Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA (SG)

Grado en Ingeniería Informática de Servicios y Aplicaciones

Estudio de una Botnet Bajo la Arquitectura Cliente-Servidor

Autor: Carlos Sánchez Mora

Tutor: Juan José Álvarez Vicente

A mi familia

Agradecimientos

Agradecimientos a mi familia y a mi pareja por creer en mí, así, como el apoyo que me han brindado a lo largo de este camino.

Resumen

Las botnets hoy en día suponen un riesgo real en el mundo IOT, ya que en los últimos años han surgido organizaciones de cibercriminales que las utilizan con fines poco éticos con la finalidad de lucrarse en términos económicos extorsionando y chantajeando a las compañías a cambio de un rescate.

Con el objetivo de proponer posibles soluciones que ayuden a las empresas a poner freno a estas redes masivas, se realizará un diseño e implementación real de una botnet mediante el uso de protocolo TCP/IP, permitiéndonos realizar un estudio en profundidad, aprendiendo así sobre técnicas de malware como la persistencia en máquinas, infección de dispositivos y la comunicación en red.

El software creado también podrá servir como herramienta de auditoría, ayudando a determinar la resistencia de un sistema.

Abstract

Botnets today pose a real risk in the IOT world, since in recent years cybercriminal organizations have emerged that use them for unethical purposes in order to profit in economic terms by extorting and blackmailing companies in exchange for a rescue.

With the aim of proposing possible solutions that help companies to stop these massive networks, a real design and implementation of a botnet will be carried out through the use of the TCP/IP protocol, allowing us to carry out an in-depth study, thus learning about techniques of malware such as persistence in machines, device infection and network communication.

The software created can also serve as an audit tool, helping to determine the resilience of a system.

Índice general.

Índice de figuras.

Índice de tablas.

Capítulo 1.

Descripción general del Proyecto.

1.1. Introducción

Vivimos en un mundo en el que la mayoría de dispositivos electrónicos están conectados a internet, a los que denominamos dispositivos IoT (internet of things), algo que nos facilita la vida en muchos aspectos y que a todo el mundo le parece un maravilloso avance pero a veces no somos conscientes del riesgo al que exponen estos dispositivos nuestros datos, debido a que muchos de ellos poseen vulnerabilidades de seguridad, o simplemente el usuario que los gobierna abusa de malas prácticas que comprometen su propia seguridad.

Es posible que los ciberatacantes puedan aprovechar los hechos anteriormente mencionados para lucrarse acosta de nuestros datos o de los equipos de computación de personas ajenas, siempre con fines delictivos.

En este proyecto se pretende realizar un estudio que permita comprender las técnicas que estos delincuentes implementan para comprometer miles de dispositivos IoT y posteriormente añadirlos a una red de máquinas infectadas de manera que puedan ser controlados por ellos mismos, esto es lo que denominamos una Botnet. Para ello se diseñará un software que permita crear una Botnet [2] basada en un servidor central que sea capaz de coordinar a las máquinas comprometidas a través del protocolo TCP/IP.

1.2. Motivación

El principal motor que mueve este proyecto es el interés personal despertado por el surgimiento de estas organizaciones y de saber cómo operan creando la infraestructura para levantar una botnet, así como las diferentes herramientas, técnicas y protocolos de comunicación entre máquinas.

También se tratarán las diferentes técnicas auxiliares que se utilizan para garantizar el éxito de este tipo de softwares como podrían ser la suplantación y ocultación de procesos, persistencia en máquinas infectadas y mecanismos para la instalación silenciosa.

Todo ello está motivado por la capacidad de conocer la metodología de ataque y poder ayudar proponiendo soluciones al respecto.

1.3. Alcance

Desarrollar, implementar y realizar un estudio de una red de ordenadores conectada a un servidor de comando y control utilizando técnicas avanzadas de infección, persistencia y comunicación en red.

La herramienta va orientada a estudiantes y profesionales del sector con la finalidad de ayudar en la labor educativa y de investigación, así como su utilización en auditorías.

Toda la información y conocimientos que pueda aportar el proyecto podrán ser utilizados para la mejora de la seguridad de los sistemas informáticos.

1.4. Objetivos del Proyecto

Como objeto de estudio y objetivo principal del proyecto será la implementación de una botnet centralizada basada en sockets bajo la arquitectura cliente-servidor [4] donde se puedan estudiar y poner en práctica algunas de las técnicas actuales y poder proporcionar una visión general del funcionamiento de una botnet únicamente con fines educativos o de investigación.

Será interesante recrear un escenario controlado donde probar el software, por lo que el despliegue consistirá en el despliegue de un servidor central en una máquina controlada por el atacante que a su vez será quien envíe las órdenes a los bots. Entre las opciones planteadas se encuentran la ejecución remota de código arbitrario en las máquinas infectadas pertenecientes a la botnet, la posibilidad de abrir una reverse Shell [16] en cualquiera de las máquinas conectadas al servidor y la opción de subir o bajar ficheros desde y hacia el servidor.

Un objetivo del proyecto es demostrar la posibilidad de evasión de algunos de los sistemas de defensa, por ello la botnet está basada en reverse Shell, lo que permitirá evadir un dispositivo firewall instalado en la máquina víctima.

Capítulo 2.

Estado del Arte.

2.1. Introducción

Una botnet es un conjunto de dispositivos informáticos comprometidos que son controlados de forma remota por un actor malintencionado y que están a expensas de recibir y cumplir órdenes que un servidor central manejado por la organización pueda ordenar. Las botnets pueden ser utilizadas para llevar a cabo una variedad de actividades, como ataques de denegación de servicio distribuido (DDoS) [17], robo de información o envío de spam. Es importante resaltar que el desarrollo y uso de este tipo de software para fines maliciosos es ilegal y perjudicial para la seguridad de los sistemas informáticos.

2.2. Contexto con la Realidad

En el contexto de la seguridad informática, el desarrollo de herramientas y tecnologías ha llevado a la creación de diversas soluciones con potencial tanto positivo como negativo.

En la actualidad, existen muy pocas herramientas que permitan simular el comportamiento de una botnet, por razones legales y éticas, ya que pueden ser usadas con fines malintencionados, pero como todo, dependiendo del enfoque que le queramos dar, podemos aprovechar este tipo de infraestructuras para aprender, auditar, y de alguna manera mejorar nuestros sistemas informáticos. Por esta razón nace **BotCommander**, una implementación de una botnet centralizada desarrollada en Python, mediante la cual se pretende concienciar a los usuarios de la existencia de este tipo de softwares, y para usuarios con un nivel más técnico permitirles conocer algunas de las técnicas utilizadas por cibercriminales de manera que este conocimiento sirva en un presente para poder avanzar en materia de la ciberseguridad.

2.3. Tipos de Botnet Existentes

Existen 2 tipos principales de Botnets [2].

1. Arquitectura centralizada: La botnet sigue el esquema cliente-servidor, los bots o máquinas zombie son conectadas a un servidor común manejado por el atacante quien es el que las gestiona.

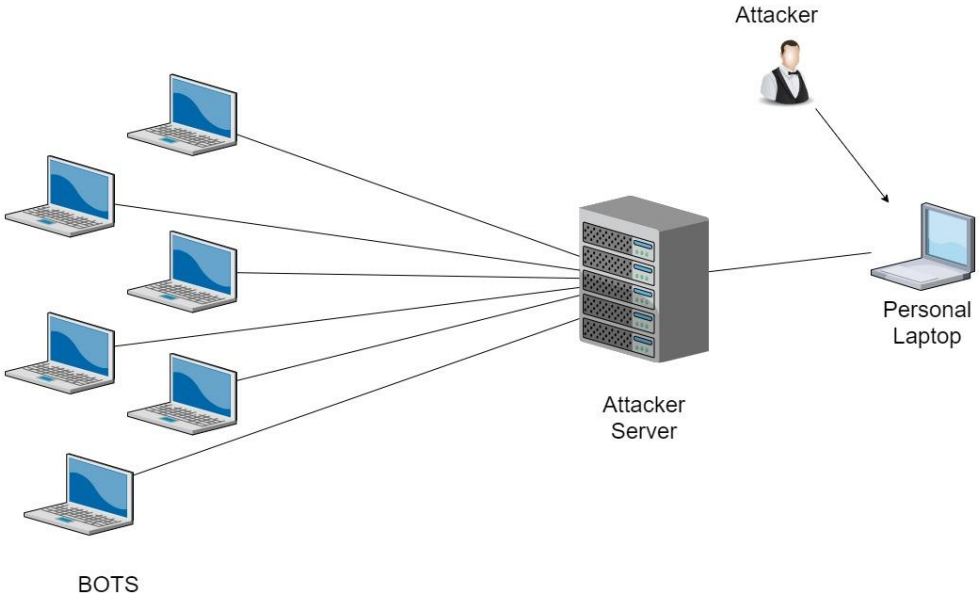


Figura 1: Arquitectura botnet centralizada.

2. Arquitectura P2P (peer-to-peer): Este tipo de botnet es más compleja, ya que no hay un servidor central que gestione la botnet, si no que los propios bots comunican las órdenes a otros bots.

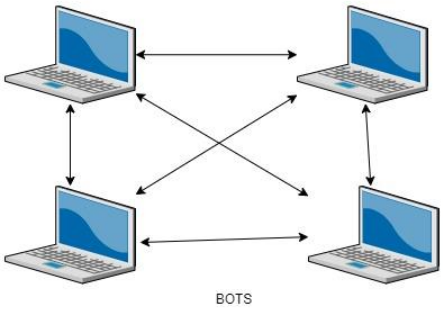


Figura 2: Arquitectura Botnet P2P.

2.3.1. Comparativa

Existen algunas aplicaciones con los fines anteriormente mencionados, como puede ser *Blacknet*, una herramienta avanzada que permite el control y monitorización de diferentes máquinas, así como sacar capturas de pantalla de los bots, cuenta con un *keylogger* incorporado, es capaz de robar datos del navegador de la víctima, así como contraseñas o cookies de sesión.

Si realizamos una comparativa de BotCommander con herramientas existentes, podemos darnos cuenta que ésta es mucho más simple y quizá no tenga tantas funcionalidades ya que son empresas grandes las que desarrollan este tipo de software por lo que cuentan con más recursos, no obstante BotCommander es ligera por lo que el tiempo de despliegue y ejecución es de apenas unos segundos, está escrita en Python, un lenguaje robusto, con mucho soporte y buena legibilidad (si necesitamos realizar algún ajuste en el código fuente es relativamente sencillo hacerlo), además de la ventaja que brinda en cuanto a su portabilidad y soporte para casi todos los sistemas operativos.

Otra de las ventajas destacables es que garantiza un alto nivel de seguridad y confidencialidad en las comunicaciones ya que incorpora el protocolo TLS.

Es una herramienta muy completa ya que posee dos módulos principales (el módulo cliente y el módulo servidor) y cuenta con un generador de payload dinámico que permite crear cargas útiles de infección teniendo en cuenta el sistema operativo de la víctima y las direcciones de acceso del servidor del atacante, todo ello simplificando al máximo la interacción con el usuario. En la parte de infección, BotCommander ha implementado técnicas propias de persistencia para que el malware persista y sobreviva en la máquina, lo que hace que sea indetectable por las firmas de antivirus y permita realizar todas las acciones comandadas por el servidor central en segundo plano.

2.4. Historia

Las botnets, o redes de bots, han sido una preocupación constante en la era digital. Su historia se remonta a los primeros días de Internet, cuando los hackers comenzaron a utilizar programas maliciosos para controlar computadoras remotamente. Estas redes de bots se convirtieron en una de las armas más poderosas en manos de los ciberdelincuentes.

La historia de las botnets comenzó a principios de la década de 1990, con el auge de los virus informáticos y los troyanos. Estos programas maliciosos permitían a los atacantes tomar el control de las computadoras infectadas y utilizarlas para llevar a cabo actividades ilegales, como el envío masivo de correos no deseados (spam) o ataques de denegación de servicio distribuido (*DDoS*).

A medida que Internet crecía, las botnets evolucionaron para aprovechar la falta de seguridad en dispositivos conectados, como routers, cámaras web y dispositivos de Internet de las cosas (IoT). Los atacantes descubrieron que podían infectar estos dispositivos con malware y agregarlos a sus redes de bots, lo que les daba un mayor poder y alcance.

En la última década, las botnets han alcanzado niveles sin precedentes en términos de tamaño y sofisticación. Ejemplos notorios incluyen la botnet Mirai, que en 2016 llevó a cabo un ataque masivo contra proveedores de servicios de Internet (ISPs) y sitios web populares, dejando a millones de usuarios sin acceso a Internet.

A medida que las defensas contra las botnets han mejorado, los ciberdelincuentes han buscado nuevas formas de evadir la detección. Han utilizado técnicas como el cifrado de comunicaciones y el uso de infraestructura descentralizada para dificultar su rastreo y desmantelamiento.

Hoy en día, las botnets continúan representando una amenaza seria para la seguridad en línea. Los actores maliciosos las utilizan para realizar ataques cibernéticos, robar información personal y financiera, propagar malware y llevar a cabo actividades ilícitas en la web oscura.

En respuesta a esta creciente amenaza, los investigadores en seguridad y las autoridades gubernamentales han intensificado sus esfuerzos para combatir las botnets. Se han establecido colaboraciones internacionales y se han desarrollado herramientas y técnicas avanzadas para detectar y neutralizar estas redes maliciosas.

A medida que avanza la tecnología y evolucionan las estrategias de los ciberdelincuentes, la historia de las botnets sigue escribiéndose. La lucha contra estas redes de bots continúa, con el objetivo de mantener la seguridad en línea y proteger a los usuarios de Internet de sus impactos perjudiciales.

En la historia de internet, existieron múltiples Botnet exitosas. Veamos algunas:

Gameover ZeuS

Era una botnet peer-to-peer que se aprovechó de un malware más antiguo, el ZeuS Trojan, que infectó más de 3,6 millones de dispositivos y fue objeto de investigación internacional del FBI.

ZeroAccess

El rootkit ZeroAccess fue el principal método usado para forzar a computadores Windows a participar de la botnet. Este se propagó de forma agresiva por medio de ingeniería social y ataques de adware, y consiguió infectar cerca de 9 millones de dispositivo

2.5. Usos habituales de las botnets

Estas redes son usadas en general para generar dinero a través de usos que generan dinero a sus controladores [2]. Entre los usos más comunes están:

Ataques de denegación de servicio distribuidos (DDoS)

Este tipo de ataques consiste en inundar a la máquina víctima de peticiones hasta llegar al punto de la incapacidad de procesarlas y que la máquina se cuelgue, con el fin de dejarla incomunicada y que deje de prestar servicio.

Son de los ataques más peligrosos y con más difícil mitigación dada la dispersión geográfica de los ordenadores que componen estas botnet, resulta casi imposible encontrar un patrón de máquinas que están realizando las solicitudes, una forma de intentar solucionar este tipo de ataques es el filtrado de paquetes mediante un firewall, pero no siempre es efectivo y depende totalmente de las reglas y configuración que se implementen en el mismo.

Ataque de fuerza bruta

Al tener gran capacidad de computación disponible para el atacante, pueden aprovechar esta ventaja para coordinar ataques de diccionario a sitios web donde pretenden probar múltiples combinaciones de credenciales con la finalidad de craquear algún acceso.

Lanzamiento de campañas de Spam

Otro uso muy frecuente es que una botnet se utilice para enviar spam a direcciones de correo electrónico. Normalmente los administradores de las botnet venden estos servicios a empresas publicitarias.

Minería de Bitcoins y Robo de Bitcoins

En la nueva era en la que vivimos y el surgimiento de las criptomonedas, los ciberdelincuentes han encontrado una nueva forma lucrativa para ganar dinero, la cual consiste en minar criptomonedas (realizar cálculos para obtener criptomoneda) a través de los ordenadores que tienen infectados en la botnet, ahorrándose así en costes hardware y en consumo de energía.

También es posible robar credenciales de las wallet personales de bitcoin permitiendo al atacante robar el saldo de la cuenta de las personas cuyo equipo está comprometido. En el caso de la red Pony, sustraía información de los equipos infectados. Se estima que con esta red se obtuvieron credenciales (usuario/password) de al menos 2 millones de equipos.

Fraudes publicitarios

Normalmente, los servicios de anuncios en Internet pagan a los administradores de las webs donde figuran en función de varios factores, entre los que se incluye la visualización del anuncio y el hacer clic en el mismo. Los ciberdelincuentes ordenan a los bots que visiten sus propios sitios web y hagan clic en los anuncios para generar beneficios económicos de manera fraudulenta.

Vías de Entrada a una Botnet

Es importante destacar que no hay una sola manera de infectar un dispositivo y que este pase a formar parte de una botnet, es una cuestión que depende de múltiples factores, ya sea la plataforma o sistema operativo, las tecnologías que utilice el atacante, etc.

No obstante sí que se han descubierto técnicas comunes de infección encontradas en análisis de dispositivos.

Existe algo que todas las técnicas comparten, y es la necesidad de ejecución de un programa malicioso en el equipo víctima.

Los diferentes tipos de infección de dispositivos en función de la interactividad que requiere el malware se pueden clasificar de la siguiente manera:

- Infección activa: En este tipo de infección, el malware requiere una interacción activa del usuario para propagarse. Por ejemplo, un usuario puede ser engañado para que descargue un archivo adjunto infectado en un correo electrónico o para que haga clic en un enlace malicioso. Una vez que se ejecuta el archivo o se accede al enlace, el malware se instala y se propaga en el dispositivo.

- Infección pasiva: En este caso, el malware puede infectar el dispositivo sin la interacción directa del usuario. Esto puede ocurrir a través de vulnerabilidades de seguridad en el sistema operativo o firmware, también puede suceder si se poseen configuraciones por defecto como contraseñas y usuarios de fábrica (lo que sucede en algunos dispositivos IoT).

- Infección por descarga no autorizada: En este tipo de infección, el malware se descarga e instala en el dispositivo sin el conocimiento o consentimiento del usuario. Esto puede ocurrir cuando un usuario visita un sitio web malicioso o comprometido que descarga automáticamente el malware en segundo plano.

- Infección por interacción social: Este tipo de infección aprovecha la ingeniería social para engañar al usuario y lograr que realice acciones que permitan la entrada del malware en el dispositivo.

Capítulo 3.

Planificación del Proyecto.

3.1. Metodología de Trabajo

El proyecto cuenta con un tiempo de desarrollo de cuatro meses, y las tecnologías que se van a utilizar son desconocidas. Por otro lado, el software que queremos desarrollar tiene una gran responsabilidad educativa y sobre todo la correcta arquitectura y construcción del servidor central es crítica ya que miles de máquinas dependerán de él, dada la naturaleza centralizada que hemos escogido para crear la botnet.

Mencionados los motivos anteriores, debemos construir un software de calidad en un tiempo de desarrollo relativamente corto y con tecnologías por ahora desconocidas, en consecuencia, es altamente recomendable escoger una metodología de trabajo que sea capaz de adaptarse a los cambios, nos proporcione una buena organización con el equipo de desarrollo, y además permita la generación de software funcional en poco tiempo con la finalidad de retroalimentar al cliente y a los desarrolladores.

Cuando las tecnologías son desconocidas o es la primera vez que se trabaja con ellas, considero que la mejor estrategia es ir proponiéndose pequeños Hitos e ir dando pasos que obtengan como resultado artefactos funcionales (modelo basado en iteraciones), esto es importante para la continua motivación del equipo.

Se ha escogido el marco de trabajo ágil Scrum [1], debido a que es la que mejor se adapta a los requerimientos del proyecto.

3.1.2. Scrum

¿Qué es Scrum?

Scrum es un marco de gestión de proyectos de metodología ágil que ayuda a los equipos a estructurar y gestionar el trabajo mediante un conjunto de valores, principios y prácticas.

Conceptos Esenciales

Scrum [5] posee tres principios que aseguran el éxito de un proyecto:

Transparencia

En un entorno transparente, los equipos están conscientes de los desafíos que enfrentan sus compañeros. Las reuniones regulares cara a cara entre los miembros del equipo y los responsables del proyecto evitan la falta de comunicación y los obstáculos en la transferencia de información.

Reflexión

Se incluyen momentos frecuentes de reflexión en el proceso, para que los miembros del equipo puedan revisar los progresos realizados. Los gerentes de proyecto utilizan la información obtenida en estas reuniones de revisión para hacer estimaciones y planificar para el futuro. Como resultado, los proyectos pueden llevarse a cabo de manera más eficiente, dentro del presupuesto y en el tiempo previsto.

Adaptabilidad

Los miembros del equipo pueden reorganizar las tareas según los cambios en los requisitos de los clientes. Deciden qué tareas se completan primero y cuáles se posponen para más adelante.

Valores de Scrum

Los equipos de trabajo de que implementan Scrum, deben de tener en cuenta los siguientes 5 valores para un correcto uso de la metodología:

- Compromiso
- Valentía
- Concentración
- Actitud receptiva
- Respeto

3.1.3. Funcionamiento de Scrum

Los equipos deben de ser auto organizados y deben de aportar valor al cliente en pequeños períodos de tiempo definidos como Sprints o incrementos.

Scrum define los roles, artefactos y objetivos de cada Sprint [3].

