Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN

Creación de una botnet

Alumno: Javier Barrientos González

Tutor: Blas Torregrosa García

A mi familia.

Agradecimientos

Agradecimientos a mi familia por todo el apoyo que me ha dado a lo largo de este camino.

Resumen

El objetivo de este proyecto es diseñar y desarrollar una *botnet* con fines educativos utilizando el protocolo IP. Para su desarrollo se han estudiado varios ejemplos para analizar sus métodos de comunicación, protección y ocultación.

Este proyecto requiere la implementación de un servidor central o C&C contra el que se comunicará cada *bot*, al igual que un sito web para que el administrador de la *botnet* pueda controlarlos. Cada *bot* estará desplegado remotamente y se le podrá asignar diferentes tareas por lo que también se desarrollará su lógica.

Los *bots* tienen como objetivo aquellos ordenadores cuyo sistema operativo sea base Windows.

El trabajo se ha desarrollado empleando Python como lenguaje de programación, gRPC para la comunicación entre los *bots* y el servidor y Flask como *framework* para el desarrollo de la aplicación web que se utilizará para administrar la *botnet*.

Todo el proyecto ha sido elaborado siguiendo la metodología ágil Scrum.

Abstract

The objective of this project is to design and develop a botnet for educational purposes using the IP protocol. For its development, several examples have been studied to analyze its communication, protection and concealment methods.

This project requires the implementation of a central or C&C server against which each bot will communicate, as well as a website so that the botnet administrator can control them. Each bot will be remotely deployed and can be assigned different tasks so its logic will also be developed.

The bots target those computers whose operating system is Windows based.

The work has been developed using Python as the programming language, gRPC for communication between the bots and the server, and Flask as a framework for the development of the web client that will be used to manage the botnet.

The entire project has been developed following the agile Scrum methodology.

Índice general

Agradecimientos **III**

Resumen **V**

Abstract **VII**

Lista de figuras **XV**

Lista de tablas **XVII**

1. Introducción, objetivos y motivación **1**

1.1. Introducción 1

1.2. Motivación 1

1.3. Objetivos del proyecto 2

1.3.1. Objetivos de desarrollo 2

1.3.2. Objetivos personales 2

1.4. Estructura de la memoria 2

2. Estado del arte **5**

2.1. *Botnet* 5

2.1.1. ¿Qué es una *botnet*? 5

2.1.2. Historia 5

2.1.3. Infección del dispositivo 6
2.1.4. Tipos de *botnets* 7
2.1.5. Objetivos 9
2.1.6. El impacto de las *botnets* 10
2.1.7. El futuro de las *botnets* 10
2.2. *Cyber Kill Chain* 10
2.2.1. *Cyber Kill Chain* y las *botnets* 12
2.3. MITRE ATT&CK 12

3. Planificación y desarrollo del proyecto **15**
3.1. Scrum 15
3.1.1. Introducción 15
3.1.2. Equipo Scrum 16
3.1.3. Eventos 16
3.2. Planificación 17
3.3. Tareas realizadas 18
3.3.1. *Sprint* 0 20
3.3.2. *Sprint* 1 21
3.3.3. *Sprint* 2 22
3.3.4. *Sprint* 3 23
3.3.5. *Sprint* 4 24
3.3.6. *Sprint* 5 24
3.3.7. *Sprint* 6 25
3.3.8. *Sprint* 7 27
3.3.9. Sprint 8 27

4. Tecnologías utilizadas **29**
4.1. Python 29

4.1.1. Requests . . . 29
4.1.2. pytz . . . 29
4.1.3. PyScreeze . . . 30
4.1.4. PyInstaller . . . 30
4.2. gRPC . . . 30
4.2.1. Protocol Buffers . . . 30
4.3. Flask . . . 30
4.4. SQLite . . . 31
4.4.1. DB Browser . . . 31
4.5. Leaflet . . . 31
4.6. Bootstrap . . . 31
4.6.1. Bootstrap Table . . . 31
4.7. Git . . . 32
4.7.1. GitLab . . . 32
4.8. LaTeX . . . 32
4.8.1. Overleaf . . . 32
4.9. Astah . . . 32
4.10. Trello . . . 33

5. Plan de riesgos y estimación de costes **35**
5.1. Plan de riesgos . . . 35
5.1.1. Riesgos encontrados en el proyecto . . . 36
5.2. Presupuesto . . . 38

6. Diseño **41**
6.1. Arquitectura cliente/servidor . . . 41
6.1.1. Características . . . 41

6.1.2. Ventajas 42
6.1.3. Desventajas 43
6.1.4. Arquitectura cliente/servidor en este proyecto 43
6.2. gRPC 43
6.2.1. Características 43
6.2.2. Protocol Buffers 44
6.2.3. Funcionamiento 44
6.2.4. Ventajas 46
6.2.5. Desventajas 46
6.2.6. gRPC en este proyecto 46
6.3. Modelo de dominio 47
6.4. Modelo de despliegue 47

7. Implementación **49**

7.1. Entorno de laboratorio 49
7.2. Implementación de la *botnet* 50
7.2.1. Comunicación con C&C 51
7.2.2. Persistencia 52
7.2.3. Tareas implementadas 52
7.2.4. Aplicación web 54
7.3. Estructura del código 54
7.3.1. Estructura del código del servidor gRPC 56
7.3.2. Estructura del código de la aplicación web 57
7.3.3. Estructura del código del *bot* 58
7.4. Licencia 58

8. Pruebas **59**

9. Despliegue **67**

9.1. Preparación del entorno 67

9.2. Despliegue del servidor 69

9.2.1. Servidor gRPC 69

9.2.2. Aplicación web 70

9.3. Despliegue del *bot* 71

9.3.1. Creación del *bot* 71

9.3.2. Ejecución del *bot* en la máquina Windows 71

10. Conclusiones y líneas futuras **73**

10.1. Conclusiones finales 73

10.2. Líneas futuras 74

A. Manual de usuario **75**

B. Código del fichero .proto **83**

Bibliografía **87**

Índice de figuras

2.1. *Botnet* centralizada [10] 8

2.2. *Botnet* descentralizada [10] 8

2.3. *Cyber Kill Chain* 11

2.4. Pasos que realiza una *botnet* 12

3.1. Desarrollo Scrum [16] 17

5.1. Entorno de riesgos de un proyecto. 35

6.1. Arquitectura cliente/servidor 42

6.2. Diagrama de conceptos [4] 45

6.3. Modelo de dominio 47

6.4. Modelo de despliegue 48

7.1. Entorno de laboratorio 50

7.2. Comportamiento de la *botnet* 51

A.1. Apariencia final de la página principal. 75

A.2. Apariencia final de la página de los *bots*. 76

A.3. Apariencia final de la página de información de un *bot* con la información y la geolocalización. 77

A.4. Apariencia final de la página de información de un *bot* con las tareas. 78

A.5. Apariencia final de la página de las tareas. 79

A.6. Apariencia final de la página del formulario para añadir una nueva tarea. . . 79

A.7. Apariencia final de la página de información de una tarea. 80

A.8. Apariencia final de la página de *logs*. 80

A.9. Apariencia final de la página de archivos. 80

A.10. Apariencia final de la página del mapa. 81

A.11. Apariencia final de la página del mapa con el diálogo del *bot* seleccionado. . . 81

Índice de tablas

3.1. Planificación de los *sprints* 18
3.2. Historias de usuario 19
3.3. Continuación de historias de usuario 20
3.4. Tareas Sprint 0 21
3.5. Tareas *sprint* 1 22
3.6. Tareas *sprint* 2 22
3.7. Tareas *sprint* 3 23
3.8. Tareas *sprint* 4 24
3.9. Tareas *sprint* 5 25
3.10. Tareas *sprint* 6 26
3.11. Tareas *sprint* 7 27
3.12. Tareas *sprint* 8 28

5.1. Riesgo de falta de formación 36
5.2. Riesgo de enfermedad 37
5.3. Riesgo de cambios en los requisitos 37
5.4. Riesgo de planificación optimista 37
5.5. Riesgo de fallo del entorno de laboratorio 37
5.6. Riesgo de falta de seguimiento 38
5.7. Riesgo de cambios en las tecnologías utilizadas 38

5.8. Presupuesto final 39

8.1. Prueba ejecución del *bot* en el entorno de laboratorio 59

8.2. Prueba de comunicación del *bot* con el servidor encendido 59

8.3. Prueba de comunicación del *bot* con el servidor apagado 60

8.4. Prueba del *bot* solicitando tareas al servidor 60

8.5. Prueba del *bot* realizando la tarea de ejecución de un comando 60

8.6. Prueba del *bot* realizando la tarea de subir un fichero a la máquina remota . 60

8.7. Prueba del *bot* al realizar la tarea de enviar un fichero inexistente 61

8.8. Prueba del *bot* al realizar la tarea de enviar un fichero 61

8.9. Prueba del *bot* al realizar la tarea de tomar una captura de pantalla 61

8.10. Prueba del *bot* al realizar la tarea de apagarse 61

8.11. Prueba del *bot* al realizar la tarea de eliminarse 62

8.12. Prueba del *bot* enviando las tareas completadas 62

8.13. Prueba acceso a la página de los *bots* con el servidor apagado 62

8.14. Prueba de la aplicación accediendo a la página de los *bots* 62

8.15. Prueba de la aplicación accediendo a un *bot* existente 63

8.16. Prueba de la aplicación accediendo a un *bot* que no existe 63

8.17. Prueba de la aplicación accediendo a la página de las tareas con el servidor apagado 63

8.18. Prueba de la aplicación accediendo a la página de las tareas 63

8.19. Prueba de la aplicación accediendo a la página de añadir prueba sin tener *bots* disponibles 64

8.20. Prueba de la aplicación agregando una nueva tarea 64

8.21. Prueba de la aplicación agregando una nueva tarea sin completar los campos 64

8.22. Prueba de la aplicación descargando ficheros 64

8.23. Prueba de la aplicación descargando ficheros que no existen 65

Capítulo 1

Introducción, objetivos y motivación

1.1. Introducción

En los últimos años se ha podido ver un considerable aumento de dispositivos inteligentes IoT (Internet of Things) a lo largo de todo el globo. Si junto a esta expansión unimos la falta de seguridad en la mayoría de estos dispositivos podemos conseguir que estos sean un receptáculo perfecto para software malicioso, cuyo principal fin es obligar al dispositivo a pertenecer a una red mucho mayor con otros dispositivos infectados que, entre muchas de las posibles opciones, destacan la realización de ataques de denegación de servicio (DDoS) y el minado de criptomonedas de manera remota.

En este proyecto se abordará el proceso de creación de un servidor central y una aplicación web para el control y la monitorización de manera remota de los diferentes dispositivos Windows infectados que componen los *bots* o zombis de nuestra *botnet*, siendo estos infectados por un troyano que también se ha desarrollado.

1.2. Motivación

En lo referente a las *botnets*, documentar, estudiar y desarrollar nuevas técnicas de comunicación por red. El comprender como funcionan, las técnicas que usan actualmente y sus objetivos pueden ser mi principal motivación a la hora de realizar este proyecto.

1.3. Objetivos del proyecto

1.3.1. Objetivos de desarrollo

El principal objetivo del desarrollo de este proyecto es la creación de una *botnet* con fines educativos. Para ello se utilizará un entorno de ataque seguro utilizando una red de ordenadores previamente preparados. En dicho entorno la máquina objetivo es una máquina Windows con la última versión del sistema operativo instalado mientras que la máquina que contiene el servidor de la *botnet* tendrá un sistema operativo Kali Linux.

Para conseguir este objetivo se creará un servidor central desde donde se controlarán los *bots* y desde donde obtendrán las diferentes tareas a realizar. Para poder controlar el servidor central se creará una aplicación web que también se desplegará en la máquina del servidor. Entre las opciones planteadas se encuentra la posibilidad de ejecutar comandos de manera remota, subir ficheros desde el servidor al ordenador infectado y la capacidad de bajar ficheros de este al servidor. Todo ello se hará mediante una conexión remota por canales gRPC seguros bajo encriptación y basados en HTTP/2 como protocolo de comunicación.

1.3.2. Objetivos personales

Como el proyecto a desarrollar es completamente educativo y está desarrollado como Trabajo de Fin de Grado, también presenta unos puntos de objetivos personales:

- Conocer en profundidad el protocolo IP.
- Aumentar mi habilidad con Python como lenguaje de programación principal.
- Desarrollar mi conocimiento en tecnologías actuales de llamadas a procedimientos remotos, empleando gRPC y Protocol Buffers, pues ambos me parecen tecnologías muy potentes con una gran flexibilidad para el paso de mensajes y que, en un futuro, me gustaría seguir aprendiendo y desarrollando en base a estas tecnologías.
- Interesarme, aún más (si cabe), por el mundo de la ciberseguridad, la proliferación de nuevos retos y la creación de nuevas herramientas tanto de ataque, descubrimiento o de control.
- Comprender y explorar las posibilidades del sistema operativo Windows como base actual del proyecto a desarrollar.
- Entender y aprender el marco de trabajo de Scrum, cuyo uso hoy en día es muy habitual en múltiples procesos y entornos laborales de desarrollo software.

1.4. Estructura de la memoria

El resto del documento está estructurado de la siguiente forma:

- **Capítulo 2**: trata sobre el estado del arte, una visión general de las *botnets* por la historia y herramientas importantes utilizadas para el desarrollo de la tarea.
- **Capítulo 3**: trata sobre la metodología de trabajo, así como de las tareas desarrolladas a lo largo del proyecto.
- **Capítulo 4**: trata sobre las tecnologías utilizadas durante todo el proyecto.
- **Capítulo 5**: trata sobre los riesgos y el presupuesto del propio proyecto.
- **Capítulos 6, 7 y 8**: tratan, respectivamente, del diseño, implementación y las pruebas.
- **Capítulo 9**: trata sobre el despliegue de la aplicación y los pasos a seguir.
- **Capítulo 10**: trata sobre las conclusiones finales y las futuras mejoras que puede tener el proyecto.
- **Apéndices**: presentamos dos apéndices, el primero hace referencia al manual de usuario de la aplicación web desarrollada y el segundo es el fichero .proto utilizado durante este proyecto para la serialización de los mensajes entre el servidor y, la aplicación web o el *bot*.

Capítulo 2

Estado del arte

2.1. *Botnet*

2.1.1. ¿Qué es una *botnet*?

Una *botnet* [10] es un conjunto de dispositivos conectados a Internet que han sido infectados por un programa malicioso y que reciben órdenes de manera remota.

Los dispositivos infectados son llamados *bots* o zombis. La principal característica es que cualquier equipo con conexión a Internet puede ser infectado y pasar a formar parte de una *botnet*.

Al administrador de la red de dispositivos infectados se le llama *botmaster*. El servidor o entorno desde donde se gestiona la red se llama C&C (*Command and Control*).

2.1.2. Historia

Los *bots* fueron desarrollados en un inicio como herramientas de gestión para canales IRC [38], proporcionando servicios de manera automática al usuario y evitaban el cierre del canal debido a la inactividad, entre muchas de sus tareas.

En marzo de 1999 fue descubierto el *bot* malicioso PrettyPark [17]. Era un gusano cuyas funciones son comunes a los *bots* de hoy en día. Alguna de sus características son: la posibilidad de obtener el nombre, sistema operativo e información básica del dispositivo, la habilidad de obtener usuarios y contraseñas de los ajustes de red, redirección de tráfico o incluso ataques de denegación de servicio. Los comandos maliciosos eran transmitidos a través de un canal, siendo así una de las primeras *botnets*.

También en 1999 se descubrió otro *bot* malicioso, SubSeven [18]. En este caso se trata de un troyano cuya principal característica es el control como administrador del sistema infectado. Al igual que en el caso anterior, su comunicación se realiza a través de un canal IRC.

Ambos ejemplos mencionados anteriormente utilizaban canales IRC para el envío de información, así como para recibir los comandos maliciosos. Este tipo de *botnets* estaban basadas en un servidor centralizado desde donde se controlaban las máquinas infectadas. Al principio de los años 2000 las *botnets* evolucionaron en múltiples aspectos y, entre los más destacados, fue la implementación de la comunicación P2P [20] como alternativa al único servidor centralizado.

La primera *botnet* en utilizar P2P en la comunicación entre los *bots* fue Zeus o Zbot [19]. Fue diseñado para el robo de credenciales y de datos bancarios pero su posibilidad a la hora de ser personalizable permitía recopilar cualquier tipo de información.

A lo largo de los siguientes años se fue popularizando el uso del P2P frente al uso del servidor centralizado. Estas nuevas *botnets* desarrollaron nuevas técnicas de ocultación haciendo más difícil el descubrimiento de la *botnet*. También fueron adquiriendo un mayor número de zombis, permitiendo que ataques como, por ejemplo, de denegación de servicio, siendo este distribuido y llamándose DDoS (se lleva a cabo generando un gran flujo de información desde varios puntos de conexión hacia un mismo punto de destino).

En los últimos años las *botnets* han desarrollado técnicas para imitar el comportamiento humano para poder estafar dinero a través de la publicidad, como puede ser el caso de Methbot [9], o técnicas para el minado de criptomonedas, como es el caso de Smominru [33], que minaba criptomonedas Monero. Se estima que ambas redes generaban millones de dólares a la semana.

Con el auge del internet de las cosas o IoT [35] la cantidad de dispositivos conectados a Internet crece cada año. Muchos de estos dispositivos no presentan una seguridad suficiente, contienen una configuración básica por defecto o no se pueden actualizar, convirtiendo a este tipo de dispositivos en un objetivo para este tipo de redes. El ejemplo más conocido de los últimos años en referencia a este tipo de *botnet* es Mirai [47], cuyo ataque de DDoS en 2016 dejó a los servidores DNS más importantes fuera de servicio impidiendo acceder a los sitios con mayor relevancia.

La última *botnet* conocida y desmantelada ha sido Emotet [7]. Emotet es un *malware* polimórfico que estaba infectando a miles de ordenadores al día a través de correos electrónicos y archivos infectados. La Europol describió esta *botnet* como "una de las redes de *bots* más importantes de la última década". Su principal función era la descarga y ejecución de código malicioso o programas fraudulentos. Fue desmantelada a principios de 2021.

