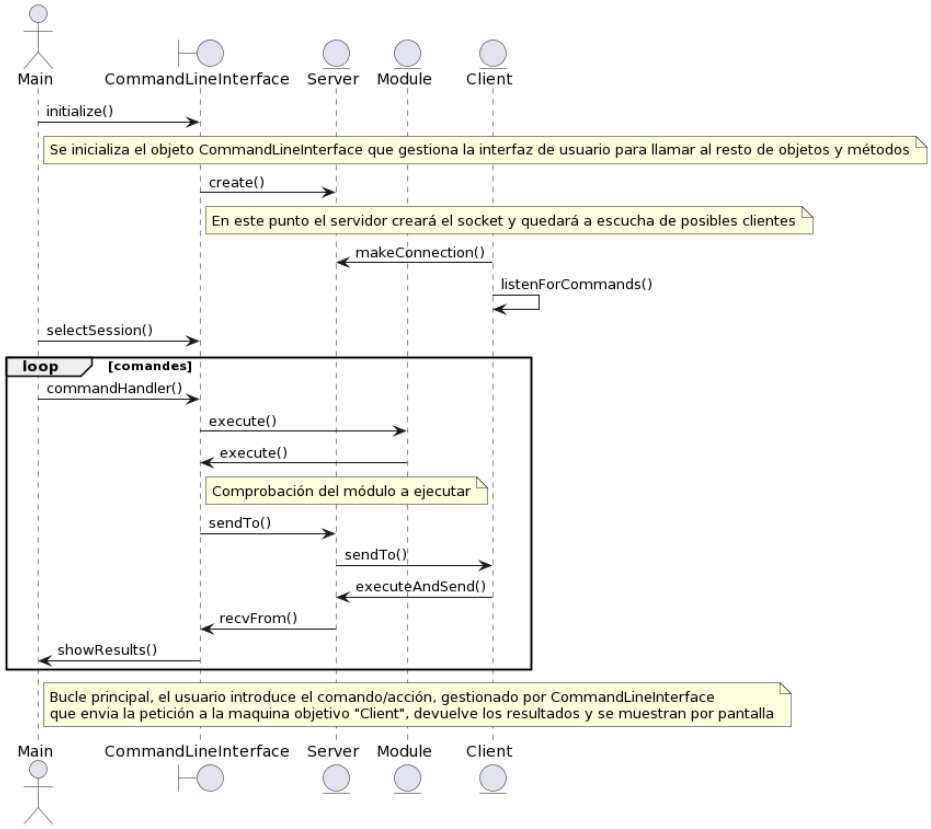

Fig. 19: Diagrama de secuencia de la herramienta

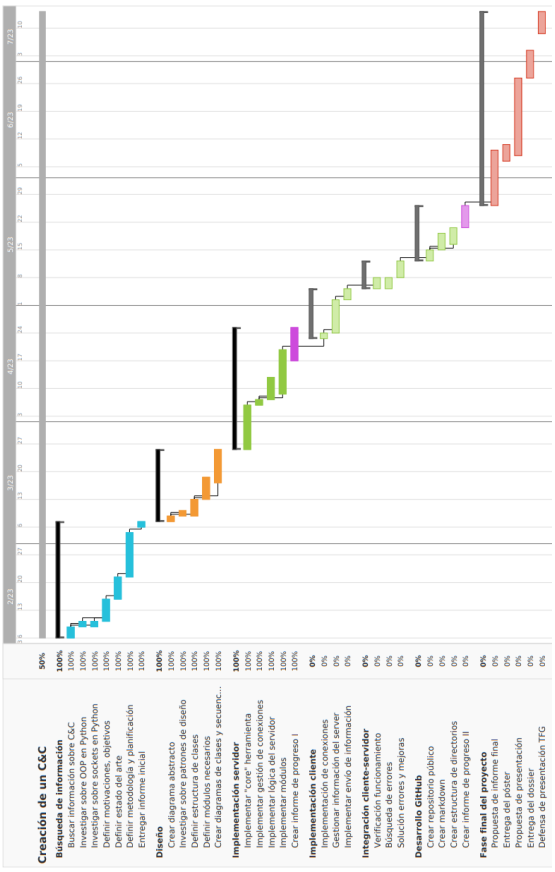

Fig. 20: Diagrama de Gantt