Artefactos

Product Backlog

Es una lista de cosas previstas para hacer y que servirán para alcanzar los objetivos del producto. Es importante revisarlo constantemente para ajustar las prioridades. Es lo que hace que la metodología sea adaptable a los cambios. El Product Owner será el encargado de actualizarlo.

Sprint Backlog

Se refiere a una lista de elementos escogidos del Product Backlog para realizar en el sprint y que el equipo de desarrollo debe completar.

Incremento

Se considera un avance, logro o hito en el proyecto. Se puede decir que es el producto final y utilizable que se genera al final de un sprint.

Es flexible y el equipo de desarrollo puede acomodar a sus necesidades los objetivos o tareas que se llevarán a cabo en el sprint

Roles de Scrum

- Product Owner: Su tarea principal es asegurarse que el equipo de desarrollo proporcione el mayor valor posible al negocio. Conoce lo que el cliente quiere y se encarga de transmitírselo al equipo.
- Scrum Master: Lidera el equipo, se asegura el correcto uso de scrum y es el encargado de formar y optimizar al personal del Development Team.
- Development Team: Está compuesto por el equipo de desarrollo que será el que efectúe el trabajo para alcanzar el producto final.

Eventos

Podemos considerar a los eventos como conjuntos de reuniones que se traducen en bloques de tiempo establecidos y que se dan a lo largo de las distintas etapas de desarrollo del proyecto, para organizar el trabajo.

Sprint

El Sprint es el periodo de tiempo durante el cual el equipo trabaja con el objetivo de completar o conseguir un incremento. La duración aproximada debe de ser entre dos a cuatro semanas, no obstante, ésta puede variar si las necesidades del proyecto lo requieren.

Por regla general, la complejidad del trabajo debe de ser inversamente proporcional a la duración del sprint.

Daily Scrum o Reunión diaria

Es una breve reunión en la que los miembros del equipo comunican el estado en el que se encuentra el trabajo, con la finalidad de coordinarlo o solucionar algún problema que pueda haber surgido.

Revisión del Sprint (Sprint Review)

Cuando el Sprint finaliza, el equipo se reúne con las partes interesadas para revisar el trabajo que ha sido concluido. El product owner ajustará el producto backlog en función del sprint en curso.

Retrospectiva del Sprint (Sprint Retrospecive)

El equipo de trabajo se reúne para analizar y documentar sobre cómo ha transcurrido el sprint y en un futuro poder mejorar.

3.2. Planificación

Como ha sido mencionado en el apartado inicial del capítulo, se ha optado por la metodología ágil Scrum siendo necesaria una adaptación, ya que el desarrollo ha sido realizado únicamente por una persona.

En otras palabras, he asumido los roles de Scrum Máster, Product Owner y Development Team, siendo este último el que más presente ha estado.

En ocasiones, he tenido que asumir el de Product Owner, en concreto para realizar la especificación del Product Backlog y definir los Sprint Backlog.

El rol de Scrum máster siempre ha estado presente en todas las etapas del proyecto, asegurando el correcto uso de la metodología, analizando bloqueos y la búsqueda de soluciones.

El Product Backlog y los Sprint Backlog han sido gestionados mediante la herramienta Trello, desde la cual se han definido, actualizado y categorizado las tareas a desarrollar.

Los requisitos del proyecto han sido recopilados en forma de historias, que, a su vez se desglosan en tareas procediendo a su posterior reparto entre los sprints del proyecto y los roles del equipo.

La fecha inicial del proyecto es el 1 de marzo de 2023 y la fecha prevista para su finalización es el 30 de junio de 2023 (16 semanas aproximadamente), y como el Trabajo de fin de grado tiene un valor de doce créditos académicos equivalentes a 300 horas de trabajo en concordancia con la guía docente, en la medida de lo posible la duración del proyecto deberá ajustarse a los requisitos temporales mencionados.

Con una duración de 3 semanas aproximadamente cada Sprint podemos deducir que habrá 6 sprints, el primero empezará el 01 de marzo y se prevé que la fecha de finalización del último sea el 30 de junio del 2023.

A continuación se muestra una tabla con la planificación del proyecto:

Sprint	Fecha de Inicio	Fecha Fin
1	01/03/2023	19/03/2023
2	20/03/2023	09/04/2023
3	10/04/2023	30/04/2023
4	01/05/2023	21/05/2023
5	22/05/2023	11/06/2023
6	12/06/2023	30/06/2023

Tabla 1 Planificación inicial del proyecto

3.2.1. Planificación del Trabajo.

En concordancia con los requisitos y objetivos del proyecto, se han identificado las siguientes historias de trabajo:

H-01 Diseño de un método de comunicación en red.

> Esta historia comprende todo el proceso de búsqueda de información y aprendizaje sobre las tecnologías necesarias para crear una comunicación funcional mediante sockets utilizando Python. Será necesario también un proceso formativo sobre las botnets con la finalidad de entender sus objetivos y algunas metodologías en uso.

H-02 Implementación del socket Cliente.

> Esta historia incluye una etapa de aprendizaje en arquitecturas cliente-servidor más bien centrada en el rol del cliente. Se programará el script de comunicación, así como la secuencia de instrucciones que debe de realizar para su correcto funcionamiento en el intercambio de mensajes con el servidor remoto.

H-03 Implementación del socket servidor.

> El desarrollo del servidor debe de hacerse de manera simultánea al del cliente, ya que ambas piezas van a complementarse. Dicho proceso corresponderá con la implementación del servidor socket en Python que cumplirá el cometido de gestionar a los bots conectados. También se tratarán aspectos de rendimiento y robustez.

H-04 Gestión de las comunicaciones.

> El principal objetivo de esta historia será el diseño de métodos que permitan gestionar de una manera correcta y eficaz las comunicaciones entre las máquinas, proporcionando garantías de funcionamiento en condiciones adversas y la securización mediante el protocolo SSL para que los datos enviados en los sockets TCP no se vean comprometidos.

H-05 Envío y recepción de archivos.

> Corresponde al soporte de las funcionalidades de intercambio de archivos entre los clientes y el servidor. Se implementarán métodos a bajo-medio nivel que permitan trabajar con archivos remotos según el administrador del servidor desee.

H-06 Creación del generador de payloads.

> Con la finalidad de garantizar la completitud del proyecto, se creará también una herramienta que sea capaz de generar payloads dinámicos de infección , independientemente del tipo de sistema del que se trate y de la dirección del servidor del atacante.

H-07 Documentación del proyecto.

Se deberán de recoger y documentar todas las partes del proyecto en una memoria, que cumplirá con los requerimientos del grado de Ingeniería Informática de Servicios y Aplicaciones de la Universidad de Valladolid, siguiendo así una estructura predefinida.

El presente documento reflejará cada uno de los aspectos del proceso de desarrollo, incluyendo la contextualización, planificación, costes y una sección técnica que sea capaz de plasmar con exactitud cómo fue llevado a cabo.

Cada historia está formada por un conjunto de tareas, son la unidad mínima de trabajo que se reflejará en los Backlog (Sprint y Product). Se repartirán equitativamente entre los 6 Sprint, no obstante, debido a la fase inicial de planificación en la que nos encontramos, es remarcable que podrían surgir cambios en la especificación de las tareas si las necesidades y objetivos del proyecto lo requieren.

A continuación se presentará el listado de tareas provisionales que componen cada historia, así como el reparto de ellas en cada sprint:

Tarea	Descripción	Puntaje
T-01-01	*Investigación sobre las botnets.* Realizar un estudio de las botnets, y sus metodologías de operación.	2
T-01-02	Formación sobre sockets y las librerías en Python.	2
T-01-03	Reunión inicial con el tutor y revisión de la idea del proyecto	1
T-01-04	Diseño de la arquitectura y modelado de la misma.	3
T-01-05	Primeras pruebas haciendo scripts de sockets en Python.	1
Tareas de la Historia de usuario 01		

Tabla 2:Tareas de la historia de usuario 01

Tarea	Descripción	Puntaje
T-02-01	Elaboración de un guión de la memoria.	1
T-02-02	Creación del entorno de desarrollo.	1
T-02-03	Reunión inicial de Sprint con el tutor y actualización del Sprint Backlog.	1
T-02-04	Programación de un script funcional de un cliente en Python.	3
T-02-05	Testing y corrección del script cliente.	2
Tareas de la Historia de usuario 02		

Tabla 3: Tareas de la historia de usuario 02

Tarea	Descripción	Puntaje
T-03-01	Reunión inicial de Sprint con el tutor y actualización del Sprint Backlog.	1
T-03-02	Diseño e implementación del socket servidor.	4
T-03-03	Estudio de las Reverse Shell.	1
T-03-04	Programación de la funcionalidad Reverse Shell remota en cliente y servidor.	3
T-03-05	Programación de la funcionalidad para lanzar campañas a los bots.	2
Tareas de la Historia de usuario 03		

Tabla 4:Tareas de la historia de usuario 03

Tarea	Descripción	Puntaje
T-04-01	Reunión inicial de Sprint con el tutor y actualización del Sprint Backlog.	1
T-04-02	Reestructuración del código e implementación de Excepciones para manejar situaciones adversas y aumentar la fiabilidad de la herramienta.	2
T-04-03	Estudio del protocolo SSL.	1
T-04-04	Implementación de SSL en las comunicaciones.	2
T-04-05	Implementación de funciones manejadoras de ordenes en cliente y servidor, con la finalidad de hacer más escalable y mantenible el código.	3
Tareas de la Historia de usuario 04		

Tabla 5:Tareas de la historia de usuario 04

Tarea	Descripción	Puntaje
T-05-01	Reunión inicial de Sprint con el tutor y actualización del Sprint Backlog.	1
T-05-02	Diseño de un protocolo de envío y recepción de archivos a través de sockets.	3
T-05-03	Implementación de los métodos de subida y bajada de ficheros remotos.	3
T-05-04	Test y corrección de las funcionalidades subida y bajada de ficheros.	2
Tareas de la Historia de usuario 05		

Tabla 6:Tareas de la historia de usuario 05

Tarea	Descripción	Puntaje
T-06-01	Reunión inicial de Sprint con el tutor y actualización del Sprint Backlog.	1
T-06-02	Estudio de técnicas de persistencia de malware en sistemas operativos.	1
T-06-03	Estudio de Daemons en Windows y Linux y de ejecución silenciosa.	1
T-06-04	Programación de un generador de payloads dinámico para BotCommander.	4
T-06-05	Test y corrección global del software.	1
Tareas de la Historia de usuario 06		

Tabla 7:Tareas de la historia de usuario 06

Tarea	Descripción	Puntaje
T-07-01	Elaboración de la introducción y los objetivos del proyecto	1
T-07-02	Confección de una planificación del trabajo y redacción de la metodología. Elaboración de Presupuestos.	2
T-07-03	Documentación del estado del arte.	1
T-07-04	Documentación de los sockets cliente y servidor.	1
T-07-05	Documentación sobre las tecnologías utilizadas.	1
T-07-06	Documentación del flujo de trabajo de la aplicación.	1
T-07-07	Diseño y documentación de un plan de riesgos	1
T-07-08	Elaboración del coste final del proyecto y confección de las conclusiones.	1
Tareas de la Historia de usuario 07		

Tabla 8:Tareas de la historia de usuario 06

En total se obtienen 63 puntos de historia que serán repartidos entre los 6 sprints que componen el proyecto.

Debido al estudio que hemos realizado y basándonos en nuestra experiencia, se ha considerado que *1 punto de historia equivale a 5 horas de trabajo útil*, es adecuado para el tipo de proyecto del que se trata. La estimación indica que la duración total será de 315 horas, considerando una dedicación promedio de unas 40 horas semanales (lo habitual en el mundo empresarial), obtenemos unas 8 horas diarias en una jornada laboral de 5 días.

Dado que el trabajo de fin de grado estima unas 300 horas aproximadas de dedicación, estaríamos dentro de los márgenes aceptables y viables para la realización (en cuanto a tiempos se refiere) del proyecto, desviándonos 15 horas por encima.

Es remarcable también, que en el conjunto de tareas no se especifican las reuniones retrospectivas ya que se realizarán en las reuniones iniciales de los sprint con el tutor.

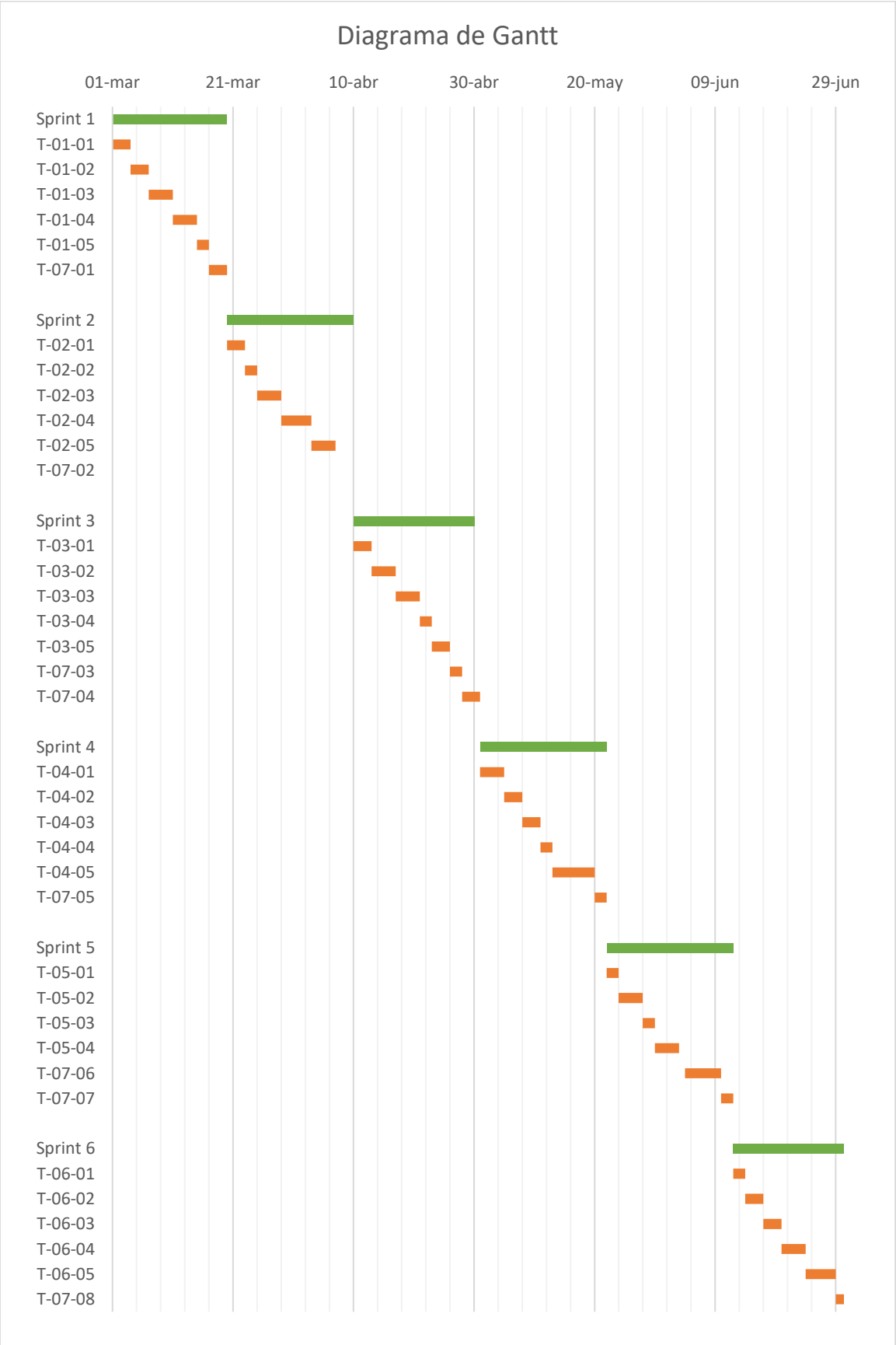
Tampoco está contemplada la defensa del proyecto y es posible que en la práctica se vean modificadas las tareas.

Sprint	Inicio	Fin	Tareas	Total Puntaje
1	01/03/2023	19/03/2023	T-01-01 T-01-02 T-01-03 T-01-04 T-01-05 T-07-01	10
2	20/03/2023	09/04/2023	T-07-02 T-02-01 T-02-02 T-02-03 T-02-04 T-02-05	10
3	10/04/2023	30/04/2023	T-03-01 T-03-02 T-03-03 T-03-04 T-03-05 T-07-03 T-07-04	13
4	01/05/2023	21/05/2023	T-04-01 T-04-02 T-04-03 T-04-04 T-04-05 T-07-05	10
5	22/05/2023	11/06/2023	T-05-01 T-05-02 T-05-03 T-05-04 T-07-06 T-07-07	11
6	12/06/2023	30/06/2023	T-06-01 T-06-02 T-06-03 T-06-04 T-06-05 T-07-08	9
		total		63

Tabla 9: Calendario previsto al inicio del proyecto

3.2.2. Diagrama de Gantt [13]

Figura 3 Diagrama de Gantt de la planificación del trabajo



3.3. Presupuestos.

3.3.1. Recursos Técnicos.

El proyecto será llevado a cabo mediante un ordenador portátil personal modelo Asus TUF GAMING fx505dt, tanto la parte documental como la programación del código. Debido a los requerimientos del proyecto, son necesarios unos recursos mínimos que otorguen el rendimiento necesario para su ejecución, por lo que se ha decidido la compra de un equipo que posee como características hardware un procesador AMD Ryzen 5 a 3.7 GHZ, la memoria RAM es DDR4 con una capacidad de 16 GB acompañados de un disco duro SDD de 500GB.

Según la información recopilada a cerca de la vida útil de un ordenador de similares características el resultado ha sido de 5 años, teniendo en cuenta la planificación del proyecto, la computadora será usada 8 horas diarias durante 4 meses, resultando en 6.67% de consumo de vida útil.

En lo que a software se refiere, no habrá costes significativos ya que el desarrollo hará uso exclusivo de tecnologías Open Source de carácter gratuito.

Por otro lado, será necesaria la adquisición de un entorno virtual en la nube que permita desplegar el servidor central con conexión a internet, la Universidad de Valladolid nos proporcionó una máquina Ubuntu con 4 núcleos de CPU y 4GB de RAM sin coste alguno.

Será necesaria una conexión a internet para poder conectarse al servidor central y poder realizar pruebas a medida que vayamos desarrollando la aplicación, aparte de poder trabajar con GitHub en el sistema local y en la máquina virtual.

Detalle del presupuesto:

Recurso	Coste/ud	%	Total
Ordenador portátil ASUS	750,00 €	6,67	50,02
Máquina virtual Uva			00,00
Distribución Anaconda			00,00
GitHub Desktop			00,00
Windows 11 Pro (incluido)			00,00
Office 365 (Licencia de la UVA)			00,00
		Total:	50,02 €

Tabla 10: Recursos técnicos

3.3.2. Recursos Humanos.

En este punto es importante reseñar que un proyecto real es probable que el equipo de desarrollo esté formado por más de una persona, a diferencia de éste, que ha sido unipersonal y los recursos humanos se han reducido al autor únicamente.

No obstante, he tenido que asumir diferentes roles (con sus respectivas variaciones salariales) dependiendo de la tarea a desempeñar o la fase del proyecto.

El presupuesto se calculará realizando una división de tareas entre los diferentes roles que componen el proyecto y calculando el número de horas que se necesitarán de cada uno.

En la siguiente tabla se muestra el reparto de tareas, así como el cómputo de horas que le corresponden a cada uno:

Rol	Tareas	Puntos totales	Horas totales
Jefe de proyecto	T-01-03, T-02-01, T-02-03, T-03-01, T-04-01, T-05-01, T-06-01, T-07-01, T-07-02, T-07-07, T-07-08	12	60h
Analista	T-01-01, T-01-04, T-03-03, T-04-03, T-05-02, T-06-02, T-06-03, T-07-03, T-07-05	14	70h
Desarrollador Python	T-01-02, T-01-05, T-02-02, T-02-04, T-02-05, T-03-02, T-03-04, T-03-05, T-04-02, T-04-04, T-04-05, T-05-03, T-05-04, T-06-04, T-06-05, T-07-04, T-07-06	37	185h
	Total	63	315 h

Tabla 11: Reparto de tareas en Sprints

Para un jefe de proyecto con experiencia media, el salario bruto por hora [20] podría estar en el rango de 30 a 50 euros, optaremos por la media de ese rango, considerando un coste bruto de 40 euros la hora.

En el caso del analista [18] estaría entre 20-30 euros la hora y el desarrollador Python 15 [19] según datos recopilados.

Para calcular los costos de contratación de los empleados a la empresa [6] , hay que considerar los aspectos correspondientes a los salarios, impuestos y cargas sociales.

1. Salario: El primer paso es determinar los salarios mensuales brutos para cada uno de los roles. Este puede variar en función de múltiples factores como la experiencia de los candidatos y la ubicación geográfica dentro del país.
2. Impuestos y cargas Sociales: En España, las empresas deben contribuir a la Seguridad Social, que incluye el pago de cotizaciones de seguridad social y otras prestaciones. El porcentaje de cotización varía según la categoría profesional y otros factores. A modo de estimación, se podría calcular un 30% adicional sobre el salario bruto para cubrir estos costos.