2.1.3. Infección del dispositivo

Los dispositivos desde donde se ejecutan las tareas maliciosas pueden llegar a infectarse a través de diferentes métodos.

Troyanos

Uno de los métodos más utilizados para la infección de dispositivos. El *botmaster* utiliza múltiples técnicas para conseguir que la víctima ejecute en su dispositivo un archivo malicioso infectado. Estos archivos muchas veces son ficheros adjuntos en un correo o en *pendrives*.

Configuraciones no seguras

Método centrado en dispositivos IoT cuya configuración por defecto es, muchas veces, poco segura junto al uso de contraseñas débiles permitiendo a los atacantes acceder al dispositivo, infectarlo y convertirlo en parte de la *botnet*.

Vulnerabilidades no parcheadas

Se centra en utilizar aquellas vulnerabilidades conocidas y no parcheadas (en su mayoría vulnerabilidades de día cero) para adquirir el control del dispositivo y poder instalar software malicioso u otro tipo de programas para conseguir convertir el dispositivo en una parte de la *botnet*.

2.1.4. Tipos de *botnets*

Centralizada (Cliente-Servidor)

Las primeras redes que aparecieron describían una arquitectura centralizada. Este tipo de *botnets* se caracterizan por presentar un servidor central al cual se conecta el *botmaster* y este se comunica con los *bots* y envían la información pertinente al servidor. Un esquema simple sobre la comunicación sería el mostrado en la figura 2.1.

La utilización de este tipo de redes presentan un único punto de fallo, haciendo referencia al servidor C&C. Si el servidor cae, la *botnet* deja de estar operativa.

El canal de comunicación entre el servidor y los dispositivos, así como con el *botmaster*, puede ser implementado utilizando múltiples protocolos. Los más comunes son IRC, HTTP y SMTP.

Descentralizada (P2P)

Las otras redes que aparecieron son las llamadas *botnets* descentralizadas. Este tipo solventa el problema principal de las redes centralizadas, el único punto de fallo, su servidor. Al utilizar la tecnología P2P cada *bot* se comunica con un conjunto de los mismos presentes en una lista, enviando los comandos a los demás vecinos de la lista. Los *botmasters* usan esta distribución para que su *botnet* adquiera una mayor durabilidad, escalabilidad y resistencia.

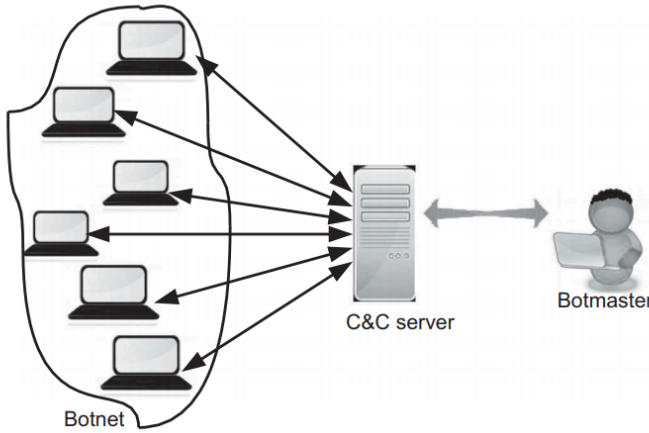

Figura 2.1: *Botnet* centralizada [10]

En este tipo de *botnets* se pueden representar como un gráfico en el que los *bots* son los vértices y los vínculos entre ellos son los bordes. El administrador de la red envía la tarea a uno o varios *bots* de la red esperando a su propagación total por todos los dispositivos. Esta propagación es mucho más lenta que presentando un servidor central. Los protocolos más utilizados son IRC y HTTP. Un esquema simple sobre la comunicación sería el mostrado en la figura 2.2.

Sin embargo, en la mayoría de las redes de *bots* P2P, la arquitectura no es completamente P2P, ya que incluye un servidor central para arrancar y obtener listas de pares iniciales.

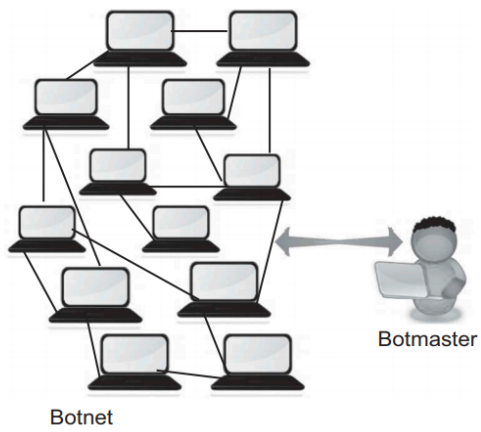

Figura 2.2: *Botnet* descentralizada [10]

2.1.5. Objetivos

Ataques de denegación de servicio distribuido (DDoS)

El principal objetivo de las *botnets*. Consiste en impedir el correcto funcionamiento de un servicio en línea. Se consigue tras obligar a miles de dispositivos zombis acceder al mismo recurso a la vez, saturando dicho servicio e impidiendo a los usuarios reales poder acceder a este.

Spam

El *botmaster* utiliza la red de dispositivos para enviar correos fraudulentos de manera masiva. Muchas veces este tipo de correos también están infectados o contienen otro tipo de código malicioso.

Fraude publicitario

Este fraude se forja haciendo que los *bots* visiten los sitios web propios e interactúen con los anuncios presentes en dichas páginas para poder generar beneficio de una manera fraudulenta.

Venta de la *botnet*

Algunas veces la *botnet* es alquilada en parte o en su totalidad por otros usuarios con el fin de utilizar los dispositivos para sus propios fines (la mayoría son minado de criptomonedas y ataques DDoS). También pueden ser vendidas en partes o fragmentos.

Minado de criptomonedas

Es el segundo objetivo más común. La *botnet* utiliza la capacidad de cómputo de todos los dispositivos infectados para minar criptomonedas, aumentando los beneficios al no gastar recursos en el minado.

Robo de información

El *bot* se encarga de obtener las credenciales de acceso y autenticación de varios sitios, ficheros, e incluso la información bancaria, para luego vender la información recopilada por todos los *bots* de la red al mejor postor.

2.1.6. El impacto de las *botnets*

El impacto de las *botnets* es conocido a lo largo del mundo. Principalmente es un impacto económico, ya sea en beneficio de los *botmaster* o en contra de las empresas afectadas.

Las redes de dispositivos zombis son conocidas en su mayoría por la realización de ataques DDoS a diferentes objetivos a lo largo de los últimos años. Este tipo de ataques, dependiendo del alcance y de las empresas afectadas, puede acarrear una cantidad de perdidas millonarias por la inutilización de sus servicios durante un periodo corto de tiempo. El ejemplo más cercano fue la *botnet* Miari, mencionada con anterioridad, que dejó a los principales servicios de DNS colapsados [52].

El beneficio económico de los creadores no solo reside en realizar ataques DDoS bajo demanda. Dichas redes son una puerta de entrada a otro software malicioso en los dispositivos zombis, pudiendo instalar programas de tipo *ramsonware*, y secuestrar los dispositivos a cambio de una gran cantidad de dinero. Uno de los últimos casos lo vivió Telefónica España en 2017 donde sus sistemas fueron secuestrados por un *ramsonware* a cambio de una gran cantidad de dinero [36].

2.1.7. El futuro de las *botnets*

La tecnología IoT es una tendencia al alza, con millones de nuevos dispositivos al año. Todos estos dispositivos avanzan en materia de seguridad después de verse involucrados en estas redes criminales de dispositivos interconectados, pero las *botnets* también avanzan. Las nuevas *botnets* automatizan procesos y brindan de cierta inteligencia al *bot* para que pueda hacer frente a diferentes situaciones sin intervención de un *botmaster*.

La tecnología *blockchain* [37] se está utilizando por parte de los *botmasters* en los últimos años para ocultar y distribuir diferente información. Un ejemplo puede ser una *botnet* de minado de criptomonedas que utilizó las transacciones de la cadena de bloques de la red Bitcoin para ocultar la dirección IP de su servidor C&C de respaldo [44].

Por último mencionar que tanto la publicidad online como las criptomonedas han crecido en los últimos años, haciendo que la mayoría de redes de dispositivos estén orientadas a la explotación de este tipo de recursos.

2.2. *Cyber Kill Chain*

El ciclo de vida de un ciberataque es llamado *Cyber Kill Chain* [8]. Este concepto fue utilizado por los militares para definir los pasos que usaba el enemigo a la hora de atacar un objetivo. Posteriormente fue utilizado por diferentes analistas como modelo para ayudar en la toma de decisiones a la hora de responder ante ciberataques.

A continuación se identifican los pasos de un ciberdelincuente para alcanzar su objetivo (Figura 2.3).

Figura 2.3: *Cyber Kill Chain*

- **Reconocimiento**: durante esta etapa se investiga y recopila la información sobre el objetivo. Con esta información, el atacante valora qué métodos de ataque podrían funcionar y con qué probabilidad de éxito.

- **Preparación**: durante esta etapa se prepara el ataque de de forma específica sobre un objetivo. Por ejemplo, desarrollar de un fichero PDF infectado o incluso redactar correos electrónicos fraudulentos.

- **Distribución**: durante esta etapa se produce la transmisión del ataque, por ejemplo, al abrir el fichero infectado mencionado en el punto anterior.

- **Explotación**: durante esta etapa se aprovecha de una vulnerabilidad y la explota con el fin de comprometer al equipo y la red del objetivo.

- **Instalación**: durante esta etapa se instala el *malware* en la víctima y establece persistencia en el equipo.

- **Comando y control**: en esta etapa el atacante cuenta con el control del sistema de la víctima, en el que podrá realizar o desde el que lanzar sus acciones maliciosas dirigidas desde un servidor central conocido como C&C (*Command and Control*).

- **Acciones sobre los objetivos**: última etapa donde el atacante realiza las diferentes acciones, como el robo de credenciales o la instalación de otro *malware*.

2.2.1. *Cyber Kill Chain* y las *botnets*

A continuación se expone el modelo de funcionamiento de una *botnet* (Figura 2.4) utilizando el ciclo de vida de un ciberataque descrito anteriormente.

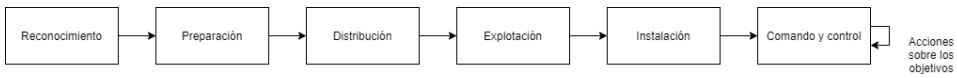

Figura 2.4: Pasos que realiza una *botnet*

La primera etapa se reconoce los dispositivos que van a ser infectados para pertenecer a la *botnet*. Durante esta etapa se identifican sus características, así como sus puntos débiles y la mejor manera de atacarles.

Tras la anterior etapa, se prepara el ataque. Hay varias maneras de infectar un dispositivo, la más común es a través de un troyano. Se puede ayudar de un correo electrónico malicioso o ficheros adjuntos maliciosos. La etapa de distribución está ligada de manera directa con esta etapa.

Una vez el *malware* está en el dispositivo objetivo, se detona el ataque, comprometiéndolo.

Si se compromete el dispositivo de manera correcta, el *malware* o troyano se instala en el dispositivo de la víctima, inicia comunicación con el servidor central y crea persistencia en el equipo.

La última etapa real es la referente al comando y control. En esta etapa el dispositivo pertenece a la *botnet*, se comunica con el servidor central y, desde este, se le envían tareas que realizar.

Los objetivos, una vez cumplidos, son reenviados al servidor central y desde este se le vuelven a asignar nuevas tareas. Es por eso que en la figura se describe un bucle sobre esta etapa, el zombi está preguntando y realizando las tareas que se le vayan asignando.

2.3. MITRE ATT&CK

MITRE [13] es una corporación no gubernamental. Provee ingeniería de sistemas, investigación y desarrollo, y soporte sobre tecnologías de la información al gobierno de Estados Unidos.

Posee un *framework* llamado MITRE ATT&CK (Tácticas, Técnicas y Conocimiento Común de Adversarios) [14], una plataforma que organiza y categoriza los distintos tipos de ataques, amenazas y procedimientos realizados por los distintos atacantes en el mundo digital y que permite identificar vulnerabilidades en los sistemas informáticos.

El entorno presenta dos matrices principales:

- **Matriz empresarial (*Enterprise*)**: la matriz empresarial contiene información para las siguientes plataformas: Windows, macOS, Linux, PRE, Azure AD, Office 365, Google Workspace, SaaS, IaaS, red y contenedores.
- **Matriz móvil (*Mobile*)**: la matriz móvil contiene información para las siguientes plataformas: iOS y Android.

Esta base de conocimiento puede ser útil para crear un mapa del sistema de defensa de una compañía o dispositivo ya que se describen principalmente los compartimientos de los atacantes. También puede ser usado a la inversa, y aprender y comprender el funcionamiento de diversas técnicas de reconocimiento, ataque y explotación.

Existe una herramienta *online* [15] para diseñar y categorizar las pruebas realizadas sobre diferentes sistemas y llevar un control de los resultados que se fueron obteniendo.

Capítulo 3

Planificación y desarrollo del proyecto

3.1. Scrum

3.1.1. Introducción

Scrum [45] es un marco de trabajo para el desarrollo ágil de *software*. En este proceso se aplican de manera regular un conjunto de buenas prácticas para poder sacar adelante el proyecto trabajando en equipo lo mejor posible. Scrum es utilizado en entornos complejos, donde se necesitan resultados rápidos y los requisitos son cambiantes. La principal idea es reducir la complejidad del producto ayudado de la fuerza del trabajo en equipo.

Scrum se basa en tres pilares:

- **Transparencia.** Los aspectos importantes del proceso deben ser visibles y entendibles por todos aquellos relacionados con el proyecto. La transparencia obliga a definir estos elementos siguiendo un estándar común.

- **Inspección.** El progreso debe ser inspeccionado de manera constante para detectar variaciones indeseadas de cara al resultado final. Este trabajo no debe ser muy repetitivo para evitar interferencias con el trabajo.

- **Adaptación.** Si se detectan variaciones indeseadas que desvían de límites aceptables a un determinado proceso, este debe ser reajustado en el menor tiempo posible para evitar desviaciones mayores.

3.1.2. Equipo Scrum

El *Scrum Team* o Equipo Scrum se compone de tres roles: el *Product Owner*, el *Scrum Master* y el *Development Team*.

- ***Product Owner.*** Representa al cliente y determina la visión del producto. Es el máximo responsable del valor del producto final y del trabajo del equipo de desarrollo ya que su labor principal es gestionar las prioridades a la hora de realizar las diferentes tareas.
- ***Scrum Master.*** Es el facilitador, su deber es potenciar la productividad del equipo y asegurarse que todos utilicen Scrum de manera correcta.
- ***Development Team.*** Son los profesionales encargados de realizar el trabajo a entregar. Es un equipo autoorganizado.

La principal característica de estos equipos es su autoorganización, evitando que las competencias para poder desarrollar el trabajo dependan de personas ajenas al equipo.

3.1.3. Eventos

Scrum presenta eventos predefinidos para minimizar la necesidad de realizar reuniones fuera del plan y crear así una cierta regularidad. Los eventos son bloques de tiempo fijo que se dan a lo largo de las diferentes etapas del proyecto.

- ***Sprint.*** Es el bloque principal que compone el desarrollo del proyecto. A lo largo del desarrollo se realizarán varios *sprints* cuyo contenido asignado es invariable. Presenta una duración máxima entre dos y cuatro semanas. Durante todo el *sprint* el *Scrum Master* se encarga de supervisar el equipo para ayudar a que se cumpla el objetivo marcado.
- **Reunión de planificación (*Sprint Planning*).** Antes de comenzar un *sprint* se debe realizar una reunión de planificación. En esta reunión el cliente presenta la lista de requisitos (*Product Backlog*) la cual el *Product Owner* ordena por nivel de prioridad. El *Development Team*, a partir de los requisitos con mayor prioridad, genera una lista de tareas o *Sprint Backlog*. Estos requisitos seleccionados son los que el equipo debe desarrollar en el *sprint*.
- **Objetivo del *sprint* (*Sprint Goal*).** Meta establecida en el *Sprint Planning* para ayudar al *Development Team* a entender el porqué del *sprint* a realizar. Es un punto de unión del grupo con el proyecto.
- **Reunión diaria (*Daily Scrum*).** Reunión corta que realiza el equipo de desarrollo para sincronizar sus actividades. Se produce a la misma hora, en el mismo lugar todos los días para reducir la complejidad. Se tiene en cuenta lo trabajado desde el anterior

Daily Scrum para la proyección del siguiente y así evaluar el progreso hacia el objetivo del *sprint*. Este tipo de reuniones mejoran la comunicación, eliminan la necesidad de realizar otras reuniones, identifican impedimentos durante el desarrollo y promueven la toma de decisiones rápida.

- **Revisión del *sprint* (Sprint Review).** Al final del *sprint* se hace una reunión de revisión donde se entregan los requisitos completados al cliente. Es posible que se adapte el proyecto y la lista de requisitos para enfocarse en nuevos objetivos.
- **Retrospectiva del *sprint* (*Sprint Retrospective*).** Es una reunión que se realiza previa al siguiente *sprint* con el *Scrum Master* para tratar diferentes aspectos como el esfuerzo y su manera de trabajar, pudiendo abordar posibles mejoras o soluciones a la hora de optimizar el trabajo en equipo en el futuro.

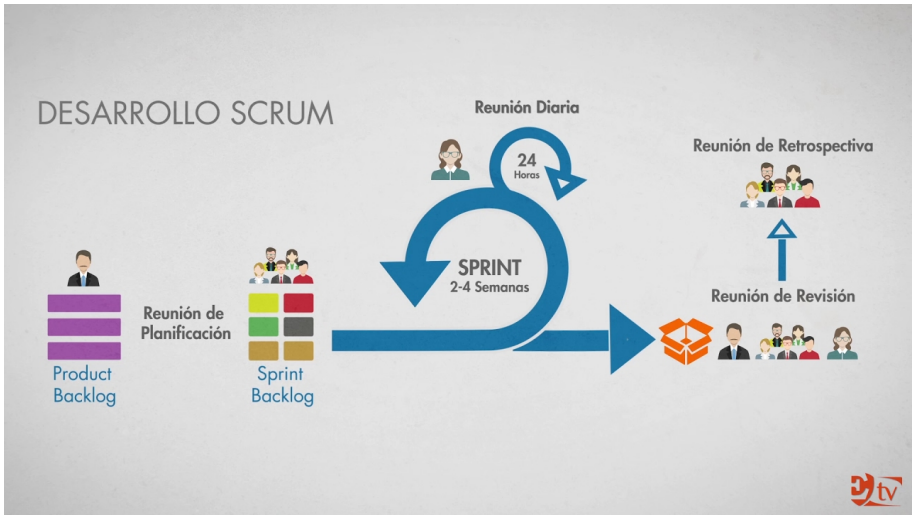

Figura 3.1: Desarrollo Scrum [16]

3.2. Planificación

Para este proyecto he adaptado el marco de trabajo de Scrum a un trabajo individual sin perder la esencia de los tres roles diferenciados mencionados con anterioridad.
La mayor parte del tiempo he representado el rol de *Development Team* y solo en las ocasiones necesarias (al principio y al final del *Sprint*) he asumido el rol del *Product Owner*. El rol del *Scrum Master* ha estado siempre presente al llevar una rutina de trabajo diario durante los diferentes *sprints*.

La duración de los *sprints* son de dos semanas. El día asignado para las reuniones de revisión y retrospectiva del *sprint* será el lunes, coincidiendo así tras el paso de las 2 semanas asignadas.

Para la lista de requisitos (*Product Backlog*), al igual que para la lista de tareas (*Sprint Backlog*), he utilizado Trello, aplicación que me permite controlar en todo momento el desarrollo y estado de cada tarea a lo largo de los *Sprint*.

Al inicio del proyecto se realizará un *sprint* 0 para preparar la documentación y la planificación inicial. Esto favorecerá el trabajo futuro dejando los demás *sprints* para el desarrollo, implementación y despliegue del proyecto, complementando esta memoria cada poco tiempo. Este primer *Sprint* comenzará el 1 de marzo de 2021.

El Trabajo Fin de Grado del grado de Ingeniería Informática de la Universidad de Valladolid [32] corresponde a un total de 12 créditos, lo equivalente unas 300 horas. En total, dada la disponibilidad, se utilizarán unas 35 horas por iteración, estimando que el proyecto será desarrollando en 8 *sprint* (contando con el *sprint* 0).

Durante el desarrollo se realizó un cambio de tecnología que sucedió al final del *sprint* 1, haciendo que el planteamiento inicial, al igual que el diseño implementado, tuviera que ser modificado. Este cambio hizo que el proyecto se alargase un *sprint* más en el mes de Julio, provocando más carga de trabajo en el resto de *sprint* para compensar las horas perdidas.

A continuación se muestra en detalle la planificación final del proyecto:

Sprint	Fecha inicio	Fecha fin	Notas
Sprint 0	01/03/2021	15/03/2021	Planificación inicial
Sprint 1	15/03/2021	29/03/2021	
Sprint 2	29/03/2021	12/04/2021	Cambio de tecnología
Sprint 3	12/04/2021	26/04/2021	
Sprint 4	26/04/2021	10/05/2021	
Sprint 5	10/05/2021	24/05/2021	
Sprint 6	24/05/2021	07/06/2021	
Sprint 7	07/06/2021	21/06/2021	
Sprint 8	21/06/2021	05/07/2021	Periodo extraordinario

Tabla 3.1: Planificación de los *sprints*

3.3. Tareas realizadas

En este apartado se mostrará una visión de las historias de usuario de este proyecto que están representadas en las tablas 3.2 y 3.3. A mayores se explicará el desarrollo de los *sprints* de manera concisa, mostrando las tareas de una manera más detallada.

Número	H.U.	Descripción
1	Mostrar los *bots* registrados	El usuario consulta los *bots* que hay registrados en la base de datos junto a la información de la máquina infectada.
2	Mostrar la información de un *bot*	El usuario consulta la información del *bot* almacenada en la base de datos (información de la máquina infectada, geolocalización, tareas pendientes, tareas completadas).
3	Mostrar tareas pendientes	El usuario consulta las tareas pendientes totales que faltan por hacer, incluyendo su estado, el tipo de tarea, la hora a la que fue creada y a que *bot* va dirigida dicha tarea.
4	Mostrar tareas completadas	El usuario consulta las tareas completadas totales, incluyendo el estado final, el tipo de tarea, la hora a la que fue creada, la hora a la que fue completada y que *bot* ha realizado dicha tarea.
5	Mostrar la información de una tarea	El usuario consulta la información de la tarea y comprueba si la tarea se ha completado. Si es así, puede ver la respueta del *bot*.
6	Añadir una tarea	El usuario añade una tarea determinada a uno o varios *bots* a su lista de tareas pendientes.
7	Mostrar los *logs* de los *bots*	El usuario consulta la lista de *logs* de los *bots* que se han conectado con el servidor, así como su posibilidad para descargar el fichero.
8	Mostrar el *log* del servidor	El usuario consulta el *log* del servidor con su posibilidad de descargar el fichero.
9	Mostrar los elementos descargados de las máquinas infectados	El usuario consulta los archivos descargados por las diferentes tareas que han realizado los *bots* con la posibilidad de descargar el fichero.
10	Mostrar los *bots* registrados en un mapa	El usuario puede consultar la ubicación geográfica de los *bots* registrados en un mapa global.
11	Registrar un *bot* en el servidor	El *bot* se conecta automáticamente con el servidor y se registra su información principal.

Tabla 3.2: Historias de usuario

Número	Tarea	Descripción
12	Obtener las tareas pendientes	El *bot* pregunta al servidor cada cierto periodo de tiempo si tiene tareas pendientes y, si es así, las recibe desde el servidor.
13	Realizar una tarea	El *bot* realiza automáticamente una tarea pendiente.
14	Enviar una tarea realizada	El *bot* envía al servidor una tarea completada.
15	Eliminar un *bot*	El usuario puede eliminar uno o varios *bots* de manera remota.

Tabla 3.3: Continuación de historias de usuario

Las tareas en detalle estarán representadas por un identificador con el número de la tarea, el tipo de tarea realizada, su descripción para ayudar a entender en qué consiste el trabajo, cuantas horas me ha llevado su realización y si se ha podido completar en el tiempo asignado.

Los tres tipos de tareas presentes a lo largo del proyecto son:

- **DOC.** Tarea centrada en la realización de la documentación, estudio y desarrollo de la parte teórica del proyecto.
- **DEV.** Tarea centrada en el desarrollo del producto.
- **TEST.** Tarea centrada en la realización de pruebas sobre la robustez y la funcionalidad.

3.3.1. *Sprint* 0

Durante este *sprint* se aborda y se trabaja la mayoría de los aspectos generales sobre el proyecto (formación y herramientas necesarias para el proyecto) así como la mayor parte del contenido teórico de la memoria. Sin embargo la tarea 2 no fue completada por una mala estimación del peso de la tarea, dejando incompleta esta tarea para el siguiente *sprint*.

En la tabla 3.4 se muestran las tareas planificadas. En total se emplearon 32 horas para la realización del *sprint* 0.

Tarea	Tipo	Descripción	Horas	Estado
T001	DOC	Redacción de los apartados de introducción y objetivos	4	Finalizada
T002	DOC	Redacción del apartado del estado del arte	5	No finalizada
T003	DOC	Redacción del apartado de análisis del proyecto	4	Finalizada
T004	DEV	Creación y configuración del entrono de trabajo virtual Python	1	Finalizada
T005	DOC	Formación sobre *sockets* en Python	6	Finalizada
T006	DOC	Formación sobre *backdoors* en Python	12	Finalizada
			32	

Tabla 3.4: Tareas Sprint 0

3.3.2. *Sprint* 1

Esta iteración comienza el mismo día que acaba el anterior *sprint*. Lo primero que se hace es finalizar la tarea sobre la redacción del estado del arte en la memoria. Una vez finalizada se continúa con el diseño del proyecto y la descripción de las tareas que realizarán los *bots*.

Una vez documentado toda la base teórica del proyecto se procede a iniciar la creación del proyecto en GitLab, seguido de la creación del servidor y el cliente (o *bot*). El servidor consta de una interfaz por terminal que permite gestionar diferentes aspectos recogidos en la tabla de tareas planteadas como la consulta de *bots* registrados o la posibilidad de añadir una tarea. El cliente es un archivo que se mantiene en bucle que cumple la principal función de preguntar si existen tareas para él y, en el caso de ser así, obtenerlas.

Se llegan a realizar todas las tareas a excepción de la creación del primer módulo para la ejecución de comandos de manera remota por parte del *bot*. En este caso la tarea quedará a medias y deberá ser completada en el siguiente *sprint*.

Por último, y tras una reunión con el tutor, se revisó el diseño principal del proyecto así como la tecnología utilizada para el paso de mensajes entre el cliente y el servidor. Al tutor no le gustó la idea de utilizar una tecnología tan rudimentaria para un trabajo donde la conectividad es un pilar fundamental. En este caso el tutor me recomendó utilizar la tecnología gRPC y Protocol Buffers frente a la tecnología de *sockets*. Por este motivo, el trabajo realizado hasta el momento quedó inservible (a excepción de la tarea a medio realizar) teniendo que replantear para el siguiente *sprint* todo el diseño y buscando nueva documentación y formación acerca de esta nueva tecnología.

En la tabla 3.5 se muestran las tareas planificadas. En total se emplearon 39.5 horas para la realización del *Sprint* 1.

Tarea	Tipo	Descripción	Horas	Estado
T007	DOC	Redacción del apartado del estado del arte	3	Finalizada
T008	DOC	Diseño del proyecto	6	Finalizada
T009	DOC	Creación del proyecto en GitLab	0.5	Finalizada
T010	DEV	Desarrollo del servidor en Python	4	Finalizada
T011	DEV	Desarrollo de la interfaz del servidor	4	Finalizada
T012	TEST	Pruebas de la interfaz del servidor	3	Finalizada
T013	DEV	Desarrollo del cliente en Python	8	Finalizada
T014	TEST	Pruebas de conexión entre el servidor y el cliente en el entorno de laboratorio	2	Finalizada
T015	DEV	Desarrollo del módulo de la tarea para ejecución de código de manera remota	8	No finalizada
			39.5	

Tabla 3.5: Tareas *sprint* 1

3.3.3. *Sprint* 2

En esta iteración se aborda de nuevo los aspectos tratados en la iteración 0, rehaciendo parte del trabajo realizado en la memoria a lo largo de las semanas pasadas. Consta, en su mayoría, de tareas de formación sobre las nuevas tecnologías mencionadas por el tutor el día de la reunión. Para completar la formación se realizan varios servidores y clientes de ejemplo con el fin de potenciar el conocimiento del *framework* gRPC.

En la tabla 3.6 se muestran las tareas planificadas. En total se emplearon 44 horas para la realización del *Sprint* 2.

Tarea	Tipo	Descripción	Horas	Estado
T016	DOC	Formación sobre gRPC	14	Finalizada
T017	DOC	Formación sobre Protocol Buffers	14	Finalizada
T018	DEV	Desarrollo de un servidor gRPC	3	Finalizada
T019	DEV	Desarrolo de un cliente gRPC	3	Finalizada
T020	TEST	Pruebas en el entorno con el servidor y el cliente gRPC	2	Finalizada
T021	DOC	Redacción de los apartados afectados por el cambio de tecnología	8	Finalizada
			44	

Tabla 3.6: Tareas *sprint* 2

3.3.4. *Sprint* 3

En esta iteración se comienza el desarrollo del proyecto de nuevo. Para evitar conflictos o reutilización de código innecesario se crea un nuevo proyecto en GitLab. Con el nuevo proyecto creado en GitLab, se diseña de nuevo todo en base al diseño nuevo teniendo un servidor central y uno o varios clientes que se conectan a dicho servidor. De momento, para evitar abarcar demasiado, se decide seguir utilizando la interfaz rudimentaria del servidor.

Se consigue terminar de crear el módulo por parte del cliente para las tareas de ejecución de código remoto, así como la de toma de captura de pantalla. Ambos módulos son probados y compilados junto al cliente para comprobar, dentro del entorno de laboratorio, que los ejecutables de Windows generados son funcionales. A mayores, por parte del servidor, se crean los módulos para la gestión de ficheros y la creación de *logs*. Por último, se deja iniciado el módulo para el envío y el recibimiento de ficheros entre ambas partes.

En la tabla 3.7 se muestran las tareas planificadas. En total se emplearon 43.5 horas para la realización del *sprint* 3.

Tarea	Tipo	Descripción	Horas	Estado
T022	DOC	Creación de un proyecto en GitLab	0.5	Finalizada
T023	DOC	Diseño del proyecto	6	Finalizada
T024	DEV	Desarrollo del servidor gRPC	2	Finalizada
T025	DEV	Desarrollo del *bot*	2	Finalizada
T026	DEV	Desarrollo del módulo del *bot* de ejecución de código remoto	3	Finalizada
T027	DEV	Desarrollo del módulo del *bot* de captura de pantalla remota	5	Finalizada
T028	DEV	Desarrollo del módulo del servidor del sistema de *logs*	5	Finalizada
T029	DEV	Desarrollo del módulo del *bot* de envío de ficheros	3	No finalizada
T030	DEV	Compilación del *bot* para crear ejecutables de Windows	6	Finalizada
T031	TEST	Pruebas de las tareas entre el servidor y el *bot* en el entorno de laboratorio	3	Finalizada
T032	TEST	Pruebas de las tareas entre el servidor y el *bot* compilado en el entorno de laboratorio	5	Finalizada
T033	DOC	Redacción de los apartados afectados por el avance del proyecto	3	Finalizada
			43.5	

Tabla 3.7: Tareas *sprint* 3

3.3.5. *Sprint* 4

La iteración aborda el resto de módulos de las diferentes tareas que debe realizar el *bot* de manera autónoma, por ello se termina el desarrollo de los módulos y, a mayores, se implementan nuevos módulos en el servidor para gestionar mejor la comunicación entre los *bots* y el servidor a través de las funciones gRPC definidas en ambos lados. La funcionalidad más destacada implementada en estas dos semanas es la presencia de canales seguros y encriptados para realizar el paso de mensajes lo más seguro posible.

Esta iteración aborda un amplio periodo de pruebas para comprobar que todos los módulos, sean del cliente o del servidor, funcionan correctamente y que estos se pueden utilizar en las siguientes fases del proyecto.

En la tabla 3.8 se muestran las tareas planificadas. En total se emplearon 41 horas para la realización del *sprint* 4.

Tarea	Tipo	Descripción	Horas	Estado
T034	DEV	Desarrollo del módulo del servidor de descarga de ficheros de manera remota	5	Finalizada
T035	DEV	Desarrollo del módulo del *bot* de subida de ficheros al servidor	5	Finalizada
T036	DEV	Desarrollo del módulo del servidor de subida de ficheros a la máquina remota	5	Finalizada
T037	DEV	Desarrollo del módulo del *bot* de descarga de ficheros de manera remota	5	Finalizada
T038	DOC	Formación sobre certificados y claves	3	Finalizada
T039	DEV	Implementación de claves y mensajes seguros entre el *bot* y el servidor	7	Finalizada
T040	TEST	Prueba de los mensajes y canales seguros bajo encriptación	3	Finalizada
T041	TEST	Pruebas de los módulos del *bot* y del servidor en el entorno de laboratorio	8	Finalizada
			41	

Tabla 3.8: Tareas *sprint* 4

3.3.6. *Sprint* 5

Esta iteración se centra en la aplicación web destinada al usuario principal para el control de la *botnet*, los *bots* y sus tareas. Esta aplicación sustituirá a la interfaz rudimentaria original presente en el servidor con el fin de facilitar la accesibilidad al usuario final. Este paso se da sabiendo que todas las tareas funcionan correctamente entre el servidor y el *bot*.

En este *sprint* dedicaremos tiempo a aprender y formarse sobre Flask y toda la tecnología que usa (Jinja2, CSS3, HTML5, etc.). Una vez se finaliza el periodo de aprendizaje, se creará

la plantilla pertinente de la página web y se desarrollaran las páginas, al igual que los métodos entre la aplicación y el servidor.

La idea era poder realizar, a parte de la página principal, la página donde salen todos los *bots* registrados por el servidor, la página de información de un *bot* determinado y la página de todas las tareas. Al final, por exceso de carga de trabajo, se quedan incompletas las dos últimas páginas mencionadas.

En la tabla 3.9 se muestran las tareas planificadas. En total se emplearon 39 horas para la realización del *sprint* 5.

Tarea	Tipo	Descripción	Horas	Estado
T042	DOC	Formación en Flask	8	Finalizada
T043	DOC	Formación en Jinja2	8	Finalizada
T044	DOC	Formación en CSS3 y HTML5	4	Finalizada
T045	DEV	Creación de la plantilla HTML5 con CSS3 y Jinja2	3	Finalizada
T046	DEV	Creación y desarrollo de la página principal de la aplicación web	3	Finalizada
T047	DEV	Creación de la página HTML5 para los *bots* registrados en el servidor	1	Finalizada
T048	DEV	Desarrollo del módulo de la aplicación para la petición sobre los *bots* registrados en el servidor	5	Finalizada
T049	DEV	Creación de la página HTML5 para la información de un *bot* determinado	1	No finalizada
T050	DEV	Desarrollo del módulo de la aplicación para la petición de la información de un *bot* determinado	3	No finalizada
T051	DEV	Creación de la página HTML5 para las tareas presentes en el servidor	1	No finalizada
T052	DEV	Desarrollo del módulo de la aplicación para la petición de las tareas presentes en el servidor	3	No finalizada
			39	

Tabla 3.9: Tareas *sprint* 5

3.3.7. *Sprint* 6

En esta iteración se sigue con el desarrollando la aplicación web para la gestión de la *botnet*. En estas dos semanas se seguirá con las páginas de la aplicación al igual que los módulos que gestionan las peticiones (ya sean al servidor gRPC o a la propia aplicación Flask).