Rol	Salario bruto /hora	Horas	Total Bruto	Costes Sociales (30%)	Coste Total (Empresa)
Jefe de proyecto	40,00	60	2400,00	720,00	3120,00
Analista	25,00	70	1750,00	525,00	2275,00
Desarrollador Python	15,00	185	2775,00	832,00	3607,00
		Total			9002,00 €

Tabla 12: Recursos humanos

Por lo que el coste total estimado del proyecto será **9052,02 €** contando los recursos humanos y técnicos.

Capítulo 4

Desarrollo del Proyecto.

4.1. Tecnologías utilizadas.

En esta sección se detallarán las tecnologías y recursos software que han sido utilizadas en el desarrollo del proyecto.

Python 3

Python 3 es un lenguaje de programación versátil y de alto nivel que se utiliza ampliamente en proyectos de ciberseguridad. Su popularidad en este campo se debe a varias ventajas que ofrece.

En primer lugar, Python 3 es conocido por su legibilidad y sintaxis clara. Esto facilita el desarrollo de código limpio y comprensible, lo cual es fundamental en la seguridad informática, donde la precisión y la transparencia son esenciales.

Por otro lado, Python 3 cuenta con una amplia variedad de bibliotecas y módulos dedicados a la ciberseguridad. Estas bibliotecas ofrecen funcionalidades específicas, como criptografía, análisis de vulnerabilidades, auditoría de seguridad y manipulación de protocolos de red, lo que permite a los desarrolladores implementar soluciones robustas de manera eficiente.

Otra ventaja significativa de Python 3 en el ámbito de la ciberseguridad es su naturaleza multiplataforma. El lenguaje es compatible con los principales sistemas operativos, lo que ha facilitado el desarrollo y la ejecución de herramientas y scripts en diferentes entornos.

Además, Python 3 cuenta con una comunidad activa y comprometida, lo que se traduce en una gran cantidad de recursos, documentación y ayuda disponibles en línea. Esto facilita el aprendizaje y la resolución de problemas, lo que es especialmente útil en proyectos de ciberseguridad, donde la rapidez y la eficacia son críticas.

4.1.2. Librerías de Python

A continuación se realizará un esbozo de las librerías que han sido de gran utilidad para conseguir los objetivos del proyecto. Cabe destacar que siempre ha primado el uso de módulos nativos de Python con la finalidad de necesitar el mínimo de dependencias a la hora de realizar el despliegue.

Socket

Proporciona una interfaz para la programación de comunicaciones de red [7]. Permite la creación de aplicaciones cliente-servidor y facilita la transferencia de datos a través de conexiones de red. Permite trabajar con sockets a bajo nivel.

Sin duda ha sido el pilar principal del proyecto ya que todas las comunicaciones están basadas en sockets TCP.

Thread

Permite la ejecución simultánea de múltiples hilos de ejecución dentro de un programa, lo que brinda la capacidad de realizar tareas en paralelo y mejorar la eficiencia en la gestión de recursos.

Su presencia ha sido importante debido a la necesidad de ejecución de tareas concurrentes lo que ha permitido mejorar el rendimiento y potencia del servidor a la hora de gestionar un gran conjunto de máquinas.

Art

Es una biblioteca para la generación de arte ASCII en la terminal, permitiendo crear representaciones visuales de texto y gráficos utilizando caracteres.

Colorama

Facilita la manipulación de colores y estilos en la salida de texto en la terminal, lo que mejora la presentación visual y la legibilidad de los mensajes.

Os

Es una librería nativa de Python que proporciona funciones para interactuar con el sistema operativo, permitiendo acceder a funcionalidades como la gestión de archivos, directorios, variables de entorno y ejecución de comandos del sistema. Su principal ventaja es que facilita una comunicación entre Python y el sistema operativo abstrayendo los detalles y manera de interactuar con los diferentes sistemas operativos.

PsUtil

Permite obtener información sobre los procesos y la utilización de recursos del sistema, como el uso de la CPU, la memoria y la red, brindando herramientas para el monitoreo y la gestión de procesos. En el proyecto sólo ha sido utilizado para conseguir datos sobre el uso de recursos del servidor con el fin de monitorizarlo.

Time

Proporciona funciones para medir el tiempo y la gestión temporal en un programa, permitiendo realizar pausas, obtener la fecha y hora actual y calcular el tiempo transcurrido.

Es un módulo básico e importante si se quieren gestionar tareas a tiempo real. En nuestro caso, lo hemos utilizado para programar los intentos de conexión de los bots.

Ssl

[9] Proporciona funciones para la implementación de comunicaciones seguras utilizando el protocolo SSL/TLS, permitiendo la encriptación de datos y garantizando la integridad y confidencialidad de la información transmitida.

El uso que se le ha dado es indiscutible ya que ha permitido cifrar punto a punto las comunicaciones entre el servidor y los bots.

Subprocess

Permite la creación y control de subprocesos en un programa, lo que posibilita la ejecución de comandos externos y la comunicación entre procesos.

Ha sido otro de los pilares fundamentales en los que se apoya el proyecto. Mediante el módulo subprocess es posible crear una ventana de comunicación entre Python y el sistema operativo subyacente permitiendo ejecutar las órdenes que envía el servidor en los bots y recuperar el output.

Platform

Proporciona información sobre la plataforma en la que se está ejecutando el programa, incluyendo detalles del sistema operativo, la arquitectura y el entorno de ejecución. Aunque Python es multiplataforma, a veces es necesario discernir entre un sistema y otro.

Sys

Proporciona acceso a variables y funcionalidades específicas del intérprete de Python, permitiendo la interacción con el sistema, la manipulación de la entrada y salida estándar y la gestión de excepciones.

PyInstaller

Es una herramienta que permite crear ejecutables independientes de Python a partir de scripts, facilitando la distribución de aplicaciones sin necesidad de instalar un intérprete de Python. El papel que desempeña en el proyecto es compilar un fichero de Python en un .exe directamente ejecutable por Windows.

Pythonw

Es una variante del intérprete de Python diseñada para ejecutar scripts de manera silenciosa en entornos de interfaz gráfica, ocultando la consola de la terminal y mejorando la experiencia del usuario.

Gracias a este módulo he podido desarrollar mi propia técnica de ejecución silenciosa de procesos en Windows sin que los sistemas se seguridad salten ya que consideran que pythonw es una actividad legítima ya que hay aplicaciones que lo utilizan.

Trello

[14] En el marco de mi trabajo de fin de grado, he utilizado la herramienta Trello para gestionar y organizar eficientemente las tareas del proyecto. Trello es una plataforma digital de gestión de proyectos basada en tableros, listas y tarjetas, que me ha permitido visualizar y controlar de manera efectiva mi progreso.

Con Trello, he creado tableros para cada uno de mis proyectos, donde he podido organizar mis tareas en listas y categorizarlas por su estado o prioridad. Las tarjetas representan cada tarea específica, y he podido agregar descripciones, fechas de vencimiento, etiquetas y adjuntos relevantes a cada una de ellas.

GitHub

GitHub es una plataforma de desarrollo colaborativo de software basada en la nube. Proporciona un sistema de control de versiones distribuido utilizando Git y herramientas de colaboración para facilitar el trabajo en equipo en proyectos de programación.

Se pueden alojar repositorios que sirven para almacenar proyectos, normalmente de software. Los repositorios permiten a los equipos de desarrollo colaborar en el desarrollo de software, realizar un seguimiento de los cambios realizados en el código y mantener un historial completo de versiones.

En la práctica hemos utilizado *Github Desktop* el cual es un cliente de git disponible para todos los sistemas operativos. Ha permitido administrar el repositorio del proyecto desde las distintas máquinas que he utilizado (servidor de Valladolid y mi máquina local).

Draw.io

Draw.io es una herramienta de diagramación y de modelado software. Permite a los usuarios crear diagramas de flujo, diagramas de procesos, diagramas de red, diagramas de clases, diagramas de UML y muchos otros tipos de diagramas.

Con ella he creado todos los diagramas presentes en este documento, exceptuando el de Gantt.

Microsoft Office Word

Es un procesador de texto creado por la firma Microsoft el cual permite crear y editar documentos en una amplia variedad de formatos. La documentación del proyecto ha sido creada mediante este software, permitiendo así crear tablas de contenido, aplicar numeración de páginas, creación de secciones, formateo de texto e inserción de recursos en el documento.

Capítulo 5.

Análisis y diseño del sistema

5.1. Introducción

El lado más dificultoso de este proyecto será implementar una infraestructura que permita una comunicación punto a punto entre el servidor y los clientes, por ello nos hemos inspirado en el modelo clásico de arquitectura cliente-servidor ya que se ha considerado ideal para el intercambio de mensajes y con el que ya estamos familiarizados.

Por otro lado, al tratarse de una botnet centralizada, no hay muchas más opciones en cuento a arquitectura se refiere.

5.2. Características Cliente-Servidor

Ambos roles pueden actuar de manera independiente y realizar sus propias actividades.

Permite la independencia geográfica, es decir el cliente y servidor pueden estar en sitios distintos e incluso no es necesario el conocimiento de su ubicación para la comunicación.

Cada plataforma puede ser escalable independientemente. Los cambios realizados en las plataformas de los Clientes o de los Servidores, ya sean por actualización o por reemplazo tecnológico, se realizan de una manera transparente para el usuario final.

La interrelación entre el hardware y el software está basada en una infraestructura poderosa, de tal forma que el acceso a los recursos de la red no muestra la complejidad de los diferentes tipos de formatos de datos y de los protocolos.

Su representación típica es un centro de trabajo (PC), en donde el usuario dispone de sus propias aplicaciones de oficina y sus propias bases de datos, sin dependencia directa del sistema central de información de la organización

Ventajas

- Una de las ventajas clave es la centralización del control. En este tipo de arquitectura, el servidor es responsable de gestionar los accesos, los recursos y garantizar la integridad de los datos.

- La escalabilidad es otra ventaja significativa. En una arquitectura Cliente/Servidor, tanto los clientes como los servidores pueden aumentar su capacidad de forma independiente.

- El fácil mantenimiento es otro aspecto destacado de las arquitecturas Cliente/Servidor. Al distribuir las funciones y responsabilidades entre varios ordenadores independientes, se facilita la tarea de reemplazar, reparar, actualizar e incluso trasladar un servidor, sin afectar a los clientes de manera significativa o incluso sin afectarlos en absoluto.

- En cuanto a la seguridad, en las redes Cliente/Servidor, los clientes no tienen acceso directo a las direcciones IP de otros clientes, lo que dificulta el rastreo y el hackeo de los usuarios. Esto brinda una capa adicional de protección y privacidad en comparación con las redes P2P, donde la visibilidad de las direcciones IP es mayor.

Inconvenientes

- Cuando una gran cantidad de clientes envía peticiones simultáneas al mismo servidor, este debe gestionarlas y por ello puede verse saturado.
- A diferencia de la arquitectura P2P, cliente-servidor no es tan robusta, por lo que en un sistema distribuido, el papel del servidor es crucial y si éste falla, lo hará todo el sistema.

5.3. Arquitectura

El reparto de roles en cuanto a arquitectura se refiere será el siguiente:

El servidor central, comandará a los bots y éstos mismos serán las máquinas que han sido infectadas anteriormente con malware. Los bots adoptarán el rol de cliente (en términos de arquitectura) y el atacante controla el servidor.

El protocolo de comunicación seguirá las siguientes reglas:

1. Los clientes son quienes establecen conexión con el servidor central, ya que es menos intrusivo y permite levantar menos sospechas (nos apoyamos en el concepto de reverse Shell). En el ámbito de la ciberseguridad, uno de los objetivos principales de un ciber atacante es conseguir el control completo de una máquina remota, esto se consigue en líneas generales de dos formas:

    a. Estableciendo una Shell Reversa, que consiste en que la máquina objetivo se infecta con un tipo de malware para que se conecte a un servidor controlado por el atacante y éste pueda manejar la máquina mediante una Shell que recibe comandos remotos.
    b. Usando una Bind Reverse Shell. Se fundamenta en lo mismo que la anterior pero con una principal diferencia, y es que en este caso, es el atacante quien se conecta a la máquina victima por medio de un puerto que ha abierto anteriormente (es más ruidosa y un dispositivo firewall lo detectará más fácilmente).

    En el caso de este proyecto, se ha decidido apostar por el concepto de Reverse Shell ya que es menos intrusiva y más difícil de detectar por sistemas de seguridad ya que no requiere de la apertura de un servicio en la máquina infectada, lo que permite bypassear dispositivos de detección.

2. El cliente siempre se mantiene en una escucha constante, esperando que el servidor le indique una orden.
3. El cliente recibe una orden, la ejecuta y manda su resultado al servidor de atacante.
4. El servidor siempre testeará la conexión con el cliente antes de proceder con el envío de una orden.

Para una mayor comprehensión del problema se detalla gráficamente:

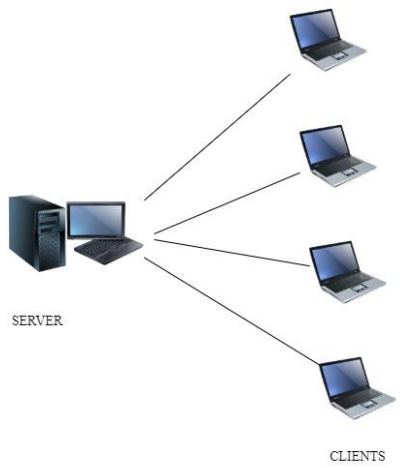


Figura 4: Arquitectura

5.2. Sockets

Un socket es una abstracción de software que permite la comunicación entre procesos en una red de computadoras. Básicamente, podemos decir que es una combinación de una dirección IP y un número de puerto que identifica un punto final de comunicación en una red.

Cuando dos programas desean comunicarse a través de una red, establecen una conexión utilizando sockets. Un programa actúa como un cliente y otro como un servidor. El cliente crea un socket y se conecta al socket del servidor utilizando la dirección IP y el número de puerto correspondientes. Una vez establecida la conexión, los programas pueden intercambiar datos a través del socket.

Pueden ser utilizados para la transferencia de todo tipo de información, ya sea la transmisión de datos en tiempo real, la transferencia de archivos o el intercambio de mensajes.

En la práctica, podemos diferenciar varios tipos de sockets, en función de la tecnología o protocolo que vayamos a utilizar, siendo así el socket TCP basado en utilizar el protocolo de la capa de transporte TCP (sockets orientados a conexión) y sockets UDP que emplean el protocolo UDP, más rápido que el anterior pero menos confiable ya que no es posible determinar el envío exitoso de un mensaje ni su recepción por parte del socket receptor.

En este caso, utilizaremos el modelo basado en TCP integrado en la arquitectura Cliente-Servidor.

5.2.1. Socket Python

Este módulo proporciona acceso a la interfaz BSD *socket*. Está disponible en todos los sistemas Unix modernos, Windows, MacOS y probablemente plataformas adicionales.

Características.

[8] Las características principales del módulo pueden resumirse en las siguientes:

1. Creación de sockets: Permite la creación y configuración de Sockets en python.
2. Tipos de socket: Podemos crear diferentes tipos de sockets según las necesidades de nuestro proyecto, como TCP (Transfer Control Protocol) y UDP (User Datagram Protocol).
3. Conexiones de servidor: Es posible crear sockets de servidor para escuchar conexiones entrantes. Puedes especificar el puerto en el que se escucharán las conexiones y, una vez que se establezca una conexión, puedes aceptarla y manejarla según tus necesidades.
4. Conexiones de cliente: Es posible crear sockets con el rol de cliente que sean capaces de conectarse a un servidor remoto dada una dirección física. Permite enviar y recibir datos a través de red.
5. Configuración de opciones: El módulo `socket` permite configurar varias opciones para los sockets, como el tiempo de espera, la reutilización de direcciones y el bloqueo o no bloqueo del socket.
6. Herramientas de resolución de nombres: socket también provee de mecanismos DNS para la resolución de nombres de dominio.
7. Control sobre las interfaces de red: permite controlar y gestionar los adaptadores de red del dispositivo.
8. Posee soporte para IPv6: es capaz de trabajar, si se configura correctamente, con ipv6.

Limitaciones.

Aunque es una herramienta muy útil y completa para las comunicaciones en red, a veces, puede resultar compleja la programación de sockets especialmente para aplicaciones que involucran una comunicación más sofisticada o protocolos complejos.

Por otra parte, el rendimiento de esta implementación puede verse afectado si se desea trabajar con grandes volúmenes de datos o en aplicaciones que requieren una baja latencia.

Otra de las limitaciones es la escalabilidad, y es que no posee soporte para la gestión de múltiples conexiones o balanceo de carga, por lo que si se necesita de alguna de estas funcionalidades, tendrá que programarse desde cero.

Principalmente está diseñado para utilizarse en python, por lo que la interoperabilidad puede verse afectada si se desea realizar una comunicación con sistemas que utilizan lenguajes de programación diferentes.

Por último, socket, trabaja con buffer de envío y de recepción estáticos, es decir, cuando vas a enviar un volumen de datos, la cantidad máxima que puedes enviar en un solo paquete está limitada al tamaño del buffer. Lo mismo sucede en la recepción, por lo que es necesario implementar métodos más sofisticados que determinen cuando ha finalizado un mensaje.

Funcionamiento.

El servidor se pone en modo de escucha para recibir conexiones entrantes de los clientes, que solicitan una aceptación. Es posible establecer reglas de aceptación según la información del cliente.

Una vez ha sido aceptado, la comunicación está basada en el envío-recepción, es decir, una de las partes envía un paquete que será recibido en el otro lado.

Hay que destacar que el flujo de comunicación se define mediante programación y a gusto del desarrollador, no obstante, por norma general sigue el siguiente esquema:

Se envía un paquete y el socket receptor estará programado para recibirlo.

La operación ***send*** permite el envío de datos a través del socket hasta un máximo definido en el tamaño del buffer, en este caso, la operación no es bloqueante, a excepción de la operación de recuperación ***recv*** que posibilita la recepción de datos que han sido enviados a través del socket. El socket se queda esperando, en estado de bloqueo hasta que recibe los datos.

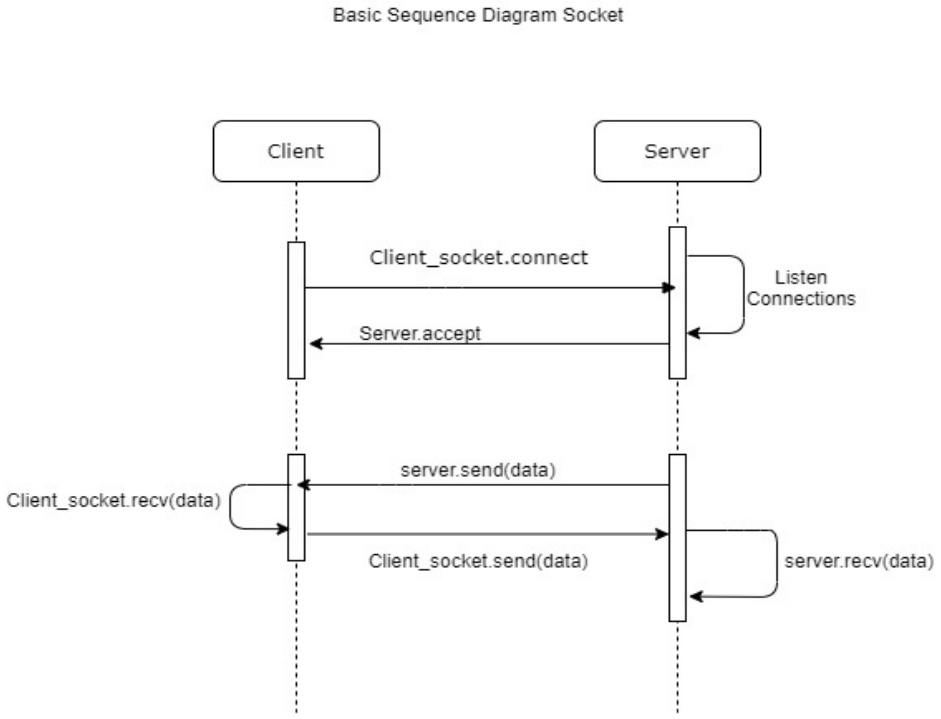


Figura 5: Diagrama de secuencia de un socket cliente-servidor.

En la práctica, utilizaremos este paradigma para llevar a cabo la comunicación de los bots y el servidor.

El bot actuará con rol de cliente siendo quién inicie la conexión hacia el servidor, que será el encargado de aceptarle.

Después el bot enviará información de su sistema operativo y arquitectura de manera que el administrador del servidor sepa en qué lenguaje tiene que enviarle las órdenes en consecuencia, a partir de ahí empezara un ciclo cerrado de conexión en la que los clientes esperan órdenes pasivamente.

5.3. Diseño del Escenario de Red.

En esta sección se explicará el diseño de red que se ha planteado:

El servidor central de comando se ubicará desplegado en una máquina virtual proporcionada por la universidad, dentro del dominio virtual.infor.uva.es y que será accesible a través de un puerto, en nuestro caso tenemos asignado el 65402.

La máquina contará con un sistema operativo Ubuntu 20.04 LTS basado en Linux. En ella será necesario tener instalado Python para que podamos desplegar el servidor correctamente. En el script del servidor tendremos que especificar una dirección física del adaptador de red de la máquina (no confundir con la dirección pública) y un puerto que se encuentre libre para desplegar el servicio.