En la tabla 3.10 se muestran las tareas planificadas. En total se emplearon 40 horas para la realización del *Sprint* 6.

Tarea	Tipo	Descripción	Horas	Estado
T053	DEV	Creación de la página HTML5 para la información de un *bot* determinado	0.5	Finalizada
T054	DEV	Desarrollo del módulo de la aplicación para la petición de la información de un *bot* determinado	2	Finalizada
T055	DEV	Creación de la página HTML5 para las tareas presentes en el servidor	0.5	Finalizada
T056	DEV	Desarrollo del módulo de la aplicación para la petición de las tareas presentes en el servidor	2	Finalizada
T057	DEV	Creación de la página HTML5 para agregar tareas a uno o varios *bots*	6	Finalizada
T058	DEV	Desarrollo del módulo de la aplicación para enviar las nuevas tareas al servidor	8	Finalizada
T059	DEV	Creación de la página HTML5 para mostrar la información de una tarea determinada	1	Finalizada
T060	DEV	Desarrollo del módulo de la aplicación para la petición de una tarea determinada	4	Finalizada
T061	DEV	Creación de la página HTML5 para mostrar todos los *logs* almacenados en el servidor	1	Finalizada
T062	DEV	Desarrollo del módulo de la aplicación para descargar un *log* determinado	3	Finalizada
T063	DEV	Creación de la página HTML5 para mostrar los archivos descargados en el servidor por los *bots*	1	Finalizada
T064	DEV	Desarrollo del módulo de la aplicación para la petición de descargar un fichero de la página de archivos	5	Finalizada
T065	TEST	Navegación por la aplicación web, así como por las diferentes páginas creadas y completadas	6	Finalizada
			40	

Tabla 3.10: Tareas *sprint* 6

3.3.8. *Sprint* 7

En la penúltima iteración, la penúltima del desarrollo del proyecto, se terminan todas las páginas de la página web. Una vez finalizadas se implementa una base de datos relacional en el servidor para la persistencia de los datos. Para esta labor me formo en la tecnología SQLite con Python y se crea un nuevo módulo del servidor encargado de comunicarse con la base de datos. Esta base de datos no puedo ser terminada dada la gran carga de trabajo que llevaba este *sprint*. Por último, se añaden tareas nuevas que puede realizar el *bot* de la *botnet* como apagarse o eliminarse.

En la tabla 3.11 se muestran las tareas planificadas. En total se emplearon 36 horas para la realización del *sprint* 7.

Tarea	Tipo	Descripción	Horas	Estado
T066	DEV	Creación de la página HTML5 para mostrar los *bots* en un mapa	1	Finalizada
T067	DEV	Desarrollo del módulo de la aplicación para geoposición de los *bots* en el mapa	3	Finalizada
T068	DOC	Formación de SQLite con Python	8	Finalizada
T069	DEV	Desarrollo del módulo del servidor de la base de datos	8	No finalizada
T070	DEV	Desarrollo de los scripts de la base de datos	4	Finalizada
T071	DEV	Desarrollo de la tarea para apagar el *bot* de manera remota	4	Finalizada
T072	DEV	Desarrollo de la tarea para eliminar el *bot* de manera remota	8	Finalizada
			36	

Tabla 3.11: Tareas *sprint* 7

3.3.9. Sprint 8

Última iteración del proyecto. En esta iteración se termina de desarrollar el módulo de la base de datos, así como perfilar varios detalles en la aplicación web. Se realiza una jornada intensiva de pruebas con el módulo de la base de datos en el entorno de laboratorio. Al final, se añade persistencia al *bot* para que, de manera remota hasta que no sea eliminado, se conecte constantemente con la base de datos aún si el equipo se apaga.

Tras el periodo de desarrollo se añaden pequeñas modificaciones al proyecto como, por ejemplo, la creación de *scripts* de arranque o nuevos certificados para comprobar la funcionalidad total.

Por último, se asigna el resto del tiempo presente para la finalización de la memoria.

En la tabla 3.12 se muestran las tareas planificadas. En total se emplearon 44 horas para la realización del *sprint* 8.

Tarea	Tipo	Descripción	Horas	Estado
T073	DEV	Desarrollo del módulo del servidor de la base de datos	8	Finalizada
T074	TEST	Pruebas entre el *bot*, el servidor y la aplicación web	12	Finalizada
T075	DEV	Desarrollo del módulo de persistencia del *bot*	8	Finalizada
T076	DEV	Despliegue del proyecto en el entorno de laboratorio de la escuela	2	Finalizada
T077	TEST	Pruebas entre el *bot*, el servidor y la aplicación web en en el entorno de laboratorio	6	Finalizada
T078	DOC	Finalización de la memoria	8	Finalizada
			44	

Tabla 3.12: Tareas *sprint* 8

Capítulo 4

Tecnologías utilizadas

4.1. Python

Python [43] es un lenguaje de programación interpretado, multiparadigma, orientado a objetos y de alto nivel. Apareció en 1991 y a día de hoy es uno de los lenguajes más utilizados en todo el mundo. Su última versión estable es la 3.9.5.

La elección de Python para realizar el proyecto ha sido su facilidad para realizar scripts al igual que la creación de módulos instalables y de fácil modificación.

Todo el desarrollo de este proyecto se ha realizado bajo la versión estable 3.8.6.

4.1.1. Requests

El paquete Requests [40] permite realizar en Python, de manera simple, peticiones HTTP. Es uno de los paquetes más descargados en la actualidad y su utilidad es muy extendida entre la comunidad. Se utilizará para la interacción entre el *frontend* y el *backend* de la aplicación web de gestión de la *botnet*.

4.1.2. pytz

El paquete pytz [6] contiene la base de datos Olson tz, también llamada tz o zoneinfo [28], que permite los cálculos de zonas horarias de manera precisa. Permite obtener la zona horaria a través de la localización pasada. El principal uso es complementar la asignación de tiempos añadiendo a estos la zona horaria correspondiente.

4.1.3. PyScreeze

El paquete PyScreeze [48] permite tomar capturas de pantalla y posee varios métodos para su almacenamiento. Es necesario el módulo Pillow [11]. Se utilizará para la confección de una de las tareas de los *bots*.

4.1.4. PyInstaller

El paquete PyInstaller [22] permite agrupar una aplicación escrita en Python y sus dependencias en un único ejecutable. Dicho programa resultante puede ser utilizado sin necesidad de tener un intérprete de Python instalado. La aplicación final, referente al *bot* del proyecto, es generado utilizando este paquete.

4.2. gRPC

gRPC [4] es un *framework* de llamada de procedimiento remoto (RPC) de código abierto que puede ejecutarse en cualquier entorno. Fue desarrollado en un inicio por Google. Se utiliza para conectar servicios de manera eficiente con múltiples aplicaciones como el balanceo de carga o la verificación y autenticación.

Para este proyecto se utilizarán llamadas gRPC en Python [5] para transmitir la información entre las diferentes partes del proyecto (cliente web, *bots* y servidor gRPC) a través del protocolo HTTP/2.

4.2.1. Protocol Buffers

Las llamadas gRPC son definidas utilizando los llamados Protocol Buffers [23], un mecanismo de serialización de estructuras de datos. La idea de los Protocol Buffers es definir como se van a estructurar los datos y, una vez compilados, se genera el código fuente que permite escribir y leer fácilmente dichos datos. Su implementación permite a las llamadas gRPC la creación y composición de diferentes tipos de mensajes.

4.3. Flask

Flask [41] es un *framework* minimalista escrito en Python. Se utiliza para la creación de aplicaciones web utilizando Python y Jinja2 [42] para el tratamiento de plantillas.

En este proyecto se utilizará para crear la aplicación web que controlará el servidor gRPC, constituyendo así el servidor C&C de la *botnet*.

4.4. SQLite

SQLite [26] es un sistema de gestión de bases de datos relacional escrito en C cuyo motor SQL destaca en ser pequeño, rápido, autónomo y de gran confiabilidad. Está integrado en la mayoría de dispositivos móviles, ordenadores y aplicaciones. Su formato es multiplataforma y compatible con versiones anteriores. El código fuente es de dominio público.

Para este proyecto se utilizará una base de datos relacional para el almacenamiento y gestión de los diferentes *bots* y sus correspondientes tareas.

4.4.1. DB Browser

DB Browser [39] es una herramienta de código abierto que permite crear, diseñar y editar archivos de bases de datos que son compatibles con SQLite.

Se ha utilizado la aplicación para comprobar y verificar, de manera visual bajo un entorno amigable, la base de datos del proyecto (datos, tablas, etc.).

4.5. Leaflet

Leaflet [2] es una biblioteca JavaScript de código abierto basada en OpenStreetMaps [12] que permite la realización de mapas interactivos cuyo enfoque es la simplicidad, el rendimiento y la facilidad de uso. Al ser de código abierto su implementación es totalmente gratuita.

Se utiliza esta biblioteca para el mapeo unitario o global de los *bots* conectados y registrados por la *botnet* a partir de su dirección.

4.6. Bootstrap

Bootstrap [50] es uno de los *frameworks* de HTML, CSS y JavaScript más populares en la actualidad para la composición de páginas web. Una de sus principales prestaciones es el diseño móvil.

En este proyecto se utilizará para crear la parte *frontend* de la aplicación web de gestión de la *botnet*.

4.6.1. Bootstrap Table

Bootstrap Table [51] es una extensión de las funcionalidades originales del *framework* Bootstrap.

Se ha utilizado para complementar los elementos originales de Bootstrap (principalmente las tablas creadas) para añadir, en este caso, filtros dinámicos del contenido de la tabla.

4.7. Git

Git [24] es una herramienta gratuita de control de versiones de código abierto. Se utiliza para el desarrollo y el mantenimiento del código, así como de las diferentes versiones de las aplicaciones. Se estructura en repositorios, cada uno de los cuales almacena los archivos que constituyen el proyecto.

En este proyecto he utilizado las diferentes herramientas que me ha ofrecido Git para mantener, organizar y actualizar el código fuente del proyecto.

4.7.1. GitLab

Una de las herramientas que ofrece la Escuela de Ingeniería Informática de la Universidad de Valladolid de manera gratuita es un entorno de GitLab [31] para poder gestionar los proyectos creados a lo largo de la carrera. GitLab [29] es un servicio web de control de versiones y desarrollo de software colaborativo basado en Git.

4.8. LaTeX

LaTeX [34] es un sistema de composición de textos, orientado a la creación de documentos escritos que presenten una alta calidad tipográfica. LaTeX es el estándar para la comunicación y publicación de documentos científicos y está disponible como software libre y gratuito.

4.8.1. Overleaf

Overleaf [25] es un editor colaborativo de LaTeX basado en la nube que se utiliza para escribir, editar y publicar documentos científicos. He utilizado Overleaf para poder escribir y editar la memoria del proyecto. Al ser un servicio basado en la nube se puede realizar dicho trabajo desde cualquier lugar con la ventaja de que presenta un compilador integrado.

4.9. Astah

Astah [3] es una herramienta de modelado UML.

Se ha utilizado para desarrollar los diagramas de diseño y despliegue.

4.10. Trello

Trello [49] es una herramienta que permite la creación, gestión y compartición de tableros en línea. En ellos se pueden crear listas y dentro de estas listas agregar tarjetas con cierto contenido.

Se ha utilizado para llevar a cabo el control del proyecto, así como las tareas realizadas y las que faltan por realizar.

Capítulo 5

Plan de riesgos y estimación de costes

5.1. Plan de riesgos

Un riesgo [27] es un evento o condición que, en el caso de ocurrir, puede tener un efecto positivo o negativo sobre los objetivos del proyecto.

Los riesgos pueden estar categorizados de diferentes maneras pero me basaré en el modelo definido por Kalle Lyytinen y sus compañeros. Afirma que los cuatro pilares fundamentales de un proyecto son las personas involucradas o actores, la tecnología utilizada, la estructura y las tareas a realizar. Los cuatro pilares están interconectados entre ellos, haciendo que, en diversas circunstancias, un problema dependa de otro o afecte de manera directa al resto.

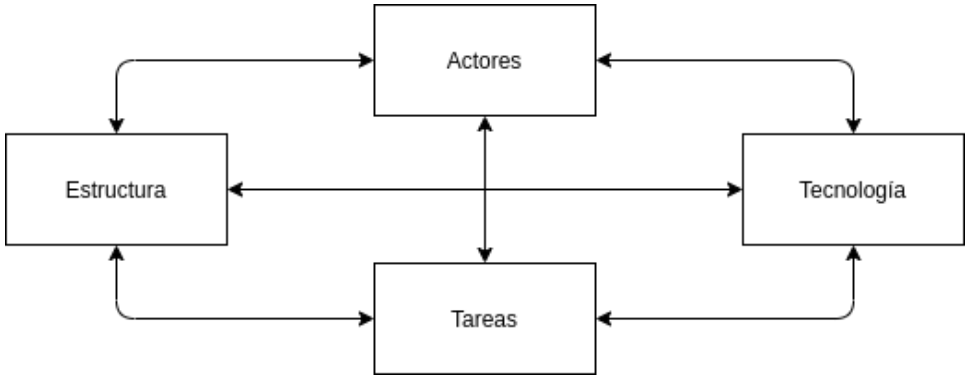

Figura 5.1: Entorno de riesgos de un proyecto.

Existen una serie de pautas o pasos para llevar a cabo un correcto plan de riesgos:

1. Identificación de riesgos.
2. Análisis de los riesgos.
3. Planificación de riesgos.
4. Monitorización de riesgos.

Lo más común es que los pasos del 1 al 3 se repitan de manera continua.

5.1.1. Riesgos encontrados en el proyecto

En este apartado se van a detallar los riesgos detectados a la hora de desarrollar el proyecto. Estos riesgos afectan a los cuatro pilares explicados con anterioridad y, dependiendo del tipo de riesgo y de su probabilidad, se explicará también el plan de mitigación y el plan de contingencia ante dicho riesgo.

El impacto, al igual que la probabilidad, puede ser bajo (poca probabilidad o poco impacto al proyecto), medio (una probabilidad moderada o un impacto sustancial) o alto (es muy probable que suceda o que el impacto al proyecto sea demasiado directo).

Los riesgos están descritos desde la tabla 5.1 hasta la tabla 5.7.

Riesgo	RG-01
Descripción	Una persona del equipo de desarrollo no presenta suficiente formación sobre las tecnologías empleadas
Impacto	Alto
Probabilidad	Medio
Plan de mitigación	Se utilizan tecnologías conocidas que se han utilizado en la carrera y en el entorno de trabajo
Plan de contingencia	Solicitar ayuda a gente con mayor experiencia

Tabla 5.1: Riesgo de falta de formación

Riesgo	RG-02
Descripción	Una persona del equipo cae enfermo y deja de poder realizar las tareas asignadas
Impacto	Alto
Probabilidad	Baja
Plan de mitigación	Disponer de tiempo extra al final del proyecto para, si hiciera falta, realizar un *sprint* de refuerzo
Plan de contingencia	Dejar varias tareas para futuras iteraciones

Tabla 5.2: Riesgo de enfermedad

Riesgo	RG-03
Descripción	Los requisitos cambian añadiendo funcionalidad al proyecto no prevista en un inicio
Impacto	Alto
Probabilidad	Baja
Plan de mitigación	Planear bien las tareas al inicio del proyecto
Plan de contingencia	Dejar varias tareas para futuras iteraciones

Tabla 5.3: Riesgo de cambios en los requisitos

Riesgo	RG-04
Descripción	No se cumplen los tiempos de entrega y el desarrollo del proyecto aumenta más de lo esperado
Impacto	Medio
Probabilidad	Media
Plan de mitigación	Centrarse en las tareas principales que conforman el núcleo del proyecto
Plan de contingencia	Eliminar tareas extra o complementarias para centrarse en las principales

Tabla 5.4: Riesgo de planificación optimista

Riesgo	RG-05
Descripción	Durante el desarrollo del proyecto se pueden producir varios fallos en las máquinas virtuales
Impacto	Medio
Probabilidad	Baja
Plan de mitigación	Mantener el entorno en condiciones óptimas
Plan de contingencia	Avisar a los técnicos

Tabla 5.5: Riesgo de fallo del entorno de laboratorio

Riesgo	RG-06
Descripción	Falta de control y de seguimiento del desarrollo por parte de los involucrados en el proyecto
Impacto	Alto
Probabilidad	Baja
Plan de mitigación	Fomentar las reuniones de rutina y mantener contacto en todo momento
Plan de contingencia	El desarrollador asume una mayor responsabilidad dentro del proyecto

Tabla 5.6: Riesgo de falta de seguimiento

Riesgo	RG-07
Descripción	Las tecnologías utilizadas son cambiadas por otras mejores para mejorar el proyecto
Impacto	Alto
Probabilidad	Media
Plan de mitigación	Formación previa sobre las nuevas tecnologías a usar
Plan de contingencia	Disponer de tiempo extra para realizar tareas incompletas

Tabla 5.7: Riesgo de cambios en las tecnologías utilizadas

5.2. Presupuesto

Para la realización del proyecto se cuenta con una persona como único desarrollador. El coste se ha obtenido a partir del sueldo base de un programador junior [30] que consta de un total de 19477 euros al año o, en horas, 9.75 euros. El proyecto consta de una estimación final de 300 horas por lo tanto el precio total del equipo de desarrollo será de 2925 euros.

A este proyecto no es necesario añadirle ningún coste por herramientas utilizadas (software, iconos, licencias, etc.) porque se ha orientado el trabajo al desarrollo con herramientas Open Source o de libre uso, así como de otras herramientas proporcionadas de manera gratuita por la universidad (gestor de repositorios, máquinas virtuales, etc.). Tampoco se tienen en cuenta los costes fijos ya que el trabajo realizado se ha hecho desde casa o desde la propia universidad, corriendo estos gastos por cuenta ajena al proyecto como tal.