Los Bots tendrán que conectarse mediante una dirección pública que sea visible desde internet, para ello será necesario habilitar un *port forwarding*, de manera que las peticiones que lleguen a la dirección *virtual.infor.uva.es:65402* sean redirigidas a la dirección interna de la máquina: 10.132.17.95:4566.

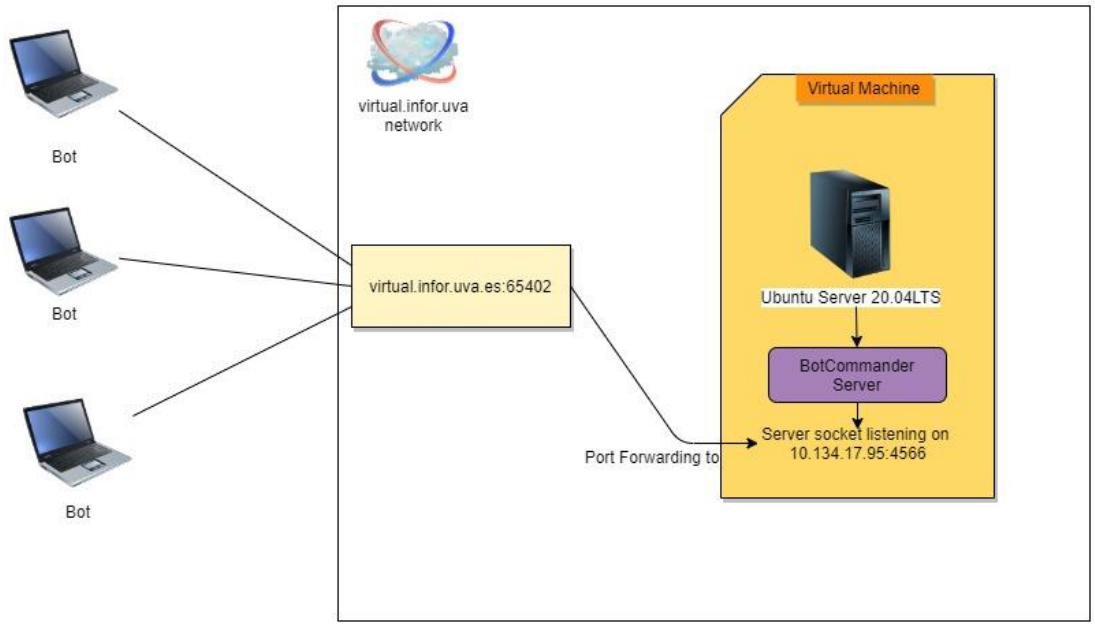


Figura 6: Diagrama del escenario de red diseñado.

5.4. Proceso de infección de dispositivos.

Debido a que aún nos encontramos en una etapa inicial de diseño, no se proporcionarán muchos detalles de implementación sobre la metodología a seguir en el proceso de infección para que una máquina llegue a convertirse en un Bot de la botnet, nos limitaremos a definir una serie de pautas o secuencias que se deben de seguir para conseguir el objetivo mencionado.

En primera instancia, la idea será crear un Payload o carga útil que servirá para abrir una puerta al malware en el dispositivo de la víctima, para ello, se ha pensado en crear un instalador "aparentemente legítimo" de Python que facilite su instalación a aquellos usuarios inexpertos. En este software colocaremos los ficheros que se encargarán de establecerse en la máquina y contarán con las instrucciones necesarias que garanticen una conexión con el servidor central.

Se ha elegido un instalador de Python como vía de entrada, ya que es necesario tenerlo instalado en la máquina para correr el script de comunicación, por lo que el payload instalará la versión de Python correcta para la máquina y de paso la infectará. Si ya se encuentra instalado, sólo infectará el dispositivo informando al usuario del estatus de la instalación.

5.5. Consideraciones de la Botnet.

En etapas tempranas del proyecto, se hizo una pequeña recopilación de conceptos que se tuvieron en cuenta en etapas posteriores, aunque algunas de ellas fueron descartadas al avanzar el proyecto ya que se encontraron alternativas más eficientes.

Reflexionando acerca de la arquitectura centralizada de la botnet, será necesario tener en cuenta una serie de aspectos:

1. Por su arquitectura, ya que se delega todo el control en un solo nodo, es importante saber que si ese nodo falla, probablemente se pierda toda la comunicación con los bots, por ello, es necesario adoptar medidas que intenten garantizar a toda costa la conexión con la red de bots.
    1. Una solución sería, programar el software alojado en los bots (clientes) de manera que se cargue y establezca la conexión en cuanto la máquina se encienda. (Habrá que tocar comandos de sistema y tener en cuenta el soporte multiplataforma).
    2. Otra solución es, programar tareas cron (en Linux), de manera que cada cierto tiempo, el cliente intente conectarse al servidor de control, así si en un momento falla la comunicación, no la perderemos para siempre, si no que la perderemos solo en el período hasta que vuelva a establecer la conexión según la programación elegida.

2. La dirección del servidor no puede cambiar, o de lo contrario se perderá la comunicación con los bots de manera irreversible, por lo que se asume que el servidor se encuentra siempre activo.
    1. Si se disponen de varios servidores, o puntos de acceso para los bots, sería posible definir un archivo con las direcciones ip de los servidores a los que pueden conectarse si uno de ellos falla, así tendremos más recursos para volver a reestablecer la comunicación.

5.6. Árbol de características

En la siguiente Figura se muestra el árbol de características del sistema. Las características principales vienen representadas en un cuadro amarillo, y en las ramas de cada una figuran las sub-características que las componen.

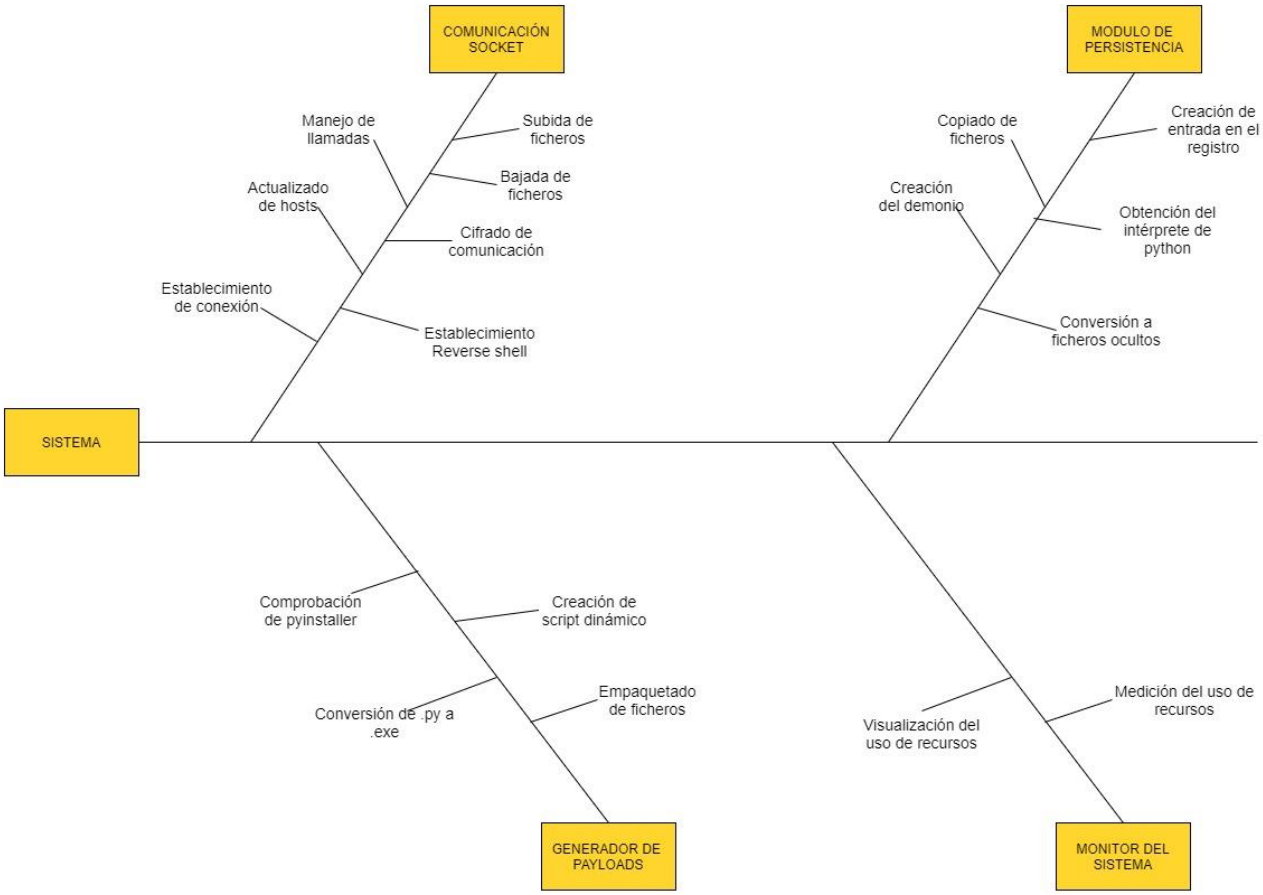


Figura 7: Árbol de características.

Esta tabla se compone de todas las características del sistema, a las que se le ha añadido un identificador del tipo C-XX y a las sub-características C-XX-XX, derivado de la característica que las contiene.

ID	Nombre
C-01	Comunicación Socket.
C-01-02	Manejo de llamadas.
C-01-03	Subida de ficheros.
C-01-04	Bajada de ficheros.
C-01-05	Actualizado de hosts.
C-01-06	Cifrado de comunicación.
C-01-07	Establecimiento de conexión.
C-01-08	Establecimiento de Reverse Shell.
C-02	Módulo de persistencia.
C-02-02	Copiado de Ficheros.
C-02-03	Creación del demonio.
C-02-04	Creación de entrada en el registro.
C-02-05	Obtención del intérprete de Python.
C-02-06	Conversión a ficheros ocultos.
C-03	Generador de Payloads.
C-03-02	Empaquetado de ficheros.
C-03-03	Creación de Script dinámico.
C-03-04	Conversión de .py a .exe.
C-03-05	Comprobación de pyinstaller.
C-04	Monitor del sistema.
C-04-02	Medición del uso de recursos.
C-04-03	Visualización del uso de recursos.

Tabla 13: Características del sistema.

5.7. Restricciones

En esta sección se reflejarán las posibles restricciones a las que tendrá que hacer frente el proyecto, ya que puede estar sujeto a limitaciones e influencie en ciertas decisiones de diseño o implementación. Principalmente son de carácter de tipo técnico como el tamaño permitido de los ficheros que se pueden subir o descargar, la tecnología de desarrollo empleada o la implementación de estándares de seguridad de la comunicación en red.

Cada restricción tiene un identificador asociado que sigue el formato R-XX:

ID	Restricción
R-01	El sistema debe de ser capaz de subir y descargar cualquier fichero siempre que no supere los 2GB
R-02	Se utilizará Python y sus recursos como lenguaje de programación para agilizar el desarrollo.
R-03	El sistema deberá esperar un máximo de 2 segundos antes de determinar una conexión como perdida.
R-04	El sistema deberá de utilizar el protocolo SSL en todos los mensajes que envía y reciba.

Tabla 14: Restricciones

5.8. Requisitos Funcionales

Los requisitos funcionales son elementos clave para el desarrollo de sistemas y aplicaciones, ya que definen las funcionalidades y comportamientos que se esperan de un sistema o software específico.

El objetivo principal de esta sección es identificar y describir de manera clara y concisa los requisitos funcionales necesarios para el diseño y desarrollo del proyecto.

Poseen un identificador RF-XX.

Id	Requisito
RF-01	El sistema debe de poder admitir conexiones entrantes de los bots.
RF-02	El sistema debe de integrar el protocolo ssl.
RF-03	El sistema debe de gestionar todas las conexiones con las máquinas de manera correcta y sin fallos.
RF-04	El sistema debe de permitir establecer una reverse Shell con el Bot que el usuario quiera.
RF-05	El sistema debe de poder lanzar campañas de órdenes a todas las máquinas a la vez.
RF-06	El sistema debe de ser robusto y gestionar los errores sin interrumpir la ejecución.
RF-07	El sistema debe de tener soporte para máquinas Linux y Windows.
RF-08	El sistema debe de mostrar al usuario todos los mensajes que reciba de las máquinas.
RF-10	El sistema debe de permitir la subida de ficheros a máquinas remotas.
RF-11	El sistema debe de permitir la descarga de ficheros de máquinas remotas.
RF-12	El sistema debe de poder generar sus propios payload para Windows y Linux.
RF-13	El sistema debe de contar con mecanismos de instalación de dependencias para su correcto funcionamiento.
RF-14	El sistema debe de comprobar antes de realizar cualquier operación si el software requerido se encuentra instalado en el dispositivo.
RF-15	El sistema debe de poder instalar persistencia en el dispositivo infectado y sobrevivir a reinicios.
RF-16	El script de comunicación del cliente debe de ejecutarse en la máquina objetivo de manera silenciosa y en segundo plano.
RF-17	El sistema generará Payloads en formato ejecutable sin necesidad de una librería instalada.

Tabla 15: Requisitos funcionales

5.9. Requisitos de Usuario

En esta sección recopilaremos los requisitos de usuario reflejando así las funcionalidades que deben de percibir lo usuarios que utilicen la aplicación.

Es importante destacar que nuestro sistema no posee ni interactúa con interfaces externas, debido que los requerimientos del proyecto no lo solicitan.

Cuentan con un identificador RU-XX

Id	Requisito
RU-01	El usuario debe de poder generar sus propios Payloads compatibles con Linux y Windows.
RU-02	El usuario tendrá la capacidad de lanzar ordenes de multidifusión entre todas las máquinas.
RU-03	El usuario podrá visualizar un listado de bots conectados a la botnet.
RU-04	El usuario podrá disponer de Shells reversas de la máquina que quiera siempre y cuando se encuentre conectada.
RU-05	El usuario podrá elegir una máquina y descargar un fichero remoto de ella.
RU-06	El usuario podrá elegir una máquina y subir un fichero local hacia ella y ejecutarlo.
RU-07	El usuario podrá monitorizar el uso de recursos del servidor cuando lo solicite.
RU-08	El usuario podrá cerrar todas las conexiones cuando lo requiera.
RU-09	El usuario podrá desplegar el servidor en un entorno real.
RU-10	El usuario podrá ejecutar cualquier comando (mientras las tecnologías lo permitan) en las máquinas de la botnet.

Tabla 16: Requisitos de usuario.

5.10. Casos de uso

A pesar de que hemos elegido la metodología scrum y la recopilación de los requisitos de usuario se ha llevado a cabo mediante historias, decidí realizar un modelado de casos de uso para entender mejor el problema y ver cómo los diferentes actores interactuaban con el sistema.

En la siguiente imagen contamos con dos actores, el Bot representa a la máquina que ha sido infectada y ya pertenece a la red. El atacante podría decirse que representa al usuario final al que va dirigido el software.

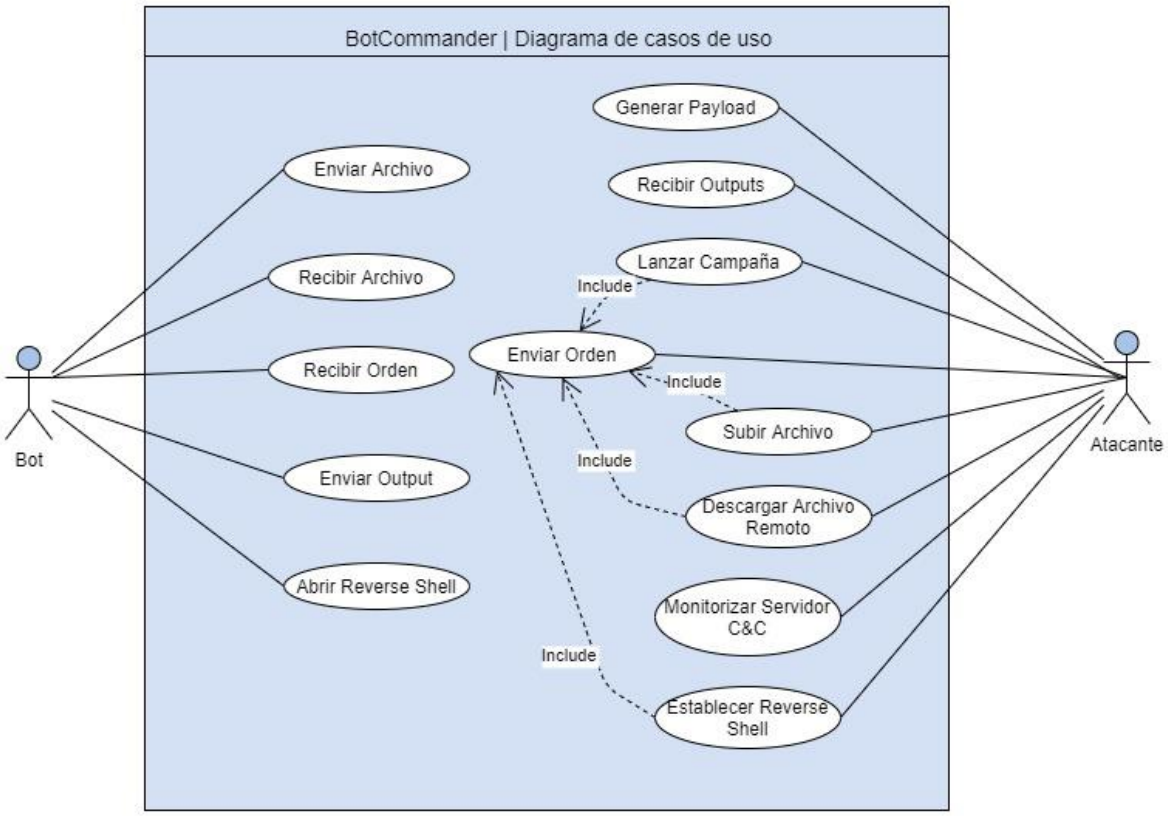


Figura 8: Modelado de casos de uso.

Capítulo 6.

Plan de Riesgos.

6. Introducción.

Un riesgo puede definirse como un evento o un problema potencial que puede o no ocurrir.

Podemos decir que un riesgo:

- Implica un cambio.
- Nos enfrenta a elecciones.
- Afecta a futuros acontecimientos.

Las principales etapas de un proyecto desde la proyección de la gestión y el análisis de riesgos son las siguientes:

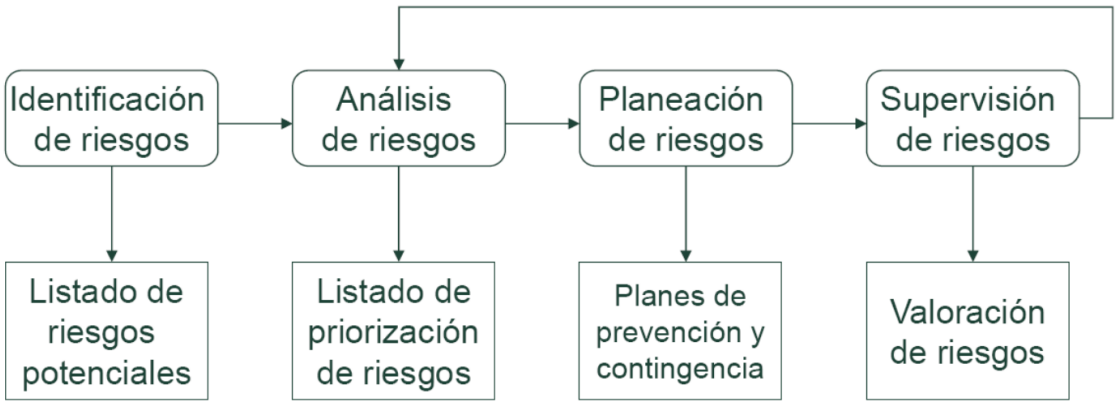


Figura 9: Proceso de gestión de riesgos.

6.1. Identificación de Riesgos

En este apartado, se abordará el proceso de identificación de riesgos específicos asociados a nuestro TFG. Mediante un análisis minucioso y una evaluación cuidadosa, buscamos reconocer y comprender los elementos inciertos y potencialmente perjudiciales que podrían impactar el cumplimiento de nuestros objetivos y la calidad de nuestro trabajo.

La identificación de riesgos nos permite estar preparados y tomar acciones preventivas, minimizando así los impactos negativos y maximizando nuestras posibilidades de éxito. Al reconocer de manera proactiva los posibles desafíos que podríamos enfrentar, estamos mejor equipados para gestionarlos de manera eficiente y efectiva,

A continuación se ha elaborado una lista con los riesgos identificados en el proyecto.

- RG-01: Escasa formación o experiencia de los miembros del equipo.
- RG-02: Posibles bugs en librerías utilizadas.
- RG-03: Incapacidad de cumplir con los tiempos de entrega.
- RG-04: Feedback insuficiente por parte del cliente.
- RG-05: Falta de control y seguimiento del desarrollo por parte del equipo.
- RG-06: Cambios en la especificación de requisitos.

6.2. Análisis de Riesgos

El análisis de riesgos es una fase esencial en la gestión de cualquier proyecto. En esta etapa, se lleva a cabo una evaluación exhaustiva y sistemática de los posibles riesgos identificados, con el objetivo de comprender su impacto potencial y desarrollar estrategias efectivas para mitigarlos.