Por último, detallar el valor del hardware utilizado así como su amortización durante el proyecto. Se ha utilizado una máquina con un valor de 1250 euros y, consultando la tabla de amortizaciones lineales [1], se puede observar que el coeficiente lineal máximo tiene un valor del 25 %. Por lo tanto, el coste de hardware durante 4 meses será de $1250 * 0{,}25 * (4/12)$, suponiendo un coste final de 104.17 euros.

Tipo de coste	Precio (€)
Herramientas	0
Costes fijos	0
Equipo de desarrollo	2925
Material	104.17
Total	3029.17

Tabla 5.8: Presupuesto final

Capítulo 6

Diseño

Este proyecto sigue una arquitectura cliente/servidor. A lo largo del capítulo se presentará la arquitectura, el diseño de la aplicación y el diagrama de despliegue.

6.1. Arquitectura cliente/servidor

La arquitectura cliente/servidor [46] es un modelo de diseño de software donde las tareas están divididas entre los proveedores de un recurso, llamados servidores, y los solicitantes de dichos servicios, llamados clientes. Ambos elementos se comunican a través de una red de comunicaciones: el cliente solicita un servicio al servidor, el servidor recibe la petición, la procesa y devuelve la respuesta solicitada. Un esquema sencillo sobre la arquitectura puede ser el mostrado en 6.1.

6.1.1. Características

Algunas de las características más importantes de esta arquitectura pueden ser:

- **Protocolos asimétricos:** el cliente inicia siempre la comunicación con el servidor. Los servidores esperan a las solicitudes de los clientes.
- **Encapsulación de servicios:** el servidor maneja el mensaje enviado y él determina como realizar el trabajo. Los servidores pueden presentar más de un servicio y actuar de manera independiente sin afectar a los clientes.
- **Integridad:** los clientes son independientes entre ellos, mientras que el código y los datos del servidor se mantienen centralizados permitiendo una mejor protección de los mismos.

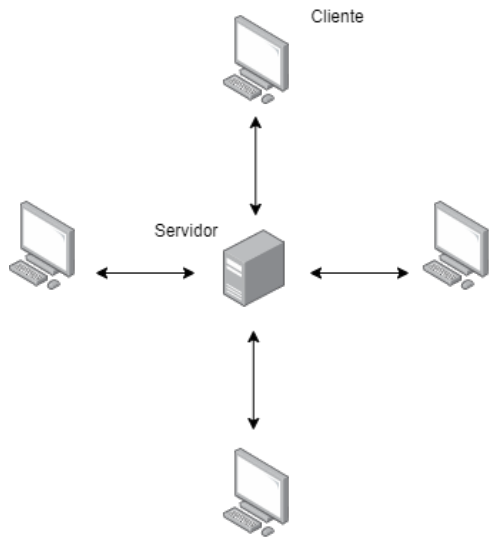

Figura 6.1: Arquitectura cliente/servidor

- **Transparencia de localización:** permite el acceso tanto al cliente como al servidor sin conocimiento de su localización.
- **Intercambios basados en mensajes:** la manera de comunicación establecida entre el cliente y el servidor es a través de mensajes.
- **Modularidad:** la presentación de un diseño modular permite que la aplicación sea tolerante a fallos.
- **Escalabilidad:** los sistemas pueden escalar verticalmente (migración a una máquina más potente) o escalar horizontalmente (añadir o eliminar clientes sin que el rendimiento se vea afectado).

6.1.2. Ventajas

- Un único servidor aloja toda la información, facilitando la gestión y protección de dicha información.
- Se pueden agregar recursos adicionales sin sufrir ningún tipo de interrupción.
- El acceso a los datos es transparente.
- Los nodos son independientes y estos solo solicitan información al servidor.
- La comunicación entre cliente y servidor se puede realizar sin importar la plataforma en la que se encuentran.

6.1.3. Desventajas

- Problemas de sobrecarga si existen demasiadas peticiones en un intervalo de tiempo muy corto. El servidor pueden presentar una congestión del tráfico y no poder resolver todas las demandas.
- Si el servidor falla los clientes no pueden recibir respuesta a sus peticiones.

6.1.4. Arquitectura cliente/servidor en este proyecto

Se ha utilizado la arquitectura cliente/servidor para la realización de este proyecto. Se ha tomado como referencia la estructura de una *botnet* centralizada (ejemplo ilustrado en la figura 2.1), presentando un servidor C&C central al cual se conectarán todos los *bots* existentes.

6.2. gRPC

gRPC es *framework* RPC (*Remote Procedure Call*) moderno, de código abierto y de alto rendimiento, que puede ser ejecutado en cualquier entorno. Fue desarrollado inicialmente por Google.

El modelo de gRPC consiste en llamadas a procedimientos remotos. Esta tecnología abstrae el código de la aplicación permitiendo al usuario crear llamadas entre diferentes elementos que están ubicados en otro lugar (una máquina remota), incluso entre diferentes lenguajes y arquitecturas. La conexión es transparente para el usuario, haciendo que nunca se preocupe por los detalles de comunicación entre el cliente y el servidor.

6.2.1. Características

- Se ejecuta sobre HTTP/2.
- Bajo consumo de CPU, ancho de banda y latencia.
- Gracias a HTTP/2 se obtiene un control del flujo de datos, con capacidad de enviar múltiples peticiones simultáneamente.
- Multilenguaje, está presente en los lenguajes de programación más usados en todo el mundo.
- Presenta balance de carga.
- Posee soporte de autenticación basada en SSL, ALTS o Google-Token.
- Por defecto utiliza Proto3 como método de serialización en vez de JSON.

6.2.2. Protocol Buffers

Los Protocol Buffers son un mecanismo de código abierto que se utiliza para serializar los datos de una manera estructurada y neutral, tanto al lenguaje como a la plataforma utilizada.

Se utiliza en gRPC para describir tanto la interfaz de servicio como la estructura de los mensajes de dicha interfaz.

Sus principales características son las siguientes:

- Son eficientes, es decir, son detallados y descriptivos. Al ser de un tamaño mucho menor que sus competidores proporcionan un mayor rendimiento.
- Se utilizan para intercambiar mensajes entre servicios, no entre navegadores.
- Son compilables, pudiendo generar bibliotecas y otros recursos en múltiples lenguajes de programación.
- Se pueden especificar tipos concretos de datos, así como campos de validación.

En un fichero de extensión `.proto` se definen los servicios y sus llamadas RPC y los mensajes que se van utilizar. Como se puede observar, en estos mensajes se define el tipo de dato así como su orden. El fichero debe ser compilado por un compilador específico para obtener las bibliotecas y el código necesario para su utilización. A continuación podemos ver un ejemplo de la estructura de un fichero `.proto`.

```
// Definición del servicio
service HelloService {
  // Envía un saludo
  rpc SayHello (HelloRequest) returns (HelloResponse);
}

// Mensaje de petición
message HelloRequest {
  string greeting = 1;
}

// Mensaje de respuesta
message HelloResponse {
  string reply = 1;
}
```

6.2.3. Funcionamiento

gRPC se basa en la idea de definir un servicio, especificando los métodos con sus parámetros y sus respuestas. Todos estos servicios y métodos, junto a los mensajes, son definidos

en un fichero `.proto`. En el lado del servidor se implementa una interfaz y se ejecuta un servidor gRPC para manejar las peticiones de los clientes. Por otro lado, el cliente posee código auxiliar que proporciona los mismos métodos que el servidor.

El diagrama 6.2 muestra un esquema simple de como interactúan las diferentes interfaces entre ellas.

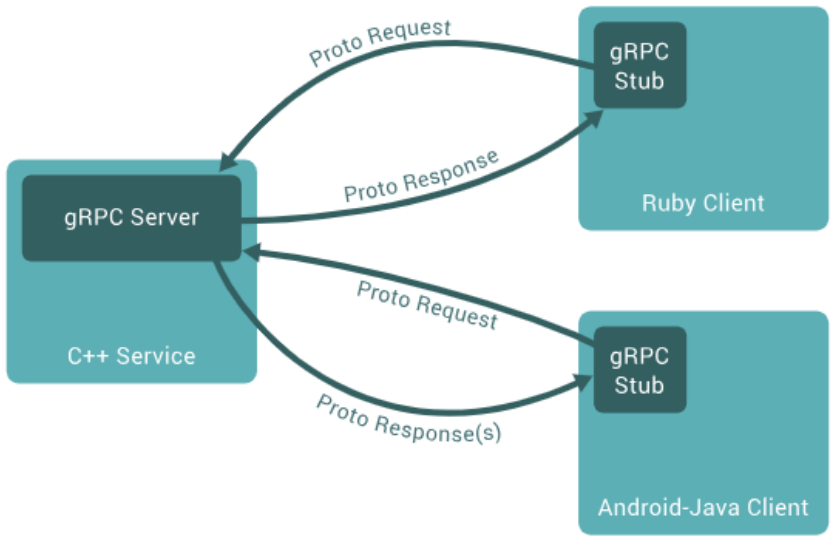

Figura 6.2: Diagrama de conceptos [4]

Como se ha mencionado anteriormente, los clientes y los servidores gRPC se pueden ejecutar en diferentes arquitecturas y lenguajes, permitiendo la escalabilidad y el multilenguaje sin esfuerzo adicional, todo gracias a los ficheros .proto y los compiladores disponibles.

En gRPC existen 4 variantes:

- ***Unary* RPCs.** El cliente envía una única solicitud y este recibe una única respuesta.

```
rpc SayHello(HelloRequest) returns (HelloResponse);
```

- ***Server streaming* RPCs.** El cliente envía una única solicitud y obtiene una secuencia de mensajes como respuesta. El cliente recibe mensajes hasta que no haya más. gRPC garantiza el orden de los mensajes.

```
rpc LotsOfReplies(HelloRequest) returns (stream HelloResponse);
```

- ***Client streaming* RPCs.** El cliente envía una secuencia de mensajes al servidor. Una vez el cliente haya terminado de enviar los mensajes, el servidor los trata y devuelve una única respuesta. gRPC garantiza el orden de los mensajes.

```
rpc LotsOfGreetings(stream HelloRequest) returns (HelloResponse);
```

- ***Bidirectional streaming* RPCs.** Transmisión bidireccional de mensajes en los que ambos envían una secuencia de mensajes. Los dos flujos operan de manera independiente, haciendo que tanto clientes como servidores puedan leer y escribir en el orden que deseen. Se conserva el orden de los mensajes en cada flujo.

```
rpc BidiHello(stream HelloRequest) returns (stream HelloResponse);
```

6.2.4. Ventajas

- Fácil de aplicar gracias a los archivos .proto.
- Mensajes sobre HTTP/2.
- La estructura por capas reduce el esfuerzo de programación.
- Presenta autenticación integrada.

6.2.5. Desventajas

- gRPC está aún en un desarrollo temprano.
- La solución de errores es compleja, requiere de instancias adicionales.
- Depende de una red estable y potente.
- No es compatible con el *multicasting*.

6.2.6. gRPC en este proyecto

Se ha utilizado gRPC frente a otros servicios como REST para la comunicación entre el servidor y los clientes o *bots*. Los *bots* pertenecientes a la *botnet* se comunican de manera directa y segura sobre HTTP/2 con el servidor. Dependiendo de la tarea asignada, el *bot* enviarán un tipo de mensaje u otro (de los descritos anteriormente y definidos en el fichero .proto (Apéndice B)) al servicio a través de la interfaz.

A mayores, la aplicación web de administración que utiliza el *botmaster*, se comunica con el servidor también a través de canales creados por gRPC. Esta aplicación web reside en el propio servidor, tal y como se puede observar en el modelo de despliegue.

6.3. Modelo de dominio

Una vez analizadas las historias de usuario presentes en las tablas 3.2 y 3.3 se elabora el modelo de dominio presente en la figura 6.3.

El modelo está centrado en los mensajes generados para la comunicación gRPC. La estructura de este fichero está disponible en el Apéndice B.

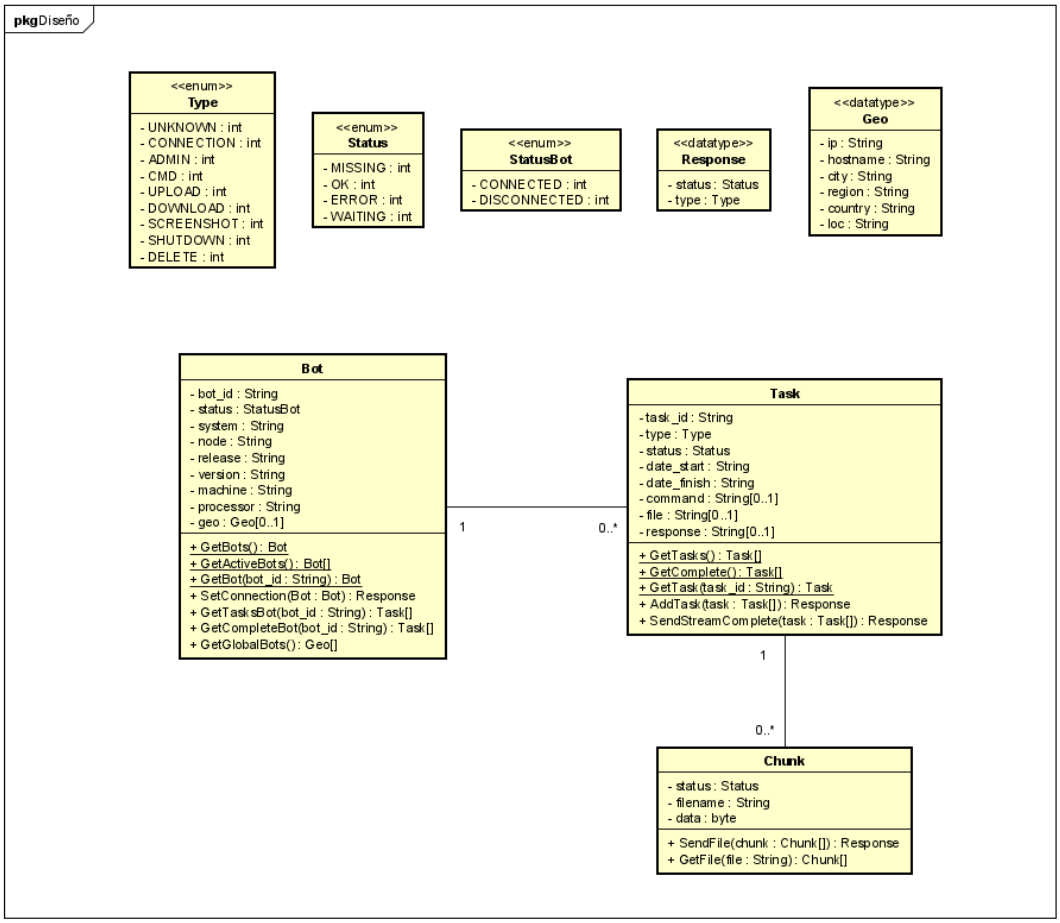

Figura 6.3: Modelo de dominio

6.4. Modelo de despliegue

El despliegue del proyecto está representado en la figura 6.4.

Los *bots* se conectarán con el servidor mediante conexiones HTTP/2, utilizando las interfaces creadas por gRPC para la comunicación. El servidor está compuesto por el servidor gRPC que atiende las llamadas RPC tanto del *botmaster* como la de los *bots*.

A mayores el servidor contiene una aplicación Flask que utilizará el *botmaster* para gestional la *botnet* y acceder al servidor gRPC. Para acceder a la aplicación Flask, el *botmaster* utilizará un navegador web cualquiera.

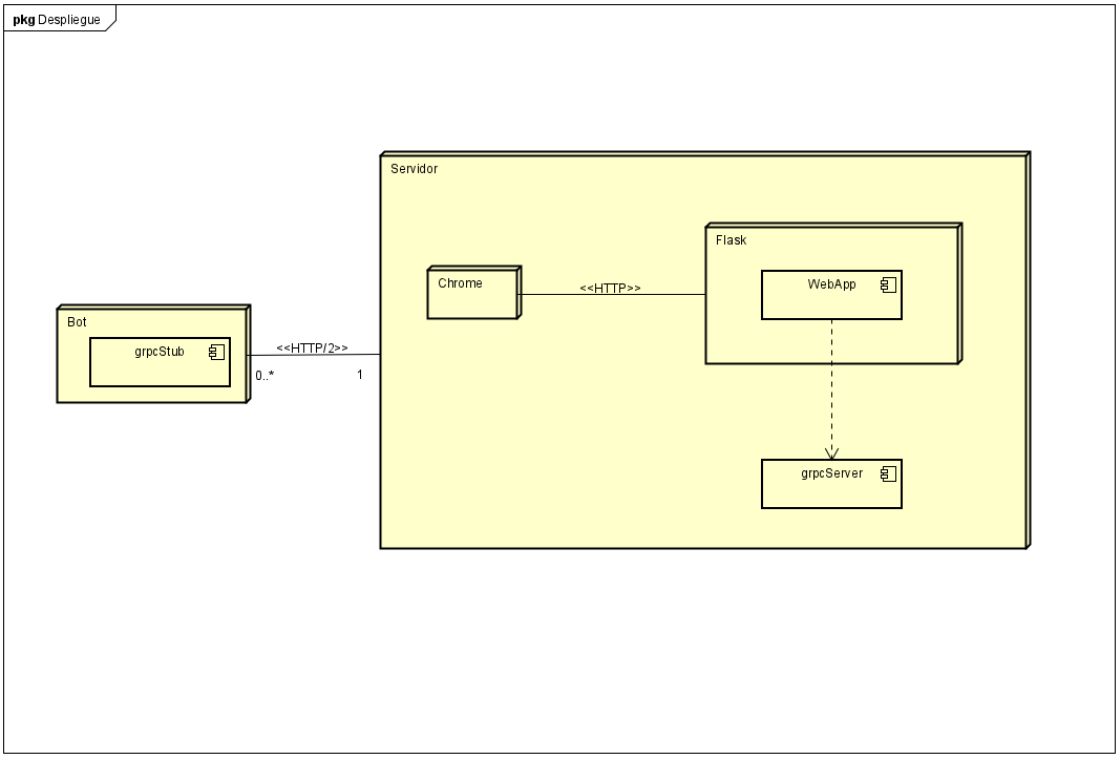

Figura 6.4: Modelo de despliegue

Capítulo 7

Implementación

Este capítulo trata sobre el entorno de laboratorio donde se ha desarrollado el proyecto, así como la estructura final del código y la implementación de la *botnet*.

7.1. Entorno de laboratorio

Para el desarrollo de la *botnet*, así como las pruebas y su despliegue, desde la Escuela de Ingeniería Informática de la Universidad de Valladolid se ha utilizado un entorno de laboratorio controlado.

La figura 7.1 representa el esquema del entorno de laboratorio. En él podemos observar la organización de la red entre diferentes secciones. Aunque presente muchas redes y sistemas, nuestro servidor se encuentra en una máquina de la red "ATACANTES", mientras que nuestra máquina objetivo está en la red "INTRANET", protegida por un *firewall* de red.

La máquina Kali Linux que cumplirá el rol de servidor C&C posee las siguientes características:

- **Versión S.O.**: Kali Linux 2021.1 (x64)
- **Procesador**: 2 procesadores Common KVM processor 2.53 GHz
- **Memoria**: 2 GB RAM
- **Almacenamiento**: 20 GB de espacio de almacenamiento

La máquina Windows presente en la red "INTRANET" tendrá asignado el rol de sistema infectado, siendo un sistema vulnerable donde realizar la ejecución del *bot* de manera segura y controlada. Posee las siguientes características:

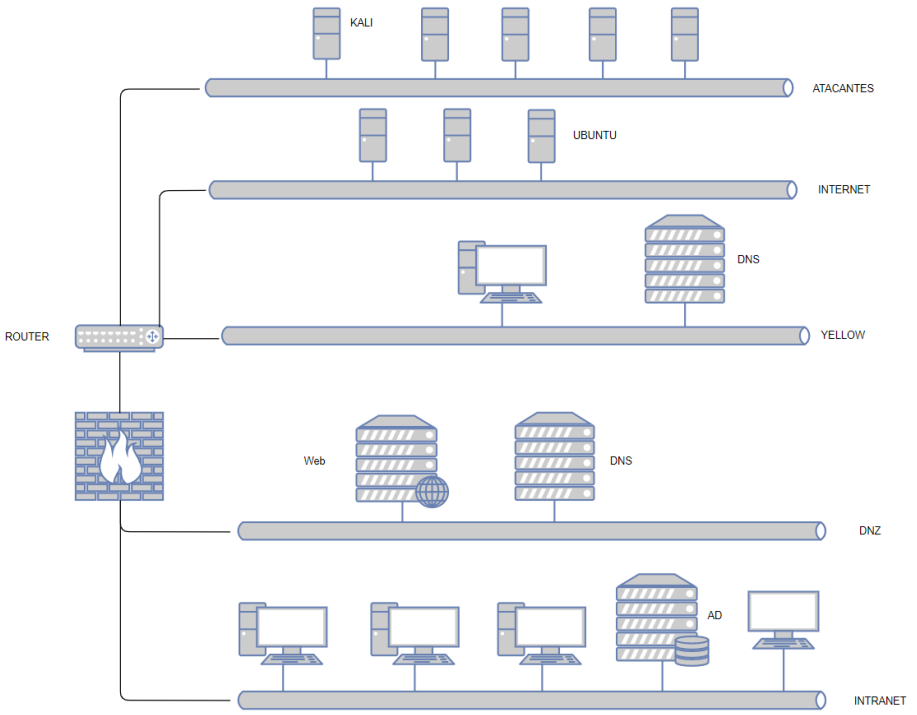

Figura 7.1: Entorno de laboratorio

- **Versión S.O.**: Windows 10 Pro compilación 19042.1083 (x64)
- **Procesador**: 2 procesadores Common KVM processor 2.53 GHz
- **Memoria**: 4 GB RAM
- **Almacenamiento**: 40 GB de espacio de almacenamiento

La idea de utilizar un entorno de laboratorio reside en controlar la ejecución de una herramienta como la desarrollada en este proyecto, evitando que terceros o personas ajenas a su desarrollo se vean afectadas de alguna manera. A mayores, al ser un entorno controlado, se puede monitorizar todas las conexiones y eventos que se realicen en las diferentes máquinas.

7.2. Implementación de la *botnet*

En este apartado trataremos el funcionamiento de la *botnet*, así como las tareas que realiza y el funcionamiento de la aplicación web de gestión de la red.

Se ha realizado, con ayuda del *framework* MITRE ATT&CK mencionado anteriormente, un esquema para definir el comportamiento de la *botnet* (Figura 7.2). Se ha utilizado la matriz de empresa ya que da soporte a la plataforma objetivo, Windows.

En la figura señalada está marcado en verde los procedimientos utilizados, así como las técnicas utilizadas.

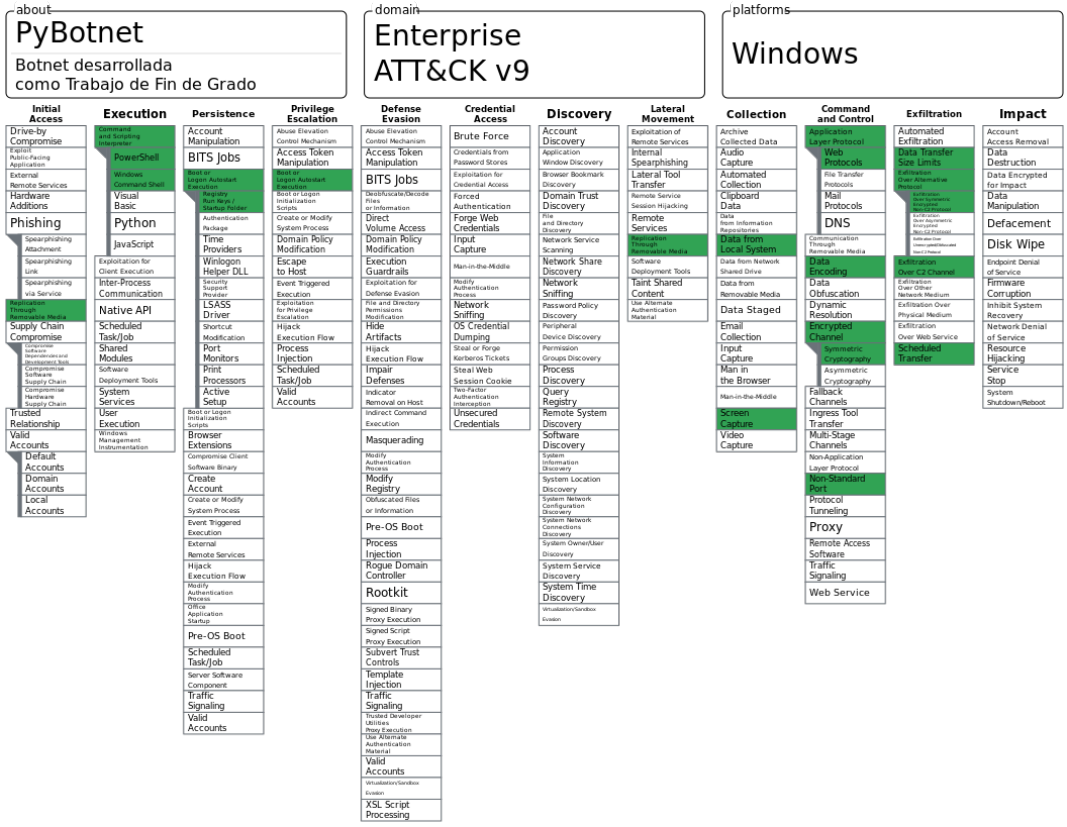

Figura 7.2: Comportamiento de la *botnet*

En el capítulo 2 se ha hablado del ciclo de vida que sigue una *botnet*. En este proyecto partimos desde el momento que el *malware* se ha ejecutado, es decir, desde el momento de explotación. A partir de aquí, los siguientes apartados hablarán de la comunicación entre el *bot* y el servidor, como se instala este en el sistema y como realiza las diferentes acciones para las cuales ha sido desarrollado.

7.2.1. Comunicación con C&C

El servidor C&C presenta una arquitectura cliente/servidor centralizada. Este servidor está situado en la máquina Kali Linux descrita anteriormente. Es diseñado bajo la tecnología gRPC con el lenguaje de programación Python. La comunicación siempre se realiza desde el cliente al servidor, es decir, desde el zombi al servidor C&C.

La comunicación con el servidor central se realiza a través de las llamadas RPC descritas en el fichero .proto presente en el Apéndice B utilizando canales seguros y encriptados.

La *botnet* está desarrollada para que, aunque el servidor no esté disponible, los *bots* en los distintos dispositivos se mantengan a la espera hasta que vuelvan a poder ponerse en contacto con el servidor C&C.

Cuando el zombi consigue establecer conexión con el servidor, ya sea por primera vez o después de no haber podido ponerse en contacto, se le envía la información básica del dispositivo infectado, así como su geoposición. Cuando el servidor recibe dicha información puede hacer dos cosas: registrar un nuevo *bot* o actualizar su estado.

Al registrar un nuevo *bot*, se registra la información del dispositivo obtenida en la base de datos, así como su posición. Se le da un valor de "conectado"haciendo que aparezca disponible para la asignación de tareas. En cambio, si ya está registrado en la base de datos y solo si es necesario, se actualizan los valores guardados. A mayores, sea el estado que sea anterior, volverá a aparecer como "conectado".

La primera vez que un *bot* se conecta se creará, tanto un *log* en la carpeta destinada a ellos, como una carpeta para las tareas de descargar ficheros o capturas de pantalla.

7.2.2. Persistencia

Los *bots* están diseñados para mantenerse en el dispositivo objetivo. En este caso lo primero que se hace es crear un número de serie único, que será el número identificativo del *bot* de cara al servidor. Este número es escondido en la carpeta del sistema `%appdata%`, seguido de una copia del propio *malware* para una ejecución paralela.

En este caso se ha diseñado que, si no existe persistencia en el dispositivo, se cree permitiendo que cada vez que se inicie sesión en la cuenta de usuario el *bot* copiado a la carpeta del sistema se ejecute y establezca conexión con el servidor (siguiendo el formato descrito en el apartado anterior) utilizando el número de serie generado anteriormente. Si dicho número no existe o se ha eliminado, se volverá a crear.

7.2.3. Tareas implementadas

A parte de establecer comunicación entre el dispositivo infectado y el servidor los *bots* pueden realizar diferentes tareas de manera independiente. Esta es la parte central del desarrollo. Para saber que tareas debe realizar se ha establecido un sistema de sondeo al servidor cada cierto tiempo para preguntarlo.

Estas tareas han sido diseñadas de manera modular tanto en el servidor como en el *bot*, permitiendo añadir nuevos tipos de tareas o quitarlas sin afectar al comportamiento final de la *botnet*. La estructura del mensaje tarea está presente en el fichero `.proto` en el Apéndice B.

Las tareas que soporta esta *botnet* son las siguientes:

Ejecución de código remoto

Esta tarea consiste en ejecutar comandos en el dispositivo de manera remota y, una vez completados, obtener la respuesta y guardar esta en el servidor y en la base de datos.

Para la realización de esta tarea el *bot* presenta un módulo con las funciones necesarias para la creación de instancias de PowerShell en Windows para la ejecución de las líneas de comando que se le pida. El comando va insertado dentro de la tarea que ha obtenido de la lista de tareas pendientes que el servidor le ha devuelto.

Subir un fichero a la máquina remota

Esta tarea consiste en subir un fichero a la máquina o dispositivo remoto a la misma carpeta donde se encuentra ejecutándose el *malware* (carpeta `%appdata%`).

En este caso la tarea contiene el nombre del fichero que el *botmaster* desea subir al dispositivo. Con esta información el *bot* envía una petición al servidor pidiendo el fichero y la respuesta del servidor será una serie de paquetes de datos o *chunks*, con la información acerca del fichero.

Para asegurarnos que el fichero está disponible a la hora de enviarlo se crea una copia en una carpeta del servidor por si, por un error, el original es borrado o modificado.

Descargar un fichero de la máquina remota

Esta tarea consiste en bajar un fichero de la máquina o dispositivo remoto al servidor.

En este caso la tarea contiene la dirección absoluta del fichero que deseamos descargar. El *bot* lo único que debe hacer cuando lo encuentre es convertir la información en *chunks* y enviarlo al servidor, listo para guardarlo en la carpeta que posee para este tipo de ficheros. El fichero es posible que no exista, por lo que la tarea en ese caso se dará como completada pero con un estado de "no localizado".

Tomar una captura de pantalla de la máquina remota

Esta tarea consiste en la toma de una captura de pantalla de la pantalla principal del dispositivo infectado.

Una vez se ha realizado la tarea la imagen se envía de la misma manera que la descrita anteriormente para descargar un fichero de la máquina remota, solo que, en este caso, el nombre de la imagen empezará por "SCREENSHOT"seguido del identificador de la tarea.

Apagar el *bot* de manera remota

Esta tarea consiste en detener la ejecución del *bot* en el dispositivo.

De manera remota se detiene la ejecución del ejecutable en el sistema pero no se elimina. Esto produce que no esté disponible para añadirle más tareas hasta que el sistema se reinicie y se vuelva a conectar con el servidor C&C.

Eliminar el *bot* de manera remota

Esta tarea consiste en detener la ejecución del *bot*, borrar el número identificativo asignado y eliminar la persistencia del dispositivo, haciéndolo inaccesible de nuevo por parte de la *botnet*.

La eliminación del dispositivo de la *botnet* será efectiva hasta que nadie vuelva a activar el ejecutable localizado en la carpeta `%appdata%` de manera manual, ya que este seguirá presente en el dispositivo remoto.

7.2.4. Aplicación web

En el proyecto se ha desarrollado una aplicación web de gestión para la *botnet* que se comunica directamente con el servidor.

Esta aplicación permite gestionar todos los aspectos relacionados con los *bots*, así como con las tareas y los ficheros que existen en el servidor. Esta aplicación será la que utilice el *botmaster* para realizar cualquier acción sobre los dispositivos.

Alguna de las principales características son las recogidas a continuación. Se puede ver los *bots* que hay registrados, así como sus tareas relacionadas y las propiedades del dispositivo que ha infectado. Se puede gestionar, visualizar y crear nuevas tareas de manera sencilla. Posee un acceso directo a la visualización y descarga de *logs* y ficheros. Por último posee un mapa global donde están representados todos los bots registrados.

Para saber más de su uso y funcionamiento, toda la información está recogida en el manual de usuario presente en el Apéndice A.

7.3. Estructura del código

En este apartado se muestra la estructura y la jerarquía final del código. Algunas partes no se tratan ahora por complejidad, sino que serán desarrolladas más adelante.

```
|   .gitignore
|   requirements.txt
```

```
|   setup.py
|   start.sh
|   stop.sh
|
|---bot (se comentará en el apartado 7.4)
|
|---certs
|
|---flaskapp (se comentará en el apartado 7.3)
|
|---protos
|   |   botnet.proto
|   |   botnet_pb2.py
|   |   botnet_pb2_grpc.py
|   |   __init__.py
|
|---server (se comentará en el apartado 7.2)
|
|---venv
```

Los primeros ficheros que aparecen en la lista son ficheros de configuración del entorno y los *scripts* para iniciar y parar el servidor.

- **.gitignore**: fichero que permite decirle a Git que ficheros o carpetas debe ignorar.
- **requirements.txt**: fichero que se usa para especificar qué paquetes de Python se requieren para poder utilizar el proyecto.
- **setup.py**: fichero de configuración del entorno virtual de Python.
- **start.sh**: *script* para iniciar el servidor gRPC y la aplicación web sin necesidad de escribir los comandos por la terminal.
- **stop.sh**: *script* para detener el servidor gRPC y la aplicación web sin necesidad de escribir los comandos por la terminal.

Una vez descritos los ficheros presentes vamos a describir las carpetas presentes en el directorio raíz del proyecto, como su contenido.

- **bot**: carpeta que contiene el código del *bot*. Se comentará en el apartado 7.3.3.
- **certs**: carpeta destinada a almacenar la clave privada del servidor, así como los diferentes certificados que se creen.
- **flaskapp**: carpeta que contiene el código de la aplicación web. Se comentará en el apartado 7.3.2.
- **protos**: carpeta que contiene el módulo de Python del código generado por el compilador de los Protocol Buffers a partir del archivo `.proto`, presente también en la carpeta.

- **botnet.proto**: fichero proto que contiene la definición del servicio y de los mensajes para su serialización a través del compilador de los Protocol Buffers. El contenido del fichero puede ser consultado en el Apéndice B.
- **botnet_pb2.py**: archivo Python resultante de la compilación del fichero `.proto` por parte del compilador.
- **botnet_pb2_grpc.py**: archivo Python resultante de la compilación del fichero `.proto` por parte del compilador.
- **__init__.py**: archivo que permite al intérprete de Python saber que el directorio `protos` contiene código para la generación de un módulo.

- **server**: carpeta que contiene el código del servidor gRPC. Se comentará en el apartado 7.3.1.

- **venv**: entorno virtual para el proyecto. Nos permite crear un entorno controlado de directorios. Se utiliza para poder personalizar la configuración y las dependencias de paquetes de Python sin cambiar su configuración global.

7.3.1. Estructura del código del servidor gRPC

En el apartado anterior vimos la estructura general del proyecto. En esta ocasión veremos la estructura del servidor gRPC.