También nos permite comprender la probabilidad de ocurrencia de cada riesgo, así como el impacto que podría tener en el alcance, los plazos y la calidad del software.

Listado de Priorización de riesgos:

- *RG-03: Incapacidad de cumplir con los tiempos de entrega.*

Este riesgo se coloca en la primera posición debido a su impacto directo en la planificación y el cronograma del proyecto. Si no se logran cumplir los plazos establecidos, esto puede tener consecuencias negativas en términos de retrasos en la entrega del software, pérdida de confianza del cliente y posibles repercusiones financieras. La puntualidad es fundamental en el desarrollo de software, por lo que este riesgo debe ser priorizado para garantizar el cumplimiento de los plazos y la satisfacción del cliente.

- *RG-06: Cambios en la especificación de requisitos.*

Los cambios en los requisitos pueden tener un impacto significativo en un proyecto de software, ya que pueden llevar a modificaciones en el diseño, desarrollo y pruebas del software. Esto puede resultar en retrasos, aumento de costos y dificultades para cumplir con los objetivos establecidos. La gestión efectiva de los cambios en los requisitos y la comunicación clara con el cliente son fundamentales para minimizar el impacto de este riesgo.

- *RG-01: Escasa formación o experiencia de los miembros del equipo.*

La falta de formación o experiencia adecuada en el equipo puede afectar la calidad y eficiencia del desarrollo de software. Puede llevar a errores, malas prácticas de programación y falta de conocimiento en herramientas o tecnologías específicas. Esto puede tener un impacto negativo en el rendimiento del equipo y en la calidad final del producto entregado. Es importante asegurarse de que el equipo cuente con las habilidades y conocimientos necesarios para realizar el trabajo de manera efectiva.

- *RG-02: Posibles bugs en librerías utilizadas.*

El uso de librerías o componentes de terceros es común en el desarrollo de software. Sin embargo, existe el riesgo de que estas librerías contengan errores o bugs que puedan afectar el funcionamiento del software en general. Esto puede resultar en fallos, mal rendimiento o vulnerabilidades de seguridad. Es crucial realizar una evaluación rigurosa de las librerías utilizadas, mantenerlas actualizadas y tener un plan de contingencia en caso de que se descubran problemas en ellas.

- *RG-04: Feedback insuficiente por parte del cliente.*

La falta de feedback o retroalimentación por parte del cliente puede dificultar la comprensión de los requisitos y expectativas del software. Esto puede llevar a malentendidos, entregas incorrectas o insatisfactorias y, en última instancia, a la insatisfacción del cliente. Es fundamental establecer canales de comunicación claros con el cliente y fomentar una retroalimentación constante para garantizar que se cumplan sus necesidades y expectativas.

- *RG-05: Falta de control y seguimiento del desarrollo por parte del equipo.*

La falta de control y seguimiento del desarrollo puede llevar a problemas de calidad, falta de alineación con los objetivos del proyecto y dificultades para detectar y corregir problemas a tiempo. Es importante establecer mecanismos adecuados de control y seguimiento, como reuniones periódicas, revisiones de código y pruebas de calidad, para asegurar un desarrollo adecuado y minimizar el riesgo.

6.3. Planeación de Riesgos

En esta fase se estudiarán las estrategias de mitigación de cada uno de los riesgos mencionados anteriormente.

Riesgo	**RG-01**
Descripción	Escasa formación o experiencia de los miembros del equipo.
Impacto	Alto
Probabilidad	Media
Mitigación	Diseñar e implementar una fase de formación inicial para el colectivo inexperto.
Plan de contingencia	Contratar algún perfil senior dentro del equipo para que forme a los demás.

Tabla 17: Riesgo RG-01

Riesgo	**RG-02**
Descripción	Posibles bugs en librerías utilizadas.
Impacto	Medio
Probabilidad	Alta
Mitigación	Documentarse acerca de las librerías que se vayan a utilizar.
Plan de contingencia	Realizar las correcciones pertinentes o utilizar código propio en la medida de lo posible.

Tabla 18: Riesgo RG-02

Riesgo	**RG-03**
Descripción	Incapacidad de cumplir con los tiempos de entrega.
Impacto	Alto
Probabilidad	Media
Mitigación	Centrarse en las funcionalidades principales que aporten mayor valor al cliente.
Plan de contingencia	Realizar una buena estimación de tiempos al inicio y durante las primeras fases de los sprint.

Tabla 19: Riesgo RG-03.

Riesgo	**RG-04**
Descripción	Feedback insuficiente por parte del cliente.
Impacto	Alto
Probabilidad	Media
Mitigación	Realizar frecuentes reuniones con el cliente.
Plan de contingencia	Entregar productos funcionales al final de cada incremento con el fin de impulsar la motivación del cliente.

Tabla 20: Riesgo RG-04

Riesgo	**RG-05**
Descripción	Falta de control y seguimiento del desarrollo por parte del equipo.
Impacto	Alto
Probabilidad	Media
Mitigación	Implementar fases específicas de seguimiento y control.
Plan de contingencia	Reuniones de Retrospectiva.

Tabla 21: Riesgo RG-05

Riesgo	**RG-06**
Descripción	Cambios en la especificación de requisitos.
Impacto	Alto
Probabilidad	Baja
Mitigación	Planear y analizar mejor las tareas en etapas tempranas.
Plan de contingencia	Realizar un correcto análisis del impacto de los cambios.

Tabla 22: Riesgo RG-06

6.4. Supervisión de Riesgos.

Las tareas de supervisión y valoración de riesgos han estado presentes en todas las fases ya que las necesidades del proyecto iban cambiando y era necesario supervisar y evaluar de nuevo los riesgos.

Capítulo 7.

Detalles de Implementación.

Este capítulo se centrará en aportar los detalles técnicos y de implementación que se han llevado a cabo en todo el ciclo de desarrollo del software BotCommander.

Se pretende proporcionar al lector una visión más detallada enfocada al perfil técnico.

7.1. Comunicación

Como se ha explicado con anterioridad, la comunicación entre las máquinas está basada en la implementación de sockets TCP.

Se han creado dos scripts que contienen la lógica de los dos roles que intervienen en el proceso, el cliente y el servidor (serán explicados en las secciones siguientes).

7.1.1. Cliente

La lógica de los bots viene recogida en el fichero multi_client_client.py.

multi_client_client.py.

Será el script que se ejecute en segundo plano y de manera silenciosa en las máquinas que hayan sido infectadas previamente, de manera que el usuario no perciba ninguna actividad sospechosa.

Inicialmente, cuenta con una serie de datos necesarios para llevar a cabo todas las operaciones:

- Dirección del servidor de comando y control: Dirección ip y puerto del servidor central.
- Tamaño del buffer: Tamaño en bytes del buffer de recepción y de envío que será usado en las operaciones de envío y recepción de datos.
- Sistema operativo: Es una variable que almacena el tipo de plataforma de la máquina para enviarla al servidor posteriormente.
- Constante separadora: Es una cadena de caracteres elegida por conveniencia para formatear los mensajes que llegan del servidor, ya que en un mismo mensaje se envía el directorio actual en el que se encuentra y output del comando.
- Tiempo de reintento de conexión: Es una constante temporal que establece el tiempo que tardará en realizar otro intento de conexión contra el servidor.

Lo primero que realiza es una espera pasiva de 1 minuto, recordemos que la ejecución tendrá lugar cuando el sistema se inicie, por lo que es posible que los recursos de red aún no estén disponibles, si esto sucediese, no se podría establecer una conexión, por lo que es recomendable realizar una espera prudencial.

El siguiente paso es crear un objeto socket que se utilizará para establecer conexión, enviar datos y recibirlos.

Si no es capaz de conectarse, saltará una excepción y esperará un tiempo de 15 minutos (900 segundos) para volver a intentarlo.

Cuando ha obtenido un intento exitoso y tiene comunicación, el bot se pone en modo de escucha activa de órdenes. Dichas órdenes se redirigen a una función manejadora de llamadas, que clasificará y enrutará cada una en el flujo correspondiente, por ejemplo si la orden es ”open reverse shell” la manejadora invocará a la función encargada de proporcionar esta funcionalidad. Esto hace que sea mucho más escalable, permitiendo añadir un sinfín de órdenes y funcionalidades a la aplicación, sencillamente creando unas cuantas líneas de código y definiendo una nueva llamada.

Las funcionalidades para las que está programado el bot son las siguientes:

Execute

Permite recibir órdenes del servidor central, ejecutarlas y enviar el output de vuelta.

El código fuente se encuentra en la función execute().

1. Recibe la orden “execute” del servidor.
2. Envía el flag “<<Ready>>” al servidor para indicarle que está preparado para recibir el comando.
3. Recibe posteriormente el comando que el atacante ha confeccionado y enviado.
4. Mediante el módulo subprocess, interactúa con el sistema operativo del bot, permitiendo ejecutar la orden del atacante utilizando suprocess.getoutput(“comando”). Esto devolverá el resultado de la ejecución y el bot lo almacenará en una variable.
5. Se codifica el output y se envía haciendo uso de la función send() del objeto socket.

System-Data

System-data es un tipo de llamada que realiza el servidor hacia los bots para que éstos le indiquen el tipo de sistema que están utilizando.

El código fuente se encuentra en la función send_data_system().

1. Recibe la orden “system-data”.
2. Determina el tipo de sistema que es, y efectuará las operaciones pertinentes para averiguar el sistema operativo, la arquitectura y la versión del kernel.
3. Envía el resultado al servidor de atacante.

Reverse

Implementa el soporte para brindar al servidor una reverse Shell [11] remota del bot.

El código fuente se encuentra en la función open_reverse_TCP()

1. Recibe la orden “Reverse”.
2. Ejecuta el comando os.getcwd() para obtener el directorio de trabajo y enviarlo a través del socket.
3. En este punto, el bot se pone en escucha activa y el servidor ya tiene la reverse Shell establecida.
4. Recibe una orden, la ejecuta y guarda el output. Si sucede algún error o el comando no es compatible, se notifica al atacante.
5. Forma el mensaje a enviar, que estará compuesto de tres partes:
    a. La primera parte del mensaje es el output del comando.
    b. La segunda parte corresponde a un separador.
    c. La tercera parte es el directorio de trabajo. Es necesario para emular la consola de una Shell interactiva.

Download-file

También es posible descargar un archivo del bot hacia el servidor para almacenarlo de manera local.

El código fuente se encuentra en la función send_file().

1. Recibe la orden “download-file”.
2. Le indica al servidor que está preparado mediante el flag “<<Ready>>”.
3. Recupera la ruta del archivo que el servidor ha indicado que quiere descargar y la comprueba.
4. Si no existe, notifica al servidor y aborta. En caso afirmativo abre el fichero en modo lectura binaria, y mientras haya datos, envía paquetes de 1024 bytes hacia el servidor.

Upload-file

Sirve para subir un archivo a un bot desde el servidor central. Para que esto sea posible, el bot debe de saber cómo gestionar esta funcionalidad, la lógica sigue los siguientes pasos:

El código fuente se encuentra en la función receive_file().

1. Recibe la orden “upload-file”.
2. Le indica al servidor que está preparado mediante el envío del flag “<<Ready>>”.
3. Recupera la ruta del archivo donde se almacenará el fichero que se va a subir. Comprobará que la ruta es válida en el contexto de la máquina bot.
4. Abrirá un fichero en modo escritura binario, e ira recibiendo y escribiendo los paquetes de 1024 bytes.

Close-connection

Esta es una funcionalidad que permite al servidor cerrar la conexión con cualquiera de los bots de una manera segura.

El proceso detallado es:

1. Los clientes reciben la orden “close-conection”.
2. Se cerrará la conexión y la ejecución del script terminará.

7.1.2. Servidor

La lógica del servidor viene recogida en el fichero multi_client_server.py, dentro del directorio server del proyecto.

multi_client_server.py.

Será el script Python que desplegará el servidor [12] de comando y control para gobernar a los bots.

Inicialmente, cuenta con una serie de datos necesarios para llevar a cabo todas las operaciones:

- Dirección de un adaptador de red válido dentro del contexto de la máquina virtual. Es importante destacar que tiene que ser la dirección ip privada visible desde dentro de la máquina.
- El puerto de la máquina donde se desplegará el servicio. Hay que tenerlo en cuenta para redirigir el tráfico desde el exterior hacia este.
- Tamaño del buffer: Tamaño en bytes del buffer de recepción y de envío que será usado en las operaciones de envío y recepción de datos.
- Constante separadora: Es una cadena de caracteres elegida por conveniencia para formatear los mensajes que llegan del servidor, ya que en un mismo mensaje se envía el directorio actual en el que se encuentra y output del comando.
- Lista de hosts: contempla una lista con los objetos socket de todos los bots que están conectados.
- Lista de direcciones de hosts. Su uso es meramente por una cuestión de interfaz para mostrar al usuario las direcciones sin tener que parsear la información de los objetos.

Cuando se ejecuta el script, un objeto socket servidor será creado. Utilizará la función bind(ip, port) para desplegarlo.

Se pondrá en modo escucha y un nuevo hilo de ejecución será lanzado para gestionar la solicitud de conexiones entrantes de los bots, de esta manera el flujo de la aplicación no será bloqueado y se pueden realizar tareas concurrentes.

BotCommander está compuesto por un menú principal que facilita su uso permitiendo al usuario elegir de una manera sencilla qué operación quiere realizar con la botnet.

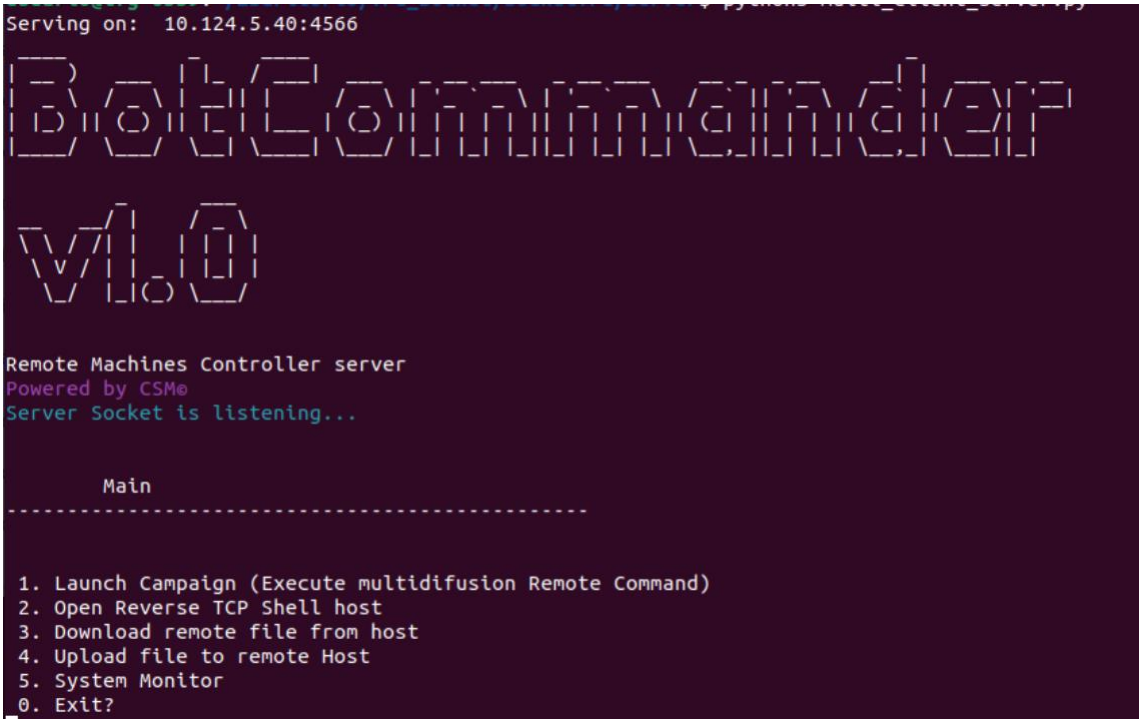


Figura 10: Imagen del servidor de comando y control.

Cuando se selecciona cualquiera de las opciones del menú (a excepción de la 5), el sistema hace uso de una función llamada verify_connections() que se encarga de testear la conexión de todas las máquinas que están conectadas, si alguna de ellas no responde en un periodo de 2 segundos, se asume que la conexión con el host se ha perdido y se actualiza la lista de hosts.

También se contempla el posible suceso de otro tipo de fallas en los sockets que serán recogidos en las excepciones oportunas, resultando así en la actualización de la lista de hosts conectados, desechando las conexiones perdidas.

1.- Launch Campaign

Esta opción permite lanzar un comando a todos los bots conectados y recibir su output.

El usuario realiza un input para indicar el comando que quiere ejecutar en la botnet. Seguidamente, se lanzará un hilo por cada bot conectado para que la orden sea enviada a cada uno siguiendo un esquema de paralelismo real, con la finalidad de aumentar el rendimiento del servidor, ya que estará sometido a la gestión múltiple de conexiones, algo que podría inducir en una sobrecarga si la programación no es adecuada o el número de hosts resulta inmanejable por una incapacidad de recursos.

La respuesta de cada bot es recibida por el hilo y presentada al usuario por consola, indicando el estatus de la ejecución de la orden.

2.- Open Reverse TCP Shell host.

Como se mencionó en la descripción del cliente, ambos roles están preparados para brindar al atacante una Shell remota de administración.

Al presionar esta opción, el sistema muestra una lista actualizada de los hosts que forman parte de la botnet, permitiendo al usuario elegir uno de ellos para establecer una Shell reversa. En la lista, se proporciona información del sistema del bot para que el usuario pueda tenerla en cuenta a la hora de interactuar con la Shell, ya que BotCommander es multiplataforma y los comandos entre distintos sistemas operativos difieren.

Cuando la máquina ha sido seleccionada, internamente se realiza el envío de la orden "reverse" para que nos brinde el servicio.

En este punto ya estaría disponible la Shell remota para el usuario y bastaría con introducir el comando "exit" para salir al menú principal.

La función que contiene toda la lógica es establish_reverse_TCP(client_socket) que recibe un objeto socket cliente. En ella el funcionamiento está basado en el intercambio de mensajes, pero hay una implementación que hay que destacar:

Cuando un mensaje a recibir de la máquina supera el tamaño del buffer de recepción, es necesario realizar más recuperaciones de información hasta que el mensaje haya terminado, es decir, no haya más datos en el socket. Para determinar cuándo es el final del mensaje, se ha establecido un timeout en el socket cliente, que esperará unos segundos la respuesta y si no llega, asumimos que el mensaje ha terminado. De otro modo, si siguen llegando datos, se concatenan en una variable resultado hasta que el timeout salta y se muestra al usuario.

Bajo mi punto de vista, es una de las mejores soluciones al problema de los bufferes estáticos que sufren este tipo de sockets, no obstante posee ciertos inconvenientes al delegar esta decisión al consumo de un timeout, por ejemplo, si sucede un pico en la red y el paquete tarda más de lo normal en llegar, lo que supondría una pérdida de información.

3.- Download remote file from host.

Al elegir esta opción, se muestran por pantalla al usuario los hosts conectados y se le pide que elija uno para descargar el fichero remoto.

Una vez que lo ha elegido, se envía la orden "download-file" al bot y este responderá con el flag "<<Ready>>".

Se solicita la ruta del fichero que quiere descargar en la máquina, y el directorio local donde será guardado. El sistema enviará la ruta y el bot responderá si es válida o no.

El servidor abrirá un fichero en modo escritura binaria e irá escribiendo en él, bloques de 1024 bytes a medida que los va recibiendo del cliente.

Se han proporcionado mecanismos para gestionar problemas con la sintaxis de rutas ya que difieren entre sistemas operativos.

4.- Upload file to remote host.

Al elegir esta opción, se muestran por pantalla al usuario los hosts conectados y se le pide que elija uno para subir el fichero.

Una vez que lo ha elegido, se envía la orden “upload-file” al bot y este responderá con el flag “<<Ready>>”.

Se solicita la ruta del fichero que quiere subirá a la máquina, y el directorio remoto donde será guardado. El sistema enviará la ruta y el bot responderá si es válida o no.

El servidor abrirá un fichero en modo lectura binaria e irá leyendo y enviando al cliente bloques de 1024 bytes.

Se han proporcionado mecanismos para gestionar problemas con la sintaxis de rutas ya que difieren entre sistemas operativos.

5.- System monitor.

Debido a la naturaleza centralizada de la arquitectura, toda la responsabilidad recae en el servidor central, por lo que se ha implementado una funcionalidad que permite monitorizar el uso de recursos del sistema en todo momento.

Como se va a gestionar un número elevado de conexiones, es importante conocer el estado de los recursos hardware, ya que si el servidor se cuelga, todas las máquinas se perderán, al menos por un periodo de tiempo, hasta que vuelvan a reconectarse.

Esta funcionalidad muestra el uso de CPU y de memoria en tiempo real con una tasa de refresco de 2 segundos.

Con la finalidad de no interferir en las conexiones abiertas y el resto de las tareas, se ha implementado un hilo que gestiona estas mediciones de manera independiente.

Cuando se selecciona esta opción, el sistema lanza un nuevo hilo de ejecución que tomará las muestras.