El servidor se encuentra dentro de la carpeta `server`. Está compuesto por un fichero Python que será el corazón del servidor, una carpeta para la base de datos, otra para los ficheros que se transmitan entre el servidor y los diferentes *bots*, la carpeta de los *logs* o registros correspondientes tanto a los *bots* como al servidor y, por último, los módulos creados para dar soporte al servidor.

```
server
|   |   grpcserver.py
|   |
|   |---db
|   |
|   |---files
|   |   |---bots
|   |   |---upload
|   |
|   |---logs
|   |   |---bots
|   |   |---server
|   |
|   |---modules
|       |
|       |---utils
```

- **grpcserver.py**: fichero Python encargado de lanzar el servidor gRPC. Contiene la lógica necesaria para las llamadas gRPC, así como las llamadas a los diferentes módulos dependiendo del tipo de tarea a realizar.
- **db**: carpeta destinada a almacenar la base de datos del servidor. Esta base de datos es un fichero llamado `database.db`.
- **files**: carpeta creada para la gestión de ficheros entre el servidor y los *bots*. A su vez contiene otras dos carpetas:
  - **bots**: carpeta destinada a almacenar los ficheros remotos que el *bot* descargue en el servidor. Está estructurada de tal manera que, por cada *bot* registrado en la *botnet*, existe una carpeta a su nombre. Estas carpetas permiten almacenar jerárquicamente el contenido.
  - **upload**: carpeta destinada a almacena una copia de los ficheros que se quieran enviar a los *bots* desde el servidor.
- **logs**: carpeta destinada a almacenar los *logs* de los *bots* y del servidor.
  - **bots**: carpeta destinada a almacenar los *logs* de los *bots*.
  - **server**: carpeta destinada a almacenar el *log* del servidor.
- **modules**: carpeta que contiene los diferentes módulos desarrollados para la comunicación del servidor con los *bots* y con la aplicación web.
  - **utils**: carpeta que contiene varios submódulos cuyas funciones de utilidad son utilizadas por los módulos principales.

7.3.2. Estructura del código de la aplicación web

La aplicación web ha sido desarrollada con el *framework* Flask y su función es dar soporte en el servidor local al *botmaster* para poder gestionar la *botnet* y los *bots*.

```
flaskapp
|   |   pybotnet.py
|   |
|   |---app
|       |---modules
|       |
|       |---static
|       |   |---css
|       |   |
|       |   |---img
|       |
|       |---templates
```

- **pybotnet.py**: fichero Python encargado de lanzar la aplicación web. Contiene la lógica necesaria para gestionar las peticiones que reciba.

- **app**: carpeta con la lógica de la aplicación.
  - **modules**: carpeta que contiene los diferentes módulos Python utilizados para la comunicación entre la aplicación y el servidor.
  - **static**: carpeta donde se almacena todo lo relacionado con los aspectos permanentes de la aplicación.
    - **css**: carpeta donde se encuentra almacenado el CSS.
    - **img**: carpeta donde se encuentran almacenadas las imágenes que se utilizan de manera estática.
  - **templates**: carpeta donde se encuentran las plantillas HTML de todas las páginas que componen la aplicación.

7.3.3. Estructura del código del *bot*

Por último, presentamos el código que compone el *bot*. Este código, como se verá en el capítulo de despliegue, es compilado con el fin de conseguir un ejecutable para las máquinas Windows.

```
bot
|   |   icon.ico
|   |   robot.py
|   |
|   |---exe
|   |
|   |---modules
```

- **icon.ico**: icono que se le asignará al ejecutable compilado del *bot*.
- **robot.py**: fichero Python correspondiente al programa principal del *bot*.
- **exe**: carpeta destinada para almacenar el ejecutable compilado del *bot*.
- **modules**: carpeta que contiene los diferentes módulos desarrollados para la realización de tareas y la comunicación entre el *bot* y el servidor.

7.4. Licencia

A este proyecto se le ha dotado de una licencia MIT [21]. Es una licencia permisiva, breve y simple cuya única condición es la preservación de los avisos de licencia y derechos de autor. Presenta permiso para uso comercial, así como de distribución, modificación y uso privado.

Capítulo 8

Pruebas

En este capítulo se muestran las diferentes pruebas realizadas a lo largo del proyecto con el fin de asegurar un producto robusto, probado y seguro frente a posibles ataques o errores de usuario.

P-001	Ejecutar el *bot* en la máquina Windows del entorno
Descripción	El *bot* compilado se ejecuta en el entorno del laboratorio dentro de la máquina Windows asignada.
Resultado esperado	El *bot* se ejecuta con normalidad y crea persistencia.
Resultado obtenido	El *bot* se ejecuta con normalidad y crea persistencia.

Tabla 8.1: Prueba ejecución del *bot* en el entorno de laboratorio

P-002	Comunicación del *bot* con el servidor encendido
Descripción	El *bot* está en ejecución y quiere ponerse en contacto con el servidor cuando este está encendido.
Resultado esperado	El *bot* envía la información al servidor y este la almacena en la base de datos.
Resultado obtenido	El servidor recibe la información de manera correcta y almacena los datos en la base de datos.

Tabla 8.2: Prueba de comunicación del *bot* con el servidor encendido

P-003	Comunicación del *bot* con el servidor apagado
Descripción	El *bot* está en ejecución y quiere ponerse en contacto con el servidor cuando este está apagado.
Resultado esperado	El *bot* se mantiene a la espera ejecutándose hasta que el servidor esté disponible de nuevo.
Resultado obtenido	El *bot* se mantiene a la espera ejecutándose hasta que el servidor esté disponible de nuevo.

Tabla 8.3: Prueba de comunicación del *bot* con el servidor apagado

P-004	El *bot* solicita tareas al servidor
Descripción	El *bot* pregunta si tiene tareas que realizar.
Resultado esperado	El *bot* recibe una posible secuencia de tareas pendientes por parte del servidor.
Resultado obtenido	El *bot* recibe una posible secuencia de tareas pendientes por parte del servidor.

Tabla 8.4: Prueba del *bot* solicitando tareas al servidor

P-005	El *bot* realiza la tarea de ejecución de un comando
Descripción	El *bot* realiza la ejecución de la tarea encargada de ejecutar comandos de manera remota y devolver el resultado al servidor.
Resultado esperado	El *bot* ejecuta el comando y devuelve el resultado.
Resultado obtenido	El *bot* ejecuta el comando y devuelve el resultado.

Tabla 8.5: Prueba del *bot* realizando la tarea de ejecución de un comando

P-006	El *bot* realiza la tarea de subir fichero desde el servidor
Descripción	El *bot* pide al servidor el fichero concreto que tiene indicado en la tarea.
Resultado esperado	El *bot* envía el nombre del fichero y este recibe los datos referentes al fichero solicitado.
Resultado obtenido	El *bot* envía el nombre del fichero y este recibe los datos referentes al fichero solicitado.

Tabla 8.6: Prueba del *bot* realizando la tarea de subir un fichero a la máquina remota

P-007	El *bot* realiza la tarea de enviar un archivo que no se encuentra al servidor
Descripción	El *bot* tiene la tarea de enviar un archivo que no existe al servidor.
Resultado esperado	El *bot* informa al servidor que el fichero solicitado no está disponible.
Resultado obtenido	El *bot* informa al servidor que el fichero solicitado no está disponible.

Tabla 8.7: Prueba del *bot* al realizar la tarea de enviar un fichero inexistente

P-008	El *bot* realiza la tarea de enviar un archivo al servidor
Descripción	El *bot* tiene la tarea de enviar un archivo que existe al servidor.
Resultado esperado	El *bot* envía al servidor los datos en bruto del fichero solicitado.
Resultado obtenido	El *bot* envía al servidor los datos en bruto del fichero solicitado.

Tabla 8.8: Prueba del *bot* al realizar la tarea de enviar un fichero

P-009	El *bot* realiza la tarea de tomar una captura de pantalla
Descripción	El *bot* realiza una captura de la pantalla del dispositivo y la envía al servidor.
Resultado esperado	El servidor recibe la captura de pantalla correctamente.
Resultado obtenido	El servidor recibe la captura de pantalla correctamente.

Tabla 8.9: Prueba del *bot* al realizar la tarea de tomar una captura de pantalla

P-010	El *bot* realiza la tarea de apagarse
Descripción	El *bot* se apaga.
Resultado esperado	El *bot* detiene su ejecución.
Resultado obtenido	El *bot* detiene su ejecución.

Tabla 8.10: Prueba del *bot* al realizar la tarea de apagarse

P-011	El *bot* realiza la tarea de eliminarse
Descripción	El *bot* elimina la persistencia y el numero de identificación del dispositivo infectado.
Resultado esperado	El *bot* queda inoperativo en la máquina infectada.
Resultado obtenido	El *bot* queda inoperativo en la máquina infectada.

Tabla 8.11: Prueba del *bot* al realizar la tarea de eliminarse

P-012	El *bot* envía las tareas completadas al servidor
Descripción	Una vez finalizadas las tareas, estas son enviadas al servidor.
Resultado esperado	El servidor recibe una sucesión de tareas completadas.
Resultado obtenido	El servidor recibe una sucesión de tareas completadas.

Tabla 8.12: Prueba del *bot* enviando las tareas completadas

P-013	Acceder a la página de los *bots* de la aplicación con el servidor apagado
Descripción	Accedemos a la página donde se encuentra el listado de todos los bots registrados por la red con el servidor apagado.
Resultado esperado	Página de error 500, no se ha encontrado el servidor.
Resultado obtenido	Página de error 500, no se ha encontrado el servidor.

Tabla 8.13: Prueba acceso a la página de los *bots* con el servidor apagado

P-014	Acceder a la página de los *bots* de la aplicación
Descripción	Accedemos a la página donde se encuentra el listado de todos los *bots* registrados por la red.
Resultado esperado	La página nos mostrará el listado completo de los *bots*.
Resultado obtenido	La página nos mostrará el listado completo de los *bots*.

Tabla 8.14: Prueba de la aplicación accediendo a la página de los *bots*

P-015	Acceder a la información de un *bot* existente
Descripción	Se pulsa sobre el identificador de un *bot* disponible en la lista.
Resultado esperado	Se muestra la página con la información del *bot*.
Resultado obtenido	Se muestra la página con la información del *bot*.

Tabla 8.15: Prueba de la aplicación accediendo a un *bot* existente

P-016	Prueba de la aplicación accediendo a un *bot* que no existe
Descripción	Insertamos una URL con un identificador de un *bot* que no existe.
Resultado esperado	Se muestra la página de la información del *bot* pero nos comunica que no se ha podido encontrar.
Resultado obtenido	Se muestra la página de la información del *bot* pero nos comunica que no se ha podido encontrar.

Tabla 8.16: Prueba de la aplicación accediendo a un *bot* que no existe

P-017	Acceder a la página de las tareas con el servidor apagado
Descripción	Se accede a la página donde se muestran todas las tareas pendientes y completadas cuando el servidor está apagado.
Resultado esperado	Página de error 500, no se ha encontrado el servidor.
Resultado obtenido	Página de error 500, no se ha encontrado el servidor.

Tabla 8.17: Prueba de la aplicación accediendo a la página de las tareas con el servidor apagado

P-018	Acceder a la pagina de las tareas
Descripción	Se accede a la página donde se muestran todas las tareas pendientes y completadas.
Resultado esperado	Se muestra la página de tareas.
Resultado obtenido	Se muestra la página de tareas.

Tabla 8.18: Prueba de la aplicación accediendo a la página de las tareas

P-019	Acceso a la página de añadir tarea sin ningún *bot* disponible
Descripción	Accedemos a la página por URL sin tener ningún *bot* conectado o disponible para poder agregar una tarea.
Resultado esperado	Error, página de aviso de que no puede acceder a esa sección.
Resultado obtenido	Error, página de aviso de que no puede acceder a esa sección.

Tabla 8.19: Prueba de la aplicación accediendo a la página de añadir prueba sin tener *bots* disponibles

P-020	Agregar una nueva tarea
Descripción	El administrador rellena los campos necesarios de las tareas para poder enviar la tarea al servidor.
Resultado esperado	Tarea añadida y redirección a la página de tareas.
Resultado obtenido	Tarea añadida y redirección a la página de tareas.

Tabla 8.20: Prueba de la aplicación agregando una nueva tarea

P-021	Agregar una tarea con algún campo sin completar
Descripción	El administrador no rellena alguno de los campos obligatorios para alguna tarea.
Resultado esperado	La página no le deja continuar, saltan avisos de que faltan campos por completar.
Resultado obtenido	La página no le deja continuar, saltan avisos de que faltan campos por completar.

Tabla 8.21: Prueba de la aplicación agregando una nueva tarea sin completar los campos

P-022	Descargar un fichero desde la aplicación
Descripción	Se descarga un fichero (*log* o cualquier fichero) desde la aplicación.
Resultado esperado	El fichero se descarga correctamente.
Resultado obtenido	El fichero se descarga correctamente.

Tabla 8.22: Prueba de la aplicación descargando ficheros

P-023	Descargar un fichero que no existe desde la aplicación
Descripción	Se descarga un fichero (*log* o cualquier fichero) que no existe desde la aplicación.
Resultado esperado	Página de error 404, no se ha encontrado el recurso solicitado.
Resultado obtenido	Página de error 404, no se ha encontrado el recurso solicitado.

Tabla 8.23: Prueba de la aplicación descargando ficheros que no existen

Capítulo 9

Despliegue

Este capítulo trata sobre el despliegue y los pasos que hay que realizar para poder desplegar las diferentes partes de la *botnet* desarrollada. A mayores, se explicará como se compila y se despliega en la máquina Windows el *bot*.

9.1. Preparación del entorno

Para preparar el entorno para el despliegue de la *botnet* necesitamos tener una versión de Python igual o superior a la versión 3.8.6.

Una vez que se tiene Python instalado en nuestro sistema debemos descargar el código fuente del proyecto, localizado en `https://gitlab.inf.uva.es/javbarr/PyBotnet`.

El servidor puede ser ejecutado tanto en Linux como en Windows. Es posible que para las máquinas Windows sea necesario instalar algún programa adicional. En este caso el despliegue se realizará sobre una máquina Kali Linux.

Una vez que tenemos el proyecto descargado en una carpeta, en el directorio raíz creamos un entorno virtual de Python que llamaremos `venv`. Para ello, estando en la raíz del proyecto, ejecutamos el siguiente comando

```
python3 -m venv venv
```

Tras crear el entorno virtual debemos activarlo. Para ello debemos escribir

```
source ./venv/bin/activate
```

Si queremos desactivar el entorno virtual solo debemos escribir

```
deactivate
```

Una vez tenemos el entorno virtual activado, debemos instalar los paquetes de Python existentes en el fichero `requirements.txt`. También debemos instalar los módulos desarrollados a lo largo del proyecto. Para ello, primero, debemos ejecutar el siguiente comando desde el directorio raíz

```
python3 -m pip install -r requirements.txt
```

Una vez instalados los módulos Python del fichero de requisitos debemos compilar el fichero .proto. Para ello utilizamos el siguiente comando, especificando la ubicación del fichero existente en la carpeta protos. El comando a ejecutar es

```
python3 -m grpc.tools.protoc --python_out=. --grpc_python_out=. --proto_path=.
./protos/botnet.proto --experimental_allow_proto3_optional
```

Ahora instalaremos nuestro propio paquete creado con los módulos del proyecto. Se instalarán bajo el nombre pybotnet 1.0 y debemos escribir, también desde el directorio raíz del proyecto

```
python3 -m pip install .
```

Una vez finalizada la instalación de los módulos debemos crear la clave privada y los certificados para nuestro servidor, nuestra aplicación web y nuestros *bots*. Para ello se utilizará OpenSSL como herramienta de creación de certificados y claves.

Si en nuestro directorio raíz no existe una carpeta certs para los certificados lo primero que tenemos que hacer es crear una carpeta con ese nombre. Esto es importante ya que, tanto los nombres aquí presentes, tanto la estructura descrita es la utilizada de manera interna por el servidor.

Una vez la carpeta exista podemos crear la clave privada del servidor con el siguiente comando. Entramos en la carpeta y ejecutamos los siguientes comandos

```
openssl genrsa -out server.key 2048
```

A partir de la clave privada creamos el certificado para las llamadas locales.

```
openssl req -new -x509 -nodes -sha256 -days 365 -key server.key -out
localhost.crt -subj "/C=ES/ST=Castilla y Leon/L=Valladolid
/O=Universidad de Valladolid/OU=Escuela de Ing Informatica CN = localhost"
```

También, a partir de la clave privada, creamos el certificado para los *bots* con la dirección IP pública del servidor de la *botnet*.

```
openssl req -new -x509 -nodes -sha256 -days 365 -key server.key -out env.crt
-subj "/C=ES/ST=Castilla y Leon/L=Valladolid/O=Universidad de Valladolid
/OU=Escuela de Ing Informatica" -addext "subjectAltName = IP:xxx.xxx.xxx.xxx"
```

9.2. Despliegue del servidor

Una vez tenemos todo el entorno preparado podemos iniciar nuestro servidor. El servidor está compuesto del servidor gRPC y de la aplicación web soportada en Flask. Ambos elementos se pueden ejecutar de manera independiente, por terminal, sin ningún problema.

Para facilitar al *botmaster* la ejecución de ambas cosas se han creado dos *scripts* de gestión.

Para poder ejecutar ambos *scripts* en la máquina Kali Linux primero debemos darle permisos a ambos ficheros.

```
chmod +x start.sh
chmod +x stop.sh
```

Ambos *scripts* se deben ejecutar desde el directorio raíz del proyecto. El *script* `start.sh` inicializa el servidor gRPC por un lado y la aplicación web en la dirección `localhost:8080` por otro. En cambio, el *script* `stop.sh` detiene ambas ejecuciones.

9.2.1. Servidor gRPC

Es posible ejecutar sólo el servidor gRPC sin necesidad de ejecutar la aplicación web.

Para ello podemos ejecutar el fichero Python directamente desde el directorio raíz utilizando los siguientes comandos.

```
# Ejecución normal
python3 ./server/grpcserver.py
```

```
# Ejecución en segundo plano
nohup python3 ./server/grpcserver.py > /dev/null 2>&1 &
```

Si la ejecución es normal podemos detener el servidor gRPC directamente utilizando sobre la terminal la combinación de teclas `CTRL+C`.