La tarea de parar un hilo puede ser un tanto ardua si no se sabe gestionar bien, y podemos sobrecargar el programa con hilos zombie, por ello se ha creado una variable que permite controlar la ejecución del hilo de la siguiente forma:

Cuando el usuario sale al menú principal después de ejecutar el monitor del sistema, se captura un evento, que es pasado como argumento a la función get_sys_info(event) para que detenga la captura de información de los recursos. Esto se consigue empleando el método .join() sobre un objeto de la clase thread, que comunicará el proceso actual con el hilo de ejecución. (Puede ver el código fuente).

7.1.3. Seguridad de las Comunicaciones.

Introducción

Hasta este nivel de desarrollo de la comunicación basada en sockets, nos apoyamos ampliamente en el protocolo TCP "plano", es decir, la comunicación viaja en texto claro y sin cifrar, algo altamente inseguro y vulnerable frente a escuchas indeseadas si alguien consigue interceptar en alguna de las partes (cliente o servidor) los paquetes que se envían por la red, quedando todas las órdenes e información expuestas de una máquina a otra.

Como solución a este problema, ha sido necesaria la implementación del protocolo SSL-TLS para securizar la comunicación entre el servidor y las máquinas cliente, por lo que se ha hecho uso del módulo ssl de Python que provee una API a alto nivel de implementación para sockets.

Módulo SSL de Python

Este módulo brinda acceso a la seguridad de la capa de transporte (a menudo conocida como «capa de sockets seguros») y las funciones de autenticación de pares para los sockets de red, tanto del lado del cliente como del lado del servidor. Este módulo utiliza la biblioteca OpenSSL. Está disponible en todos los sistemas Unix modernos, Windows, macOS y probablemente plataformas adicionales, siempre que OpenSSL esté instalado en esa plataforma.

Permite tunelizar una conexión socket TCP a una conexión TLS cifrada, abstrayendo de las operaciones que realiza el protocolo, como el handshake, intercambio de claves… etc.

SSL en el Servidor

En el desarrollo del servidor, se ha hecho uso del módulo ssl, y a su vez hemos creado un módulo que permite recibir una conexión socket y envolverla mediante el protocolo.

El proceso que hemos seguido para conseguir un servidor que implemente seguridad ssl [9] será el siguiente:

1. Generar un certificado y una clave privada que usará el servidor, para ello:
    1. Generamos la clave privada:

openssl genrsa -des3 -out server.key 2048

2. Generamos una solicitud de firma de certificado (CSR), donde especificaremos todos los atributos de identificación:

   openssl req -new -key server.key -out server.csr

3. Eliminamos la passphrase de la clave para simplificar nuestro programa:

   cp server.key server.key.org

   openssl rsa -in server.key.org -out server.key

   rm server.key.org

4. Generamos el certificado autofirmado:

   openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt

Nota: Es necesario tener en cuenta, que para que un certificado sea válido y el protocolo SSL no lo descarte, tiene que estar expedido por una entidad certificadora de confianza, la secuencia de pasos anterior indica como generar un certificado autofirmado (no es válido es requerido que el cliente autentique al servidor, más adelante desactivaremos la autenticación del servidor mediante certificado para poder hacer pruebas).

Cabe destacar que en un entorno en producción será necesario obtener un certificado legítimo (expedido por una entidad certificadora), o de lo contrario, un cliente no podría autenticar al servidor y podría ser víctima de una suplantación de servidor.

Lo siguiente será detallar cómo implementar el módulo para obtener una configuración segura:

Primero creamos un Socket de manera normal del siguiente modo:

```
main_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
main_socket.bind(("192.168.0.86", 443))
main_socket.listen(5)
```

de esta manera ya tendremos el servidor escuchando por un puerto, en este caso el 443.

El siguiente paso será aceptar conexiones de clientes y tunelizar esas conexiones mediante TLS:

```
client_socket,client_addres = main_socket.accept()
conection = ssl.wrap_socket(client_socket, server_side=True,
certfile="./server.crt", keyfile="./server.key",
ssl_version=ssl.PROTOCOL_TLS_SERVER)
```

La función ssl.wrap_socket() será la encargada de envolver o de crear el túnel SSL con la conexión del cliente, de manera que convertirá el socket stream (sólo son válidos los sockets stream) en un ssl socket.

Los parámetros que acepta son:

- Client_socket: es el objeto que contiene todos los datos de la conexión con el cliente devuelto por la función accept() del socket servidor.
- Server_side: parámetro booleano que indica si el socket se comportará como un socket servidor.
- Certfile: carga el fichero que contiene el certificado del servidor.
- Keyfile: carga el fichero que contiene la clave pública.
- Ssl_version: permite especificar la versión del protocolo ssl que queremos utilizar.

  Existen varias versiones del protocolo SSL/TLS, incluyendo SSL 1.0, SSL 2.0, SSL 3.0, TLS 1.0, TLS 1.1, TLS 1.2 y TLS 1.3. Sin embargo, SSL 1.0 y SSL 2.0 no se utilizan en la actualidad debido a que son inseguros y tienen vulnerabilidades graves.

  SSL 3.0 también es considerado inseguro y se recomienda no utilizarlo debido a que tiene vulnerabilidades conocidas como POODLE y BEAST.

  TLS 1.0 y TLS 1.1 todavía se utilizan en algunos sitios web, pero también tienen vulnerabilidades conocidas y se consideran inseguros. Es recomendable usar TLS 1.2 o superior para garantizar una seguridad adecuada.

  TLS 1.2 es actualmente la versión más utilizada y se considera segura. TLS 1.3 es la versión más reciente y es aún más segura que TLS 1.2, ya que tiene mejoras significativas en términos de seguridad y privacidad. Se recomienda utilizar TLS 1.3 siempre que sea posible.

  Si especificamos ssl_version=ssl.PROTOCOL_TLS_SERVER), usaremos la versión de ssl más reciente que haya disponible en la librería OpenSSL instalada en la máquina servidora, en el caso de uso del proyecto, utilizaremos la versión TLSv1.3 que es la más segura.

  - Ciphers: También podemos especificar los algoritmos de cifrado que utilizará la comunicación a la hora de hacer el handshake.

SSL en el cliente:

Definimos la dirección del servidor remoto:

```
server_adress = '192.168.0.86'
```

Creamos el contexto, ayuda a administrar la configuración ssl y los certificados, que luego pueden ser heredados por sockets SSL creados a través del método SSLContext.wrap_socket()

```
context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
```

Desabilitamos esta opción ya que estamos usando un ceertificado

self-signed y TLS lo descarta al no estar expedido por una CA oficial, en un entorno de producción habría que conseguir un certificado legítimo y habilitar las opciones check_hostname y verify_mode para que el cliente pueda autenticar al servidor.

```
context.check_hostname = False

context.verify_mode = ssl.CERT_NONE
```

Creamos el socket de una manera estándar

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Creamos una conexión tunelizada hacia el servidor mediante el contexto creado anteriormente y la función wrap_socket que envolverá el socket creado, finalmente conectamos y a partir de aquí toda la información viaja cifrada.

```
conn = context.wrap_socket(sock,server_hostname = server_adress)

conn.connect((server_adress, 443))
```

7.2. Módulos de persistencia

Cuando hablamos de malware, el término persistencia [21] se refiere a la capacidad de este para sobrevivir a reinicios del sistema.

Una característica de las botnets es esa, conseguir infectar la máquina de manera que siempre pertenezca a la botnet, aunque el dispositivo se apague, por ello se ha implementado un módulo de persistencia en Python para sistemas Windows y otro para Linux, a continuación se explicarán ambos:

7.2.1. Persistencia en Windows

El editor del registro de Windows es uno de los objetivos de los ciberatacantes a la hora de crear persistencia en una máquina objetivo, ya que gestiona, entre otras tareas, las aplicaciones que van a ejecutarse al iniciar sesión en la cuenta de un usuario.

Se ha diseñado un módulo (windows_persistence_unit.py), que permite alojar en la máquina victima el script de comunicación, convertirlo a fichero oculto y crear una entrada en el registro en la siguiente ruta:

Equipo\HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run

Esta entrada se compone de dos partes:

1. La dirección del intérprete de pythonw.
2. La ruta del script de comunicación del bot. (multi_client_client.py).

Pythonw

Pythonw es una versión del intérprete de Python en sistemas Windows, que utilizan algunas aplicaciones gráficas para ejecutar tareas en segundo plano. Hemos utilizado esta variante ya que facilita la implementación de un proceso Daemon para ejecutar nuestro script de comunicación del bot.

Por otro lado, es capaz de burlar a los sistemas de detección de Windows, como el defender y permite su ejecución ya que lo considera legítimo.

Una vez que está creada la entrada, cuando el usuario inicie sesión en el dispositivo, se lanzará un proceso silencioso que empezará a comunicarse con el servidor de comando y control.

7.2.2. Persistencia en Linux

En el caso de Linux, la idea es parecida pero difiere en su implementación.

Primero se copia el fichero de comunicación (en modo oculto) en el directorio /etc.

El segundo paso es obtener la ruta de instalación del intérprete de Python en la máquina, mediante la ejecución de un subproceso con la orden “which python”.

Se confecciona un fichero de servicio que servirá para crear el proceso demonio que ejecutará nuestro script. En él , se indicará la orden de llamada a Python para que ejecute el fichero de comunicación, y posteriormente, será alojado en el directorio /etc/systemd/system.

Finalmente se activa el servicio y se arranca. Llegados a este punto la persistencia ya estaría creada y se podría disponer del equipo en la botnet.

7.3. Generador de Payloads

Se ha desarrollado una herramienta que complementa a BotCommander, en este caso, la función será contar con un software que permita generar cargas útiles (payloads) para cada tipo de sistema operativo, generando así paquetes comprimidos de fácil uso con todos los componentes necesarios para infectar un dispositivo.

También es posible indicar la dirección ip y el puerto del servidor de atacante y será actualizado en el script de comunicación del bot.

Detalles de la implementación.

Un payload está compuesto por tres ficheros:

1. El script de comunicación: multi_client_client.py
2. La unidad de persistencia.
3. El fichero ejecutable (instalador de Python ficticio).

El Fichero ejecutable será la vía de entrada para el malware, ya que aparentemente es un script que comprueba la versión del sistema y te instala el paquete de Python necesario, después realizará una llamada a la unidad de persistencia, ésta efectuará los cambios necesarios en segundo plano y alojará el malware en el equipo.

Lo siguiente será indicar si quieres generar un payload para Linux o para Windows, ya que el ejecutable principal (El instalador de Python "ficticio") será un fichero .sh o un .exe según lo que indiquemos.

Después nos pedirá realizar un input con la dirección del servidor central. El programa está preparado para aceptar una dirección ip, o un dominio y posteriormente lo resolverá.

En este punto, el programa trabajará en la creación del payload, que seguirá el siguiente esquema:

> Si se trata de un payload para Windows, es necesario contar con la herramienta pyinstaller en el sistema, comprobará si lo está y en caso contrario la instalará.
>
> Generará el script de comunicación con la dirección del servidor C&C actualizada según lo que el usuario introdujo.
>
> Moverá los archivos de persistencia y el instalador a un directorio.
>
> El resultado final será un paquete comprimido, con todo lo anterior.

En el caso de Linux, creará un fichero .tar.gz con los 3 ficheros necesarios.

1. Windows_python_instaler.sh
2. Windows_persistence_unit.py
3. .multi_client_client.py

En el caso de Windows, creará un fichero .zip con los 3 ficheros necesarios.

1. Windows_python_instaler.exe
2. Windows_persistence_unit.py

3. .multi_client_client.py

7.4. Descripción del payload.

En esta sección se detallará el proceso interno que sigue el payload cuando es ejecutado en una máquina.

En Windows

En el caso de Windows, el generador de Payloads cuenta con el fichero Windows_python_installer.py, que será compilado para dar lugar a un fichero .exe mediante la herramienta pyinstaller. Es el encargado de obtener los recursos de instalación de python de la red, y una vez que ha instalado correctamente los recursos en la máquina, realiza una llamada a la unidad de persistencia, en este caso tendrá que tener en cuenta el tipo de sistema que es y llamar a la unidad correspondiente (Windows_persistence_unit.py).

Este fichero se encargará de crear toda la persistencia, moviendo el fichero de comunicación a una ruta local de la máquina y posteriormente creará una entrada en el registro de Windows con la clave del usuario logueado. En ella escribirá la orden de llamada al interprete pythonw y le pasará como argumento la ubicación del script de comunicación (multi_client_client.py).

En Linux

En Linux el proceso es un poco más sencillo, todo empieza por ejecutar Linux_python_installer.sh para instalar python y realizar una llamada a la unidad de persistencia. En esta unidad, se creará un fichero en la ruta /etc/systemd/system/ que dará de alta un servicio demonio. Este fichero contendrá la orden de llamada a python introduciéndole como argumento el script de comunicación multi_client_client.py.

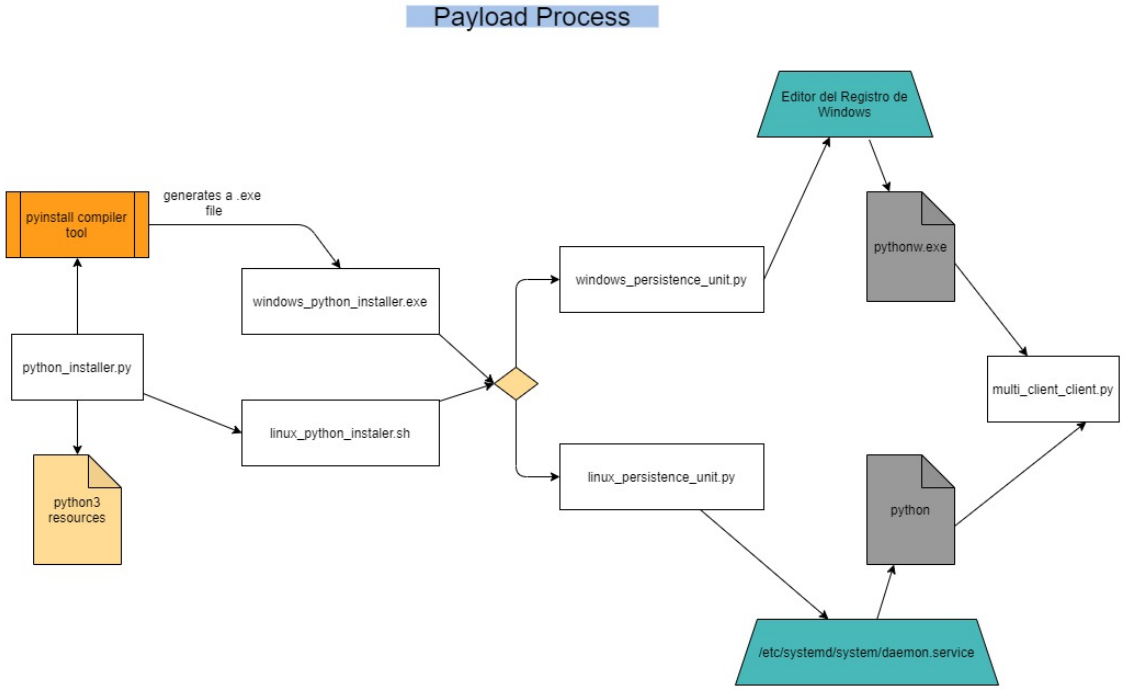


Figura 11: Proceso de infección del dispositivo mediante el payload.

Capítulo 8.

Manual de usuario.

8.1. Manual de Usuario

El proyecto cuenta con la estructura de directorios siguiente:

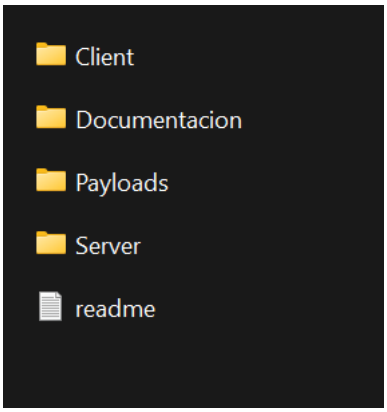


Figura 12:Estructura de directorios del proyecto.

- Client: Aquí están todos los ficheros del cliente.
- Documentación: Documentación del proyecto, diagramas e información.
- Payloads: Se encuentra el generador de Payloads.
- Server: Todo lo necesario para desplegar el servidor.
- Readme: Pequeñas instrucciones de instalación.

8.2. Despliegue del servidor

En el directorio Server:

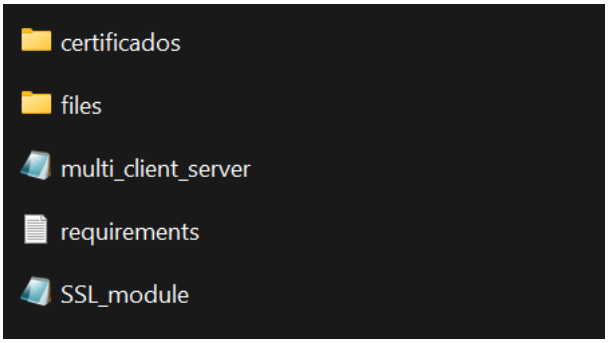


Figura 13:Estructura de archivos del servidor.

1. Ejecutar $pip install -r requirements.txt para instalar las librerías necesarias (hay que tener pip instalado).
2. Comprobar si openssl se encuentra instalado en el sistema (instalarlo si no lo está).
3. Generar el certificado y la clave privada para el servidor, para ello sigue los siguientes pasos:
4. Generar un certificado y una clave privada que usará el servidor, **es importante guardarlo todo en el directorio certificados**, para ello:

    a. Generamos la clave privada:

    openssl genrsa -des3 -out server.key 2048

    b. Generamos una solicitud de firma de certificado (CSR), donde especificaremos todos los atributos de identificación:

    openssl req -new -key server.key -out server.csr

    c. Eliminamos la passphrase de la clave para simplificar nuestro programa:

    cp server.key server.key.org

    openssl rsa -in server.key.org -out server.key

    rm server.key.org

    d. Generamos el certificado autofirmado:

    openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt

Una vez realizado este proceso, abrir el fichero multi_client_server.py con un editor de texto y modificar las líneas *host = 'your ip server', port = <<your port>>*, guardar y salir.

```
if __name__ == "__main__":

    #------------GLOBALS-----------------------------------------------
    ServerSideSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    host = '10.124.5.40'
    #host = '192.168.1.11'
    port = 4566
```

Figura 14: Fichero del cliente (Bot).

Ejecutar el script multi_client.server.py en una terminal mediante la orden $python3 mult_client_server.py y se habrá desplegado el servidor:

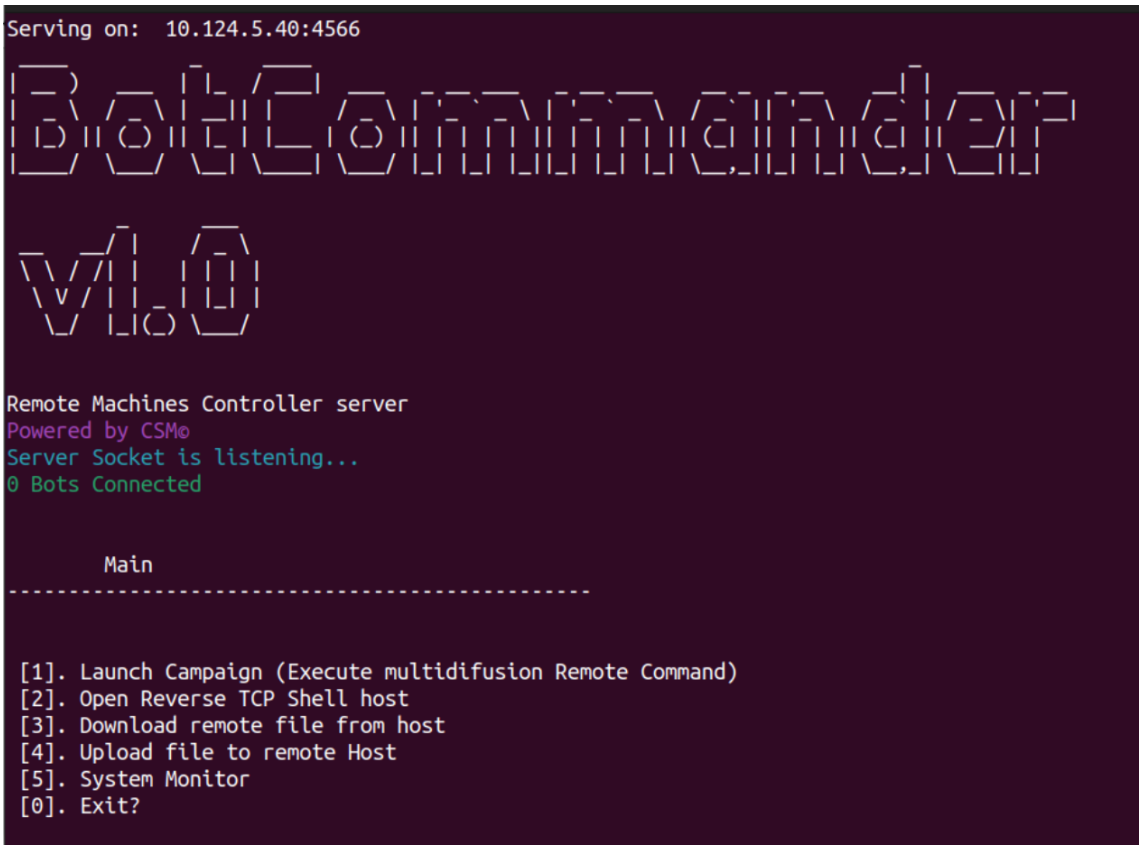


Figura 15: Imagen del servidor desplegado.

8.3. Explicación de uso del servidor.

1. Launch Campaign.

   Para lanzar una campaña con las máquinas conectadas, seleccionar la opción 1, e indicar el comando deseado, pulsar enter para proceder:

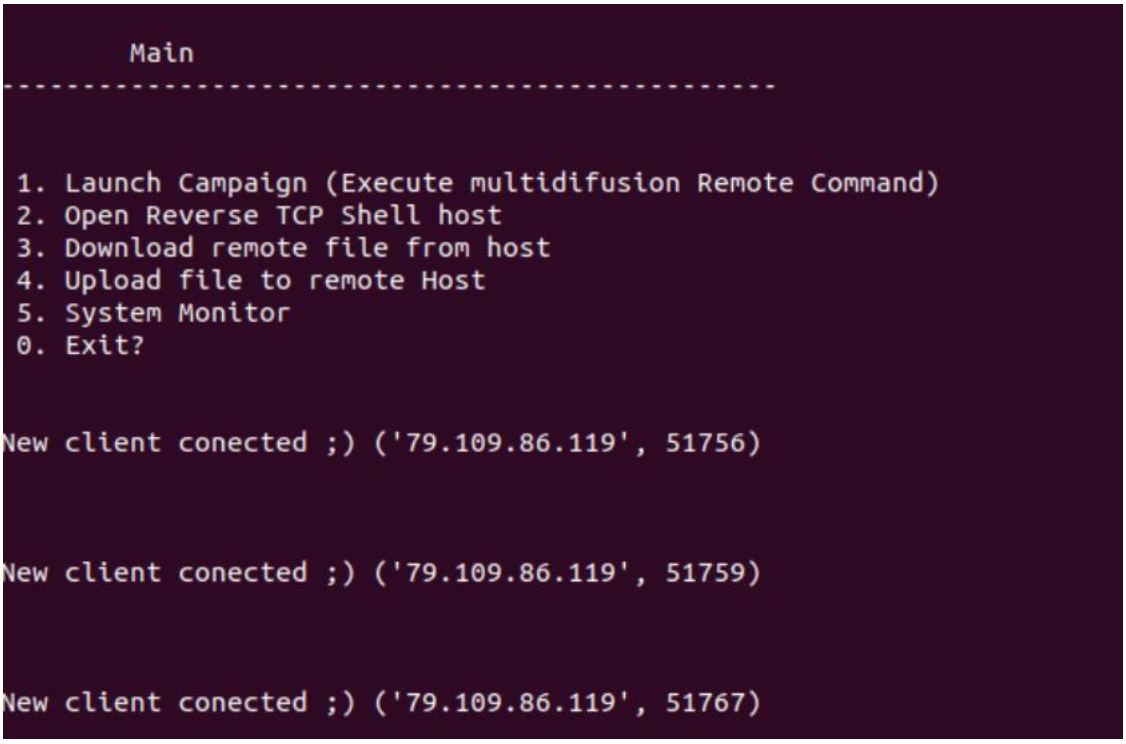


Figura 16: Imagen del servidor cuando se conectan bots nuevos.

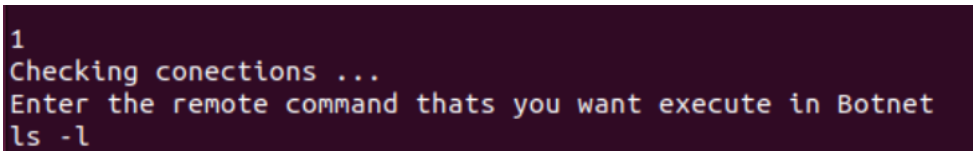


Figura 17: Imagen de la opción 1 del menú. (Launch Campaign)

   Se introduce el comando que quieres ejecutar y pulsando enter se enviaría a todas las máquinas.

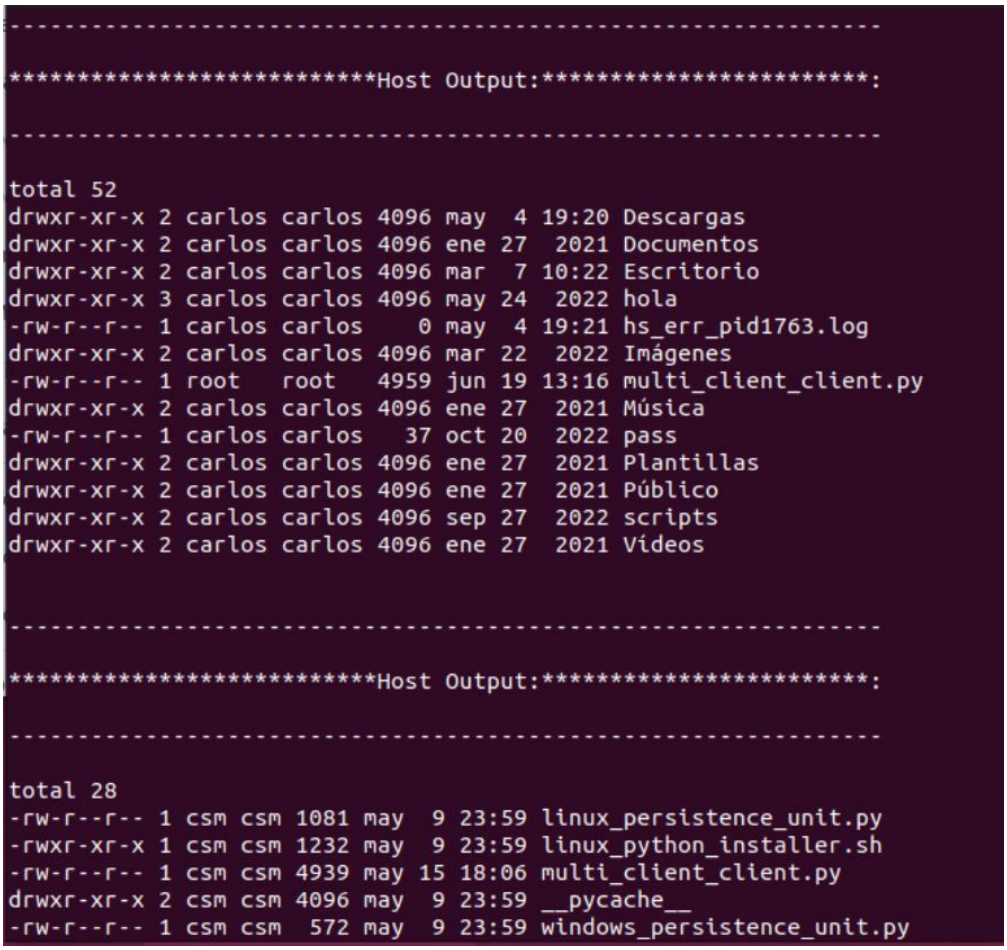


Figura 18: output recibido después de ejecutar la opción 1.

Se recibe el output de las máquinas.

2. Open Reverse TCP Shell host.

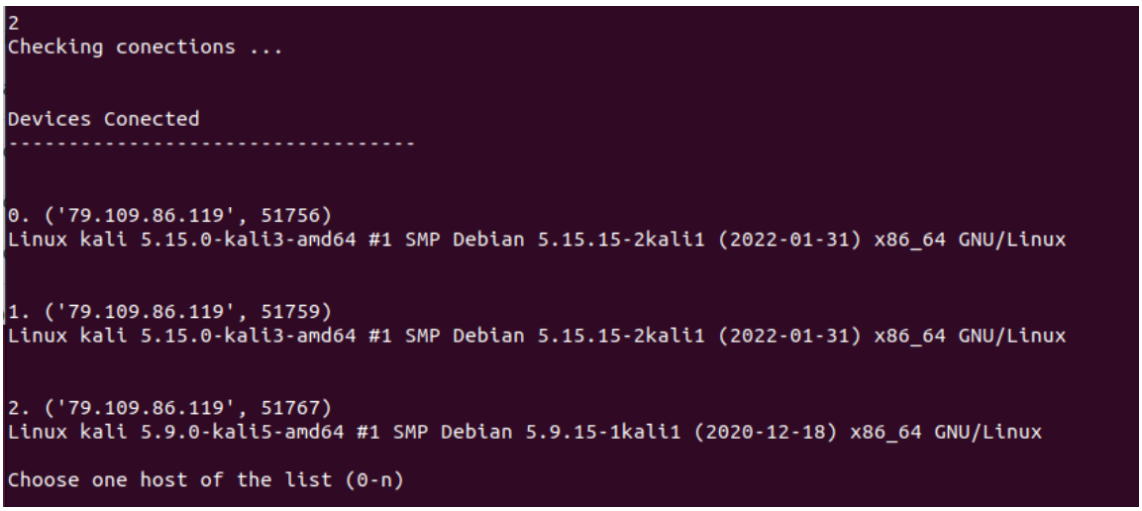


Figura 19: Opción 2 del menú.

Hay que seleccionar un host del listado.

Figura 20: Imagen de la reverse shell abierta de una máquina.

La reverse Shell se abre y puedes interactuar con la máquina.

```
/home/csm/Escritorio/TFG_Botnet/socketTFG/Client $> exit


        Main
-----------------------------------------------


 1. Launch Campaign (Execute multidifusion Remote Command)
 2. Open Reverse TCP Shell host
 3. Download remote file from host
 4. Upload file to remote Host
 5. System Monitor
 0. Exit?
```

Figura 21: Comando exit para salir al menú principal

Para salir, teclear “exit” y vuelve al menú principal.

3. Download remote file from host.

```
3
Checking conections ...


Devices Conected
---------------------------------


0. ('79.109.86.119', 52887)
Linux kali 5.9.0-kali5-amd64 #1 SMP Debian 5.9.15-1kali1 (2020-12-18) x86_64 GNU/Linux


1. ('79.109.86.119', 52891)
Linux kali 5.15.0-kali3-amd64 #1 SMP Debian 5.15.15-2kali1 (2022-01-31) x86_64 GNU/Linux

Choose one host of the list (0-n)

0
('79.109.86.119', 52887)  Ok.  Selected Host.



Introduce the Remote file path (absolute path)/home/carlos/Escritorio/hola.txt

Introduce the local directory for save the file (absolute path)
 ej:  C:\user\myfolder  or  /user/myfolder /home/usuario/Escritorio/

Donwloading... hola.txt

File saved
```

Figura 22: Opción 3 del menú. (descarga de fichero remoto).

Seleccionar un host de la lista de conectados. Después, le solicitará la ruta remota del fichero que quiere descargar de la máquina.

Debe introducir también la ruta de almacenamiento local donde guardará el archivo.

Finalmente lo descarga.

4. Upload file to remote host.

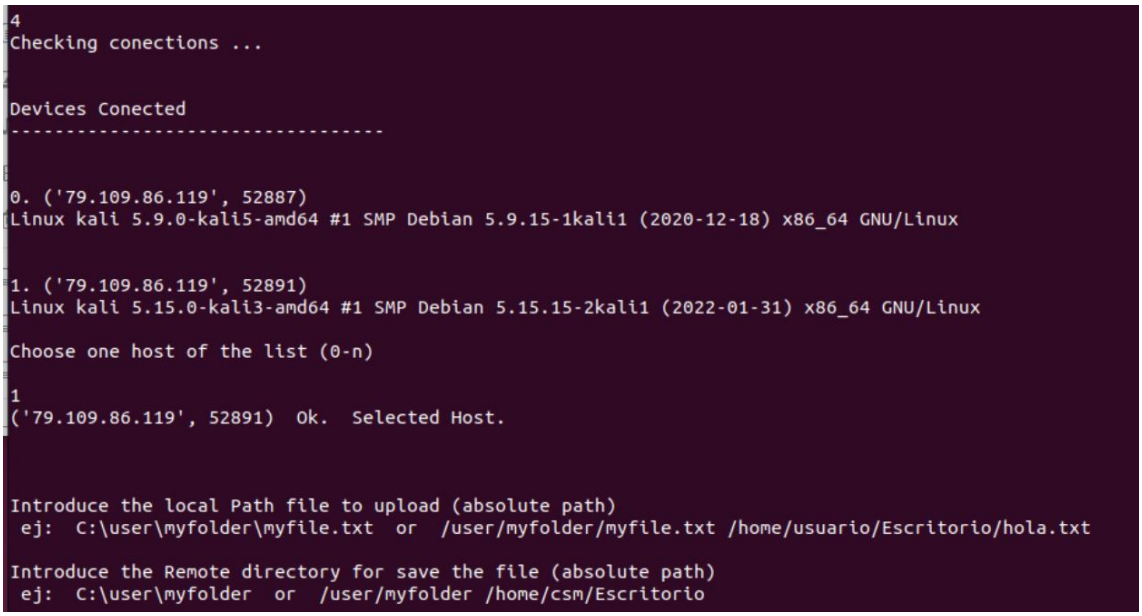


Figura 23:Opción 4 del menú. (subida de fichero a máquina remota).

El sistema le pedirá que introduzca el host al que quiere subirle el fichero.
Tendrá que especificar la ruta donde se ubique su script localmente.
También es requerida la ruta de almacenamiento en la máquina remota.

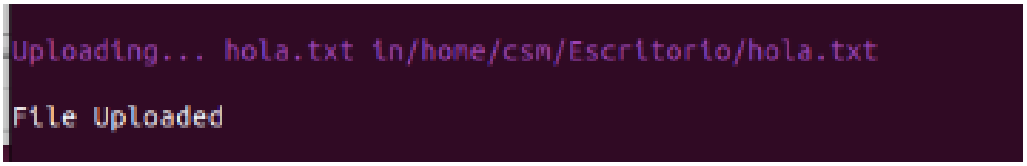


Figura 24: Resultado de ejecución de la opción 4.

5. System monitor.

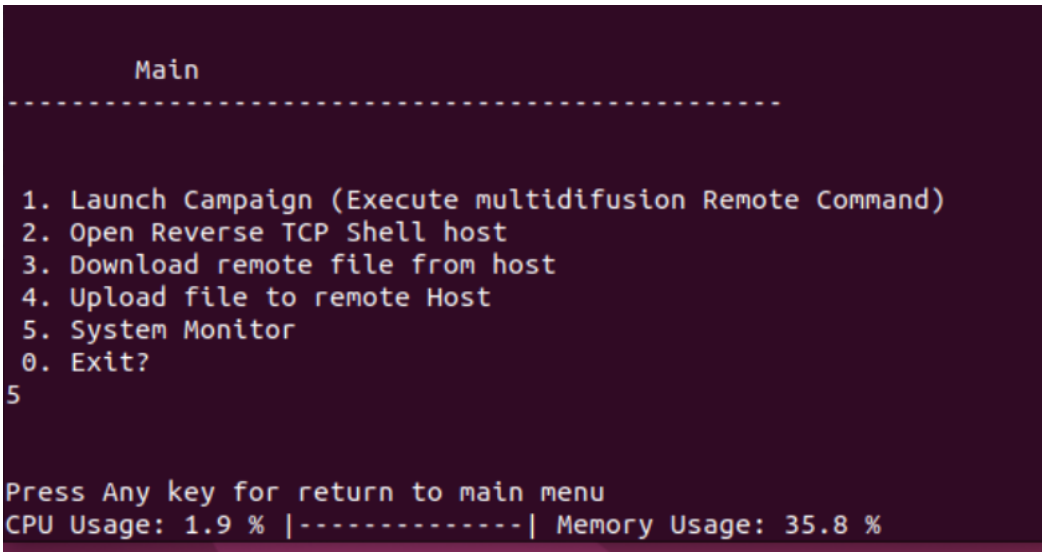


Figura 25: Imagen de la opción 5 del servidor. (monitor de recursos del sistema).

Al pulsar la opción 5, el sistema le muestra mediciones de consumo de recursos en tiempo real con una tasa de refresco de 2 segundos.

0. Exit.

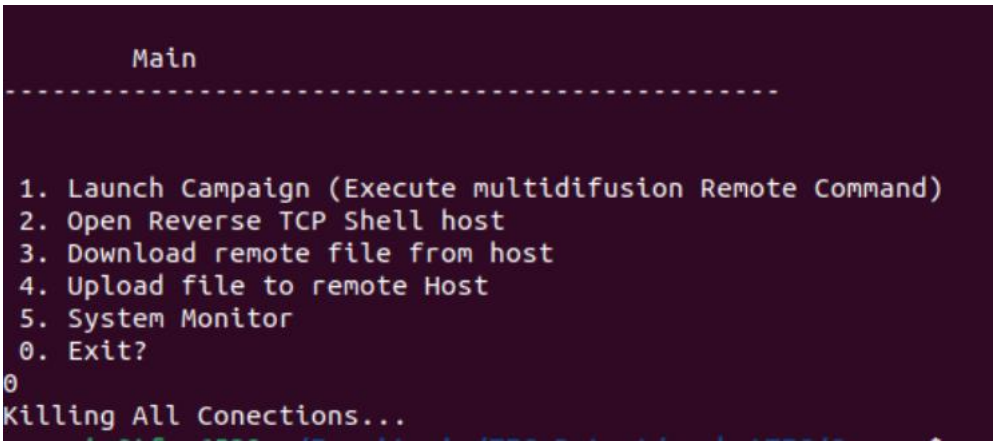


Figura 26: Opción 0 del menú.

Cuando selecciona la opción salir, el servidor ordenara el cierre de conexión a todas las máquinas que se encuentren conectadas.

8.4. Explicación de uso del generador de Payloads.

Para la herramienta de payload-generator, deberemos navegar hasta el directorio Payloads del proyecto, y abrir la carpeta Payload_generator.

Ejecutar:

$pip install -r requirements.txt

para instalar las dependencias.

Desplegar la herramienta mediante la ejecución del comando:

python3 payload_generator.py

Se abrirá la siguiente pantalla:

Figura 27: Imagen del Payload Generator desplegado.

<<Elegir el tipo de sistema para generar el payload>>.

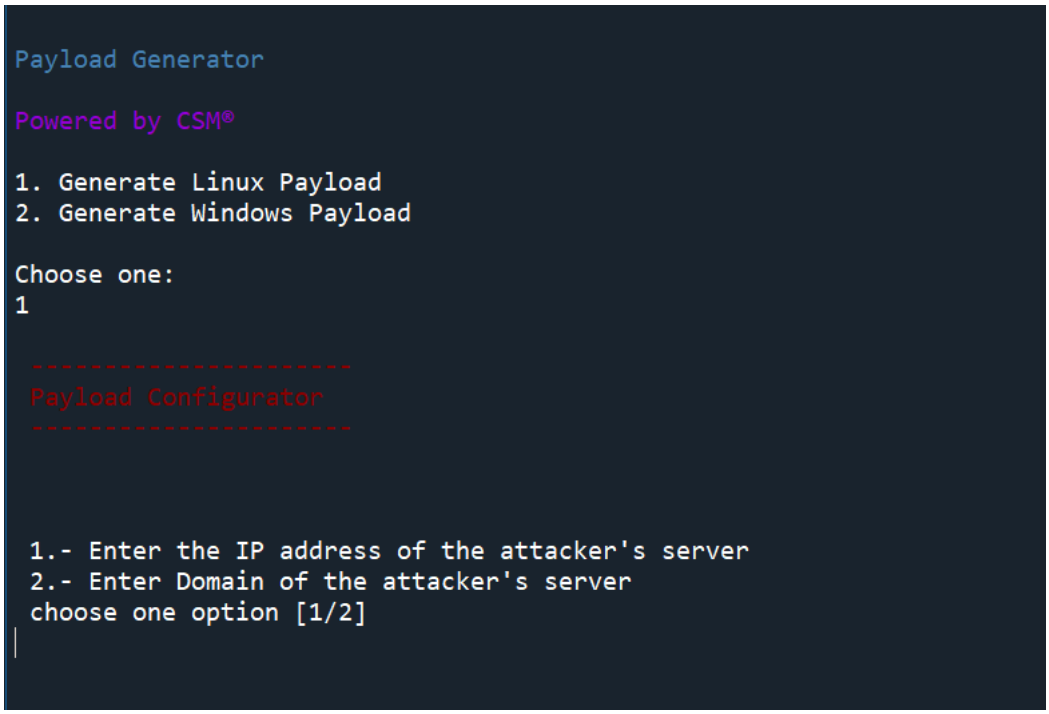


Figura 28: Imagen del configurador de payloads.

Aquí elegir 1 si vas a introducir una dirección ip, o 2 si tienes el dominio del servidor.

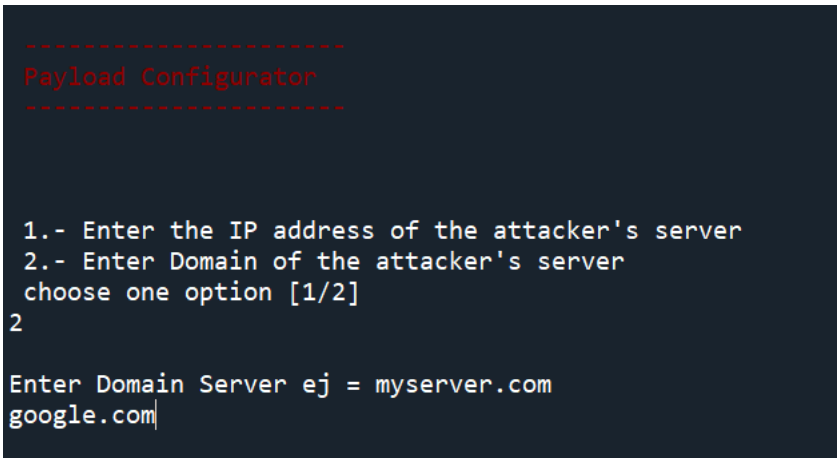


Figura 29: Opción para indicar la dirección del servidor de atacante en el payload.

Introducir la dirección del servidor de atacante.

Figura 30:Resultado del generador de payloads.

Introducir el puerto de escucha del servidor, y si todo ha ido bien se generará el paquete, dentro del directorio Payload_generator.

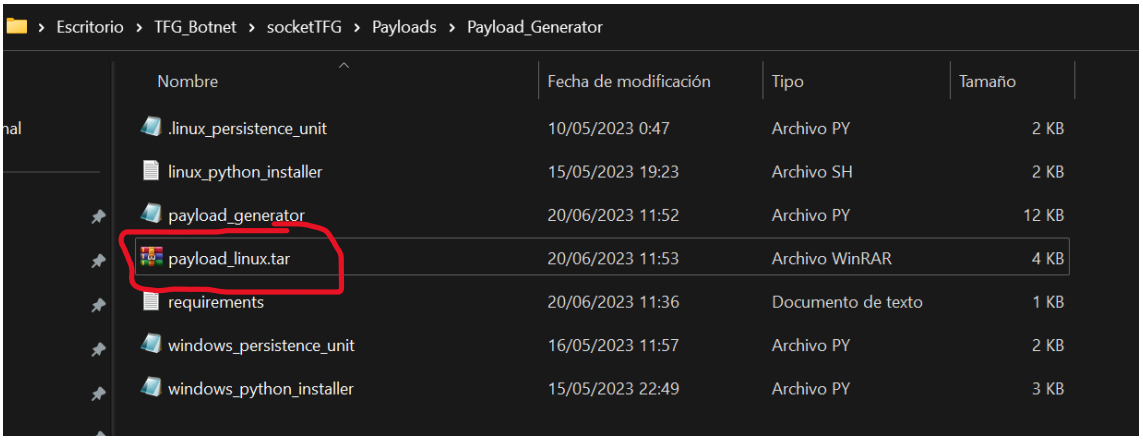


Figura 31: Fichero generado por el generador de payloads.

El fichero será el de payload_linux.tar.

Realizar el mismo proceso para generar un payload de Windows.

Descomprimir el paquete en la máquina objetivo, y ejecutar el instalador de Python (con permisos de administración).

Capítulo 9.

Coste del proyecto.

En este capítulo se realizará un estudio completo del coste real que ha tenido el proyecto, con la finalidad de hacer una comparativa con el método de estimación.

Las tablas siguientes contienen la información de las tareas, clasificadas por la historia a la que pertenecen y el número de horas que han sido dedicadas en su realización.

9.1. Coste horario de las tareas del proyecto.

Tarea	Descripción	Horas
T-01-01	*Investigación sobre las botnets.* Realizar un estudio de las botnets, y sus metodologías de operación.	5
T-01-02	Formación sobre sockets y las librerías en Python.	12
T-01-03	Reunión inicial con el tutor y revisión de la idea del proyecto	1
T-01-04	Diseño de la arquitectura y modelado de la misma.	9
T-01-05	Primeras pruebas haciendo scripts de sockets en Python.	13
Tareas de la Historia de usuario 01		

Tabla 23: Tareas de la historia de usuario 01. (Coste real).

Tarea	Descripción	Horas
T-02-01	Elaboración de un guión de la memoria.	6
T-02-02	Creación del entorno de desarrollo.	10
T-02-03	Reunión inicial de Sprint con el tutor y actualización del Sprint Backlog.	3
T-02-04	Programación de un script funcional de un cliente en Python.	18
T-02-05	Testing y corrección del script cliente.	2
Tareas de la Historia de usuario 02		

Tabla 24: Tareas de la historia de usuario 02. (Coste real).

Tarea	Descripción	Horas
T-03-01	Reunión inicial de Sprint con el tutor y actualización del Sprint Backlog.	3
T-03-02	Diseño e implementación del socket servidor.	11
T-03-03	Estudio de las Reverse Shell.	6
T-03-04	Programación de la funcionalidad Reverse Shell remota en cliente y servidor.	23
T-03-05	Programación de la funcionalidad para lanzar campañas a los bots.	14
Tareas de la Historia de usuario 03		

Tabla 25: Tareas de la historia de usuario 03. (Coste real).

Tarea	Descripción	Horas
T-04-01	Reunión inicial de Sprint con el tutor y actualización del Sprint Backlog.	2
T-04-02	Reestructuración del código e implementación de Excepciones para manejar situaciones adversas y aumentar la fiabilidad de la herramienta.	19
T-04-03	Estudio del protocolo SSL.	8
T-04-04	Implementación de SSL en las comunicaciones.	13
T-04-05	Implementación de funciones manejadoras de ordenes en cliente y servidor, con la finalidad de hacer más escalable y mantenible el código.	10
Tareas de la Historia de usuario 04		

Tabla 26: Tareas de la historia de usuario 04. (Coste real).

Tarea	Descripción	Horas
T-05-01	Reunión inicial de Sprint con el tutor y actualización del Sprint Backlog.	2
T-05-02	Diseño de un protocolo de envío y recepción de archivos a través de sockets.	10
T-05-03	Implementación de los métodos de subida y bajada de ficheros remotos.	12
T-05-04	Test y corrección de las funcionalidades subida y bajada de ficheros.	6
Tareas de la Historia de usuario 05		

Tabla 27: Tareas de la historia de usuario 02. (Coste real).

Tarea	Descripción	Horas
T-06-01	Reunión inicial de Sprint con el tutor y actualización del Sprint Backlog.	3
T-06-02	Estudio de técnicas de persistencia de malware en sistemas operativos.	8
T-06-03	Estudio de Daemons en Windows y Linux y de ejecución silenciosa.	9
T-06-04	Programación de un generador de payloads dinámico para BotCommander.	15
T-06-05	Test y corrección global del software.	7
Tareas de la Historia de usuario 06		

Tabla 28: Tareas de la historia de usuario 06. (Coste real).

Tarea	Descripción	Horas
T-07-01	Elaboración de la introducción y los objetivos del proyecto	2
T-07-02	Confección de una planificación del trabajo y redacción de la metodología. Elaboración de Presupuestos.	8
T-07-03	Documentación del estado del arte.	4
T-07-04	Documentación de los sockets cliente y servidor.	6
T-07-05	Documentación sobre las tecnologías utilizadas.	3
T-07-06	Documentación del flujo de trabajo de la aplicación.	10
T-07-07	Diseño y documentación de un plan de riesgos	5

T-07-08	Elaboración del coste final del proyecto y confección de las conclusiones.	11
Tareas de la Historia de usuario 07		

Tabla 29: Tareas de la historia de usuario 07. (Coste real).

Cabe destacar que no se han añadido ni eliminado tareas con respecto a la etapa de planificación.

El reparto de tareas en sprints tampoco ha cambiado, y se han respetado las fechas de comienzo y fin de cada uno, si bien el sprint 3 ha resultado más cargado debido a imprevistos relacionados con la complejidad de una funcionalidad.

Sprint	Inicio	Fin	Tareas	Total Horas
1	01/03/2023	19/03/2023	T-01-01 T-01-02 T-01-03 T-01-04 T-01-05 T-07-01	42
2	20/03/2023	09/04/2023	T-07-02 T-02-01 T-02-02 T-02-03 T-02-04 T-02-05	45
3	10/04/2023	30/04/2023	T-03-01 T-03-02 T-03-03 T-03-04 T-03-05 T-07-03 T-07-04	67
4	01/05/2023	21/05/2023	T-04-01 T-04-02 T-04-03 T-04-04 T-04-05 T-07-05	55
5	22/05/2023	11/06/2023	T-05-01 T-05-02 T-05-03 T-05-04 T-07-06 T-07-07	45
6	12/06/2023	30/06/2023	T-06-01 T-06-02 T-06-03 T-06-04 T-06-05 T-07-08	53
total				307

Tabla 30: Calendario final del proyecto.

9.2. Coste real del proyecto.

En esta sección se realizará el cálculo del coste total del proyecto. Los recursos humanos se han basado en las horas reales incurridas en la realización de cada tarea, así como la media de los salarios de los roles involucrados en una media jornada de 4h.

En cuanto a recursos técnicos han sido similares a los de la estimación a diferencia de la adición del gasto de internet que no fue contemplado.

9.2.1 Recursos Técnicos.

Detalle del presupuesto:

Recurso	Coste/ud	%	Total
Ordenador portátil ASUS	750,00 €	6,67	50,02
Máquina virtual Uva			00,00
Distribución Anaconda			00,00
GitHub Desktop			00,00
Windows 11 Pro (incluido)			00,00
Office 365 (Licencia de la UVA)			00,00
Conexión wifi internet			30,00
		Total:	70,02 €

Tabla 31: Recursos Técnicos (Coste final).

9.2.2. Recursos Humanos.

El coste se ha calculado mediante las horas que ha dedicado cada rol en el proyecto.

En total, ha tenido una duración de 307 horas, y contamos con un tiempo de 4 meses para su ejecución. Se ha confeccionado un reparto de tareas para calcular el número de horas que ha dedicado cada rol con la finalidad de calcular su coste final:

Rol	Tareas	Horas totales
Jefe de proyecto	T-01-03, T-02-01, T-02-03, T-03-01, T-04-01, T-05-01, T-06-01, T-07-01, T-07-02, T-07-07, T-07-08	35h
Analista	T-01-01, T-01-04, T-03-03, T-04-03, T-05-02, T-06-02, T-06-03, T-07-03, T-07-05	61h
Desarrollador Python	T-01-02, T-01-05, T-02-02, T-02-04, T-02-05, T-03-02, T-03-04, T-03-05, T-04-02, T-04-04, T-04-05, T-05-03, T-05-04, T-06-04, T-06-05, T-07-04, T-07-06	211h
	Total	307 h

Tabla 32: Calculo total de las horas dedicadas al proyecto.

Rol	Salario bruto /hora	Horas	Total Bruto	Costes Sociales (30%)	Coste Total (Empresa)
Jefe de proyecto	40,00	35	1400,00	420,00	1820,00
Analista	25,00	61	1525,00	457,50	1982.50
Desarrollador Python	15,00	211	3165,00	949,50	4114,50
		Total			7917,00 €

Tabla 33: Costes finales del proyecto.

Por lo que el coste real del proyecto será **7987,02€** contando los recursos humanos y técnicos.

Capítulo 10.

Conclusiones y trabajo futuro.

10.1. Conclusiones

A lo largo del presente documento se han proporcionado todos los detalles de la realización del proyecto de fin de grado, consistente en el estudio e implementación de una botnet centralizada. El proyecto fue una idea personal que surgió hace un año, y que fui madurando gracias a mi pasión por la ciberseguridad y mi constante formación.

También ha sido de agradecer la realización de mis estudios complementarios en *The Bridge*, donde he podido aprender técnicas y conceptos clave que han sido aplicados en la resolución de incógnitas en el proyecto, como el funcionamiento básico de los sockets.

Todo empezó con el uso de la herramienta *netcat*, comúnmente utilizada para establecer comunicaciones entre dos máquinas, lo que despertó mi interés en ella para investigar la posibilidad de abstraer su funcionamiento y crear una infraestructura que permita gestionar multitud de conexiones, así como proporcionar funcionalidades deseadas para un atacante, siempre desde un punto de vista ético y con la idea de poder mejorar la seguridad de los sistemas actuales.

En primer lugar, tuve que formarme en diversos conceptos como las redes, y su comunicación mediante sockets, algo nuevo para mí, por lo que decidí investigar el funcionamiento de herramientas parecidas e intentar replicar algunos de sus comportamientos.

El conocimiento de los protocolos de red también ha sido una pieza clave en el desarrollo, de ahí la necesidad de formarme en *TCP, TLS e IP.*

Elegí la plataforma Python ya que en ciberseguridad es un estándar, gracias a su amplia gama de recursos y librerías con una documentación completa.

Cuando el conocimiento adquirido empezó a brindarme ideas, el diseño de la comunicación entre las máquinas pasó a ser el foco de atención, apostando por la arquitectura cliente-servidor en la que adjudiqué el rol de servidor a el servidor de comando y control que jugaría el papel del atacante. El cliente pasaría a adoptar el rol del ordenador “zombie” a expensas de recibir órdenes. Esta primera decisión resultó simplificar mucho el problema, pudiendo separarlo en problemas más sencillos y asequibles. Empecé a tratar al cliente y al servidor como entes independientes pero con una clara dependencia entre sí.

El desarrollo de ambos roles se hizo de manera simultánea, ya que ambos tienen que ir coordinados, es decir, si el servidor envía, el cliente debe de estar programado para recibirlo y actuar en consecuencia.

La tecnología socket de Python utilizada, permite trabajar a bajo nivel con sockets, algo que es muy beneficioso desde el punto de vista de la escalabilidad y flexibilidad, sin embargo, la velocidad de desarrollo es lenta ya que tienes que diseñar y programar funcionalidades de uso común partiendo desde cero, como fue el caso del envío y recepción de ficheros a través de la red, algo que con un servidor http resulta muy sencillo debido a que el protocolo te abstrae de las operaciones básicas.

Otro de los inconvenientes de trabajar a bajo nivel, es que el software no es lo suficientemente maduro y es altamente necesario realizar sesiones de test y corrección, manejando múltiples excepciones prolongando el ciclo de desarrollo del software.

Desde el punto de vista del rendimiento, una de las ventajas que he observado de utilizar una conexión TCP "plana" es que es mucho más rápida que http, lo cual brinda ventajas de rendimiento, sobre todo en el servidor central.

Se ha podido comprobar también la facilidad de evasión de los dispositivos de seguridad teniendo un poco de conocimiento. Algo que puede ayudar a concienciar a los usuarios y a las empresas para adoptar medidas más estrictas de monitorización o implementación de sistemas complementarios que ayuden a proteger los sistemas.

En relación con mis objetivos personales, he podido:

- Ampliar mis conocimientos en el funcionamiento de los sistemas operativos.
- Profundizar en los métodos de comunicación en red, así como ganar conocimiento acerca de los protocolos *TCP/IP y TLS*.
- Aprender técnicas de evasión de sistemas de seguridad y de persistencia en dispositivos.
- Mejorar mis habilidades y desempeño con *Python*.
- Gestionar un proyecto de ciberseguridad, lo que me ha proporcionado una visión más realista del sector.

Desde un punto de vista más personal, se ha notado un incremento de organizaciones que operan con el uso de botnets para obtener sus objetivos lanzando ataques a empresas con el chantaje de no derribar sus servicios a cambio de una retribución económica. Este tipo de ataques es uno de los más agresivos ya que es difícil establecer reglas en los sistemas de detección que frenen la amenaza, ya que se tratan de redes distribuidas resultando complejo diferenciar cuando se trata de una solicitud maliciosa o legítima. Esto para mí ha supuesto un gran interés profesional, así como una motivación para desarrollar el proyecto, aprender, y poder aportar mi conocimiento en un futuro para ayudar a las empresas.

10.2. Trabajo Futuro

A continuación, he recopilado una serie de mejoras que podrían contemplarse en etapas futuras del proyecto, que ayudarían a obtener un mejor rendimiento, dar soporte a otras funcionalidades y que el software sea más fiable:

- Implementar una base de datos en el servidor para guardar la información de la conexión de las máquinas, de esta manera, contaremos con una unidad de persistencia de datos y hará que la recuperación ante incidentes sea más efectiva.
- De cara a ganar rapidez en la realización de las campañas de ataque, se puede diseñar y e implementar Payloads que automaticen el ataque que se quiere desempeñar, es decir, si quieres realizar un ataque Ddos mediante la técnica de SYN Flood o inundación de flags SYN, sería una buena idea programar un script o un comando que esté listo para usarse en cualquier momento.
- Sería interesante diseñar e integrar una interfaz gráfica basada en tecnologías web para facilitar el uso de la aplicación.
- Ya que el soporte para la comunicación está implementado, solo queda echarle imaginación y ver que funcionalidades pueden ser útiles si se añaden, buscar la manera de llevarlas a cabo mediante una secuencia de comandos o confeccionando scripts, de manera que se ejecuten o sean enviados a las máquinas simplemente presionando un botón.

Bibliografía

[1]. Atlassian (no date) *Scrum: Qué es, cómo funciona y cómo empezar*, *Atlassian*. Available at: **https://www.atlassian.com/es/agile/scrum** (Accessed: 14 April 2023).

[2]. *Botnet* (2023) *Wikipedia*. Available at: **https://es.wikipedia.org/wiki/Botnet** (Accessed: 11 March 2023).

[3]. Chuuck (2016) *Ventajas de Usar Scrum en tu proyecto*, *Platzi*. Available at: **https://platzi.com/blog/ventajas-scrum/#:~:text=Scrum%20permite%20a%20los%20equipos%20de%20desarrollo%20priorizar%20los%20m%C3%B3dulos,las%20cambiantes%20necesidades%20del%20proyecto**. (Accessed: 17 April 2023).

[4]. *Cliente-Servidor* (2023) *Wikipedia*. Available at: **https://es.wikipedia.org/wiki/Cliente-servidor** (Accessed: 05 May 2023).

[5]. Lahtela, M. and Kaplan, P. (Provenance) (1966) *ES*, *Amazon*. Available at: **https://aws.amazon.com/es/what-is/scrum/** (Accessed: 22 April 2023).

[6]. Pascualena, J.S. (2022) *¿Cuánto Cuesta contratar un trabajador?*, *Infoautonomos*. Available at: **https://www.infoautonomos.com/blog/cuanto-cuesta-contratar-un-trabajador/** (Accessed: 22 June 2023).

[7]. *Socket - interfaz de red de bajo nivel* (no date) *socket - interfaz de red de bajo nivel - documentación de Python - 3.10.11*. Available at: **https://docs.python.org/es/3.10/library/socket.html** (Accessed: 22 March 2023).

[8]. *Socket de internet* (2022) *Wikipedia*. Available at: **https://es.wikipedia.org/wiki/Socket_de_Internet** (Accessed: 09 May 2023).

[9]. *SSL -empaquetador o wrapper TLS/SSL para objetos de tipo socket* (no date) *Python documentation*. Available at: **https://docs.python.org/es/dev/library/ssl.html** (Accessed: 26 June 2023).

[10]. Villanueva, A. (2021) *Botnet: Redes de Cibercriminales - OSTEC: Segurança Digital De resultados*, *OSTEC*. Available at: **https://ostec.blog/es/aprendizaje-descubrimiento/botnet-redes-de-cibercriminales/?cn-reloaded=1** (Accessed: 07 March 2023).

[11]. Rockikz, A. (2019) *How to create a reverse shell in python*, *Python Code*. Available at: **https://www.thepythoncode.com/article/create-reverse-shell-python** (Accessed: 10 April 2023).

[12]. Digamber (2023) *How to create socket server with multiple clients in python*, *positronX.io*. Available at: **https://www.positronx.io/create-socket-server-with-multiple-clients-in-python/** (Accessed: 02 May 2023).

[13]. *Cómo crear un diagrama de gantt en excel [Cronograma Usando Los Gráficos]* (2017) *YouTube*. Available at: **https://www.youtube.com/watch?v=chR6kx4btDQ** (Accessed: 10 June 2023).

[14]. Trello. Trello. url: **https://trello.com** (Accessed: 3 May 2023)

[15]. *Chatgpt* (no date) *ChatGPT*. Available at: **https://openai.com/chatgpt** (Accessed: 05 May 2023).

[16]. *¿Qué es una Reverse Shell?* Available at: **https://keepcoding.io/blog/que-es-una-shell-inversa/** (Accessed: 09 May 2023).

[17]. *¿Qué es un ataque DDos?* Available at: https://www.akamai.com/es/glossary/what-is-ddos (Accessed: 09 May 2023).
[18]. *¿Salarios sofware en España?* Available at: https://es.talent.com/salary?job=analista+de+software#:~:text=El%20salario%20analista%20de%20software,hasta%20%E2%82%AC%2051.145%20al%20a%C3%B1o. (Accessed: 22 April 2023).
[19]. *¿Salario programador python?* Available at: https://www.tokioschool.com/formaciones/cursos-programacion/python/sueldo/#:~:text=El%20sueldo%20medio%20de%20un,368%20euros%20brutos%20anuales. (Accessed: 22 April 2023).
[20]. *¿Salario programador jefe de proyecto?* Available at: https://blog.euncet.com/project-manager-que-es-competencias-y-salario/#:~:text=La%20retribuci%C3%B3n%20media%20de%20un,los%2080.000%E2%82%AC%20brutos%20anuales. (Accessed: 22 April 2023).
[21]. *Persistencia en sistemas operativos* Available at: https://keepcoding.io/blog/rutas-en-donde-se-puede-encontrar-persistencia/#:~:text=La%20persistencia%20se%20refiere%20a,trav%C3%A9s%20de%20reinicios%20del%20sistema. (Accessed: 10 April 2023).