En cambio, si la ejecución es en segundo plano, podemos utilizar la siguiente línea como uno de los muchos métodos para detener la ejecución de manera segura.

```
kill $(pgrep -f './server/grpcserver.py')
```

9.2.2. Aplicación web

También es posible ejecutar solo la aplicación web sin necesidad de ejecutar el servidor gRPC.

En este caso las comunicaciones de las páginas que necesiten del servidor gRPC (no todas) darán un error 500, informando que el servidor gRPC no se ha iniciado. Aún así, es posible su ejecución de manera independiente, siguiendo unos pasos muy similares a los descritos en el apartado anterior.

Podemos ejecutar el fichero Python directamente desde el directorio raíz utilizando los siguientes comandos.

```
# Ejecución normal
python3 ./flaskapp/pybotnet.py

# Ejecución en segundo plano
nohup python3 ./flaskapp/pybotnet.py > /dev/null 2>&1 &
```

Si la ejecución es normal podemos detener la aplicación al igual que el servidor gRPC, utilizando la combinación de teclas `CTRL+C`.

En cambio, si la ejecución es en segundo plano, podemos utilizar la siguiente línea como uno de los muchos métodos para detener la ejecución de manera segura.

```
kill $(pgrep -f './flaskapp/pybotnet.py')
```

9.3. Despliegue del *bot*

El *bot* es un archivo ejecutable `.exe` para las máquinas Windows que contiene la lógica para realizar las tareas asignadas, ponerse en contacto con el servidor C&C y ocultarse en el dispositivo.

9.3.1. Creación del *bot*

Lo primero que se debe hacer es crear, a partir de los ficheros y los módulos del proyecto, un ejecutable `.exe` válido para esa tarea. Para ello utilizaremos el paquete PyInstaller con el fin de generar el ejecutable.

Antes de nada debemos asegurarnos que la dirección IP presente en el archivo `./bot/robot.py` corresponde a la dirección del servidor gRPC.

Si es así, lo siguiente que debemos realizar es la compilación desde línea de comandos. Para ello se establece un icono al ejecutable para dotar a este de una mayor credibilidad y, también, se le añade el certificado creado con la dirección IP del servidor para establecer la conexión de manera segura. Ambos archivos deben ir especificados con la ruta absoluta.

Este comando se debe ejecutar desde el directorio raíz del proyecto y utilizando el entorno virtual.

```
pyinstaller -D -F --add-data "ruta_absoluta/certs/env.crt;." -n "bot" --specpath
"./bot/exe/spec" --distpath "./bot/exe" --workpath "./bot/exe/build"
--noconsole --icon=ruta_absoluta/bot/icon.ico ./bot/robot.py
```

Al finalizar, en la carpeta `./bot/exe` vamos a tener un archivo ejecutable llamado `bot.exe` que será el que usemos para infectar la máquina Windows.

9.3.2. Ejecución del *bot* en la máquina Windows

Una vez creado el *bot* debemos desplegarlo en la máquina objetivo. La infección remota de la máquina no está desarrollada en este proyecto, es decir, suponemos que el ejecutable llega a la máquina objetivo. Una vez está el ejecutable en la máquina solo debemos ejecutar el `.exe`. Para que el *bot* se active una vez ejecutado el fichero, es necesario reiniciar la máquina.

Firewall

Para poder ejecutar el ejecutable que contiene el código del *bot* sin ningún inconveniente, se desactiva todas las protecciones del *firewall* de Windows.

Capítulo 10

Conclusiones y líneas futuras

10.1. Conclusiones finales

A continuación se resume brevemente los objetivos de desarrollo completados en este proyecto:

- Se ha estudiado la documentación existente sobre las *botnets*.
- Se ha configurado un entorno de laboratorio controlado para la realización del proyecto.
- Se ha diseñado y desarrollado la funcionalidad de una *botnet*.
- Se han estudiado y se han implementado el *framework* gRPC y los Protocol Buffers para el paso de mensajes entre los diferentes elementos del proyecto.
- Se ha diseñado y desarrollado una aplicación web para el control de la *botnet*.
- Se han estudiado varios mecanismos de seguridad del sistema operativo Windows.

En este proyecto se ha realizado un estudio sobre las *botnets*, así como su arquitectura y estructura y las ventajas e inconvenientes de cada una. Junto a este estudio se ha conseguido desarrollar un servidor central utilizando la tecnología gRPC y los Protocol Buffers para la comunicación entre las diferentes partes del proyecto. Una de esas partes del proyecto es el *bot* o troyano, que infecta la máquina objetivo y realiza diferentes tareas de manera remota e independiente. La otra parte del proyecto ha sido la aplicación web escrita en Flask para poder controlar la *botnet*. Para comprobar la funcionalidad de todo se ha utilizado un entorno de laboratorio controlado.

En relación con los objetivos personales:

- He aumentado mi conocimiento de Python y del uso de módulos y librerias externas.

- La utilización de Scrum para este proyecto me ha permitido aprender en profundidad como se trabaja bajo este modo.
- Se ha aumentado mi interés en cuanto a las *botnets* y el protocolo IP.
- He aumentado mi conocimiento en cuanto al sistema operativo Windows.

10.2. Líneas futuras

Las posibles mejoras que se pueden añadir a este proyecto son:

- **Dominios dinámicos**: los *bots* se comunican con el servidor C&C a través de una IP estática. Una versión futura debería implementar una lista de dominios variable para evitar descubrir y bloquear la comunicación tan fácilmente.
- **Profundizar en técnicas de ataque y escalado de privilegios**: existen un gran número de técnicas que podrían ser implementadas para explotar aún más los recursos de los dispositivos infectados, así como obtener un mayor control sobre ellos. Una versión futura debería implementar dichas técnicas con el fin de presentar una *botnet* con mayor funcionalidad.
- **Técnicas de evasión de antivirus**: los *bots* suelen tener técnicas de evasión o para evitar que los antivirus detecten actividad inusual ya que muchos van ocultos en otros archivos. Una versión futura debería implementar e inyectar el *malware* directamente sobre otro archivo para eludir el antivirus.
- **Desarrollo de una aplicación móvil**: el control del servidor C&C se realiza desde la misma máquina. En una versión futura lo ideal sería desarrollar una aplicación móvil propia para controlar los dispositivos sin tener que utilizar directamente la máquina del servidor.

Apéndice A

Manual de usuario

Este manual está destinado al uso de la aplicación web para el control de la *botnet*.

La página principal de la web, cuya apariencia se puede ver en la figura A.1, describe el proyecto desarrollado, así como las diferentes características de la aplicación web. Presenta una barra superior fija que contiene el logo con el nombre y las diferentes páginas existentes.

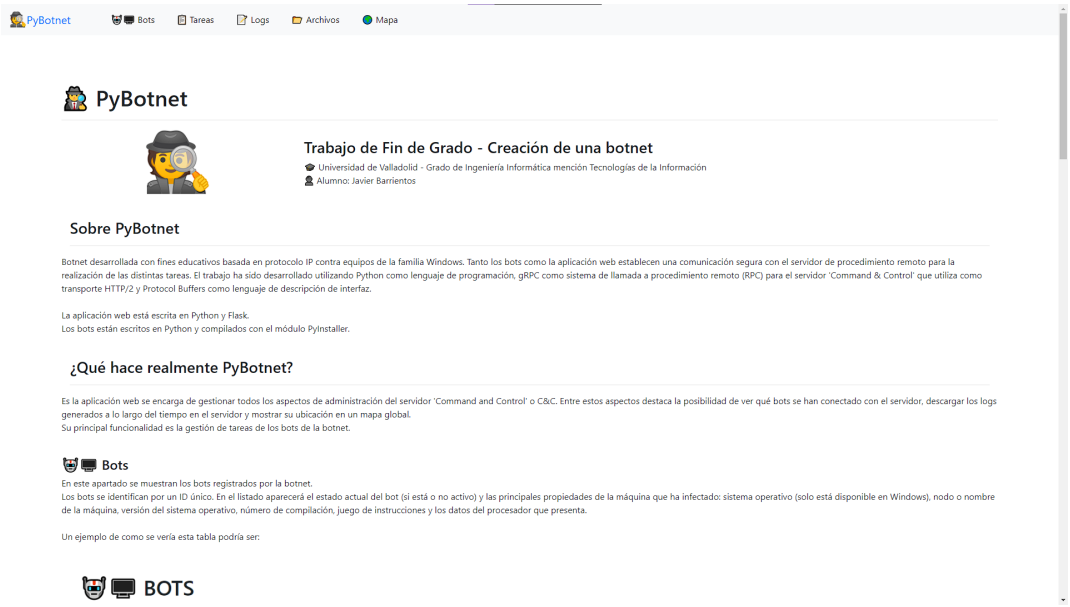

Figura A.1: Apariencia final de la página principal.

El botón "Bots"nos llevará a la página (Figura A.2) que contiene el listado de todos los *bots* disponibles en la *botnet*. Los *bots* se identifican por un ID único. En el listado aparecerá

el estado actual del *bot* (si está o no activo) y las principales propiedades de la máquina que ha infectado: sistema operativo (solo está disponible en Windows), nodo o nombre de la máquina, versión del sistema operativo, número de compilación, juego de instrucciones y los datos del procesador que presenta.

Posee un contador para saber el número total de bots, así como un botón para actualizar la página directamente.

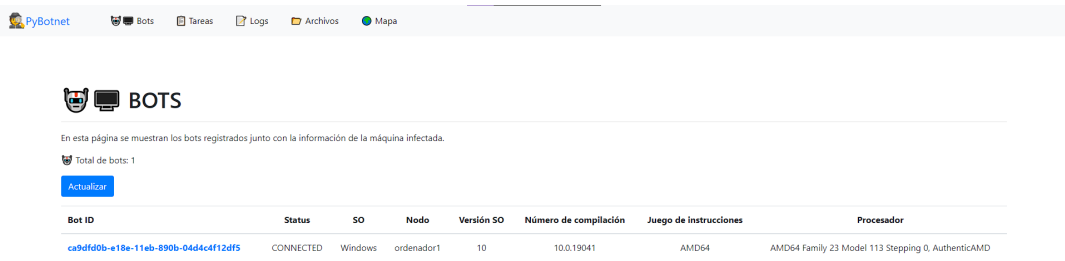

Figura A.2: Apariencia final de la página de los *bots*.

En caso de no presentar ningún *bot* la *botnet* nos saldrá un mensaje en vez de la tabla avisándonos que no hay ningún *bot* en este momento.

Pulsando en el ID del *bot* nos llevará a la página de información del *bot* seleccionado. Dicha página la podemos ver desglosada en dos partes. La primera parte (Figura A.3) presenta la información básica del *bot*, tales como su estado, las tareas que tiene pendientes y las tareas completadas. A su vez tiene, de nuevo, la información de la máquina infectada así como su geoposición aproximada en un mapa.

En la parte final de la página podemos ver un listado de las tareas pendientes que tiene el *bot* por hacer, como también un listado de las tareas completadas (Figura A.4).

Las tablas de las tareas contiene el ID de la tarea, el tipo de tarea realizada o por realizar, la fecha de cuando se ha agregado dicha tarea y, si se ha completado, la fecha de finalización de la tarea. Si se pulsa sobre el ID de la tarea te llevará a la página de información de las tarea (Figura A.7).

Si a la hora de acceder a la información del *bot* este no está disponible, saldrá un mensaje de información avisando que el *bot* que está buscando no se ha encontrado en la base de datos.

El botón "Tareas"nos llevará a la página principal de gestión de las tareas de la *botnet* (Figura A.5). En esta página tenemos arriba del todo (si hay algún *bot* disponible) un botón con el título "Añadir Tarea", para ir a la página con el formulario para añadir una nueva tarea (Figura A.6). También dispone de un botón para actualizar la página.

Está completada por dos tablas que contienen, si existen tareas de su condición, tareas. Estas tareas están representadas por su ID, el tipo de tarea, su estado, la fecha de cuando

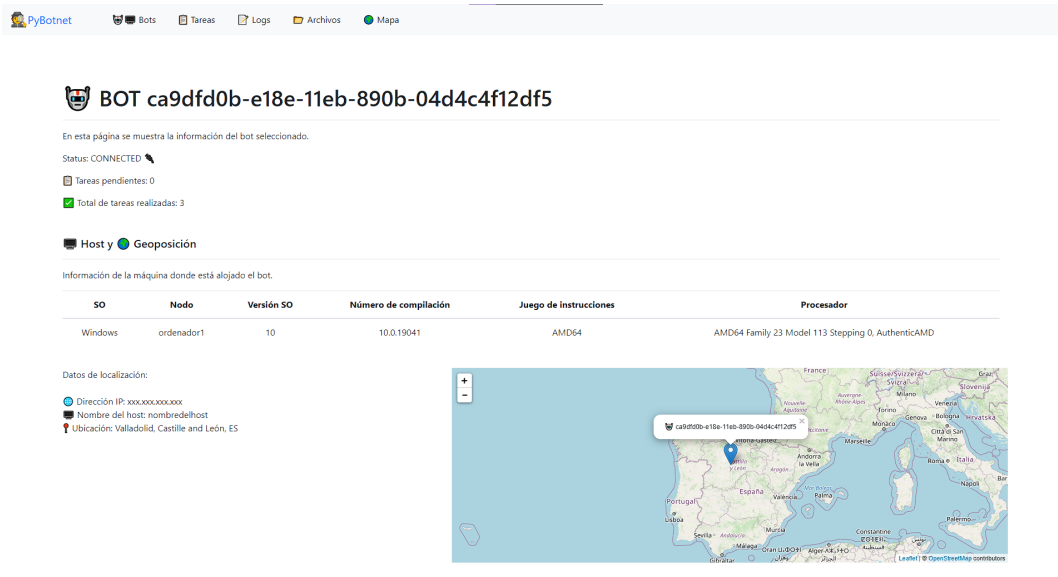

Figura A.3: Apariencia final de la página de información de un *bot* con la información y la geolocalización.

fueron añadidas, la fecha, si la tarea se ha completado, de cuando fueron completadas y el ID del *bot* que ha realizado dicha tarea o la va a realizar.

Las tablas poseen unos filtros para poder filtrar las tareas según el tipo de tarea, el estado de la tarea y por el ID del *bot*. Esto es muy útil cuando se poseen demasiadas tareas registradas esperando o completadas.

Si la *botnet* presenta uno o varios *bots*, podemos añadir tareas para realizar. Pulsando el botón para añadir una nueva tarea nos llevará a la página del formulario (Figura A.6) donde tendremos una lista con todos los *bots* y varias opciones.

De la lista de *bots* debemos seleccionar mínimo un *bot* y, en el caso que se quiera, se pueden seleccionar todos a la vez. Una vez seleccionado cuales son los *bots* que queremos que hagan la tarea debemos seleccionar el tipo de tarea.

Actualmente hay 6 tareas:

- Ejecución de código remoto
- Subir fichero al host remoto
- Descargar fichero del host remoto
- Tomar captura de pantalla del host remoto
- Apagar *bot*

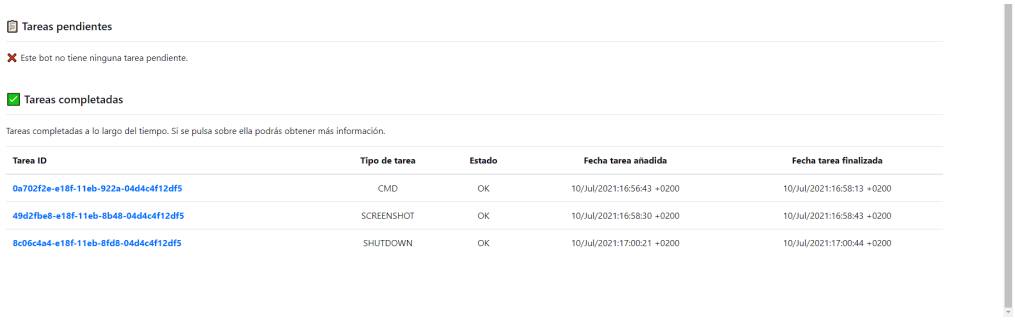

Figura A.4: Apariencia final de la página de información de un *bot* con las tareas.

- Eliminar *bot*

Dependiendo de la tarea seleccionada se podrá desbloquear un cuadro de diálogo de los disponibles para añadir información complementaria a la tarea tal como, el comando que se desea ejecutar, el fichero que se quiere subir o la dirección del fichero que se quiere descargar.

Una vez completado el formulario solo debemos darle al botón de Añadir Tarea y se nos redirigirá a la página principal de las tareas.

Si pulsamos sobre el ID de la tarea nos llevará a la página de la información sobre la tarea seleccionada (Figura A.7). En esta página tenemos la información base de la tarea, el tipo de tarea, su estado, la fecha de cuando se ha creado, la fecha, si existe, de cuando se ha completado y el *bot* que ha llevado a cabo o va a llevar a cabo la tarea. Si se pulsa sobre el *bot* vas directamente a su página de información.

Dependiendo de la tarea realizada se mostrará, a continuación de la tabla de información básica, la descripción de la tarea. Se informará también si la tarea no se ha completado.

El botón "Logs"nos llevará a la página donde se ubican todos los *logs* recogidos a lo largo del tiempo por el servidor (Figura A.8). Están presentes los *logs* de los diferentes *bots* y el *log* del servidor global. Cuando se pulsa sobre el nombre de un *log* puedes abrirlo en una nueva ventana o incluso descargarlo.

El botón "Archivos"nos llevará a la página donde se ubican todos los archivos descargados por los diferentes *bots* en el servidor (Figura A.9). Se ordenan por carpetas cuyo nombre es el identificador del *bot*. Dependiendo del archivo presente y su extensión tendrá un icono u otro con el fin de mejorar la compresión del archivo presente. Si se pulsa sobr el archivo se puede abrir en una nueva pestaña o descargarlo.

El botón "Mapa"nos llevará a la página con un mapa global con todos los *bots* presentes (Figura A.10). Este mapa contiene un icono de localización por cada *bot* y, si pulsamos sobre uno de ellos, podemos obtener más información acerca de la geolocalización como son la dirección IP, el nombre del dispositivo y la ciudad donde se ubica (Figura A.11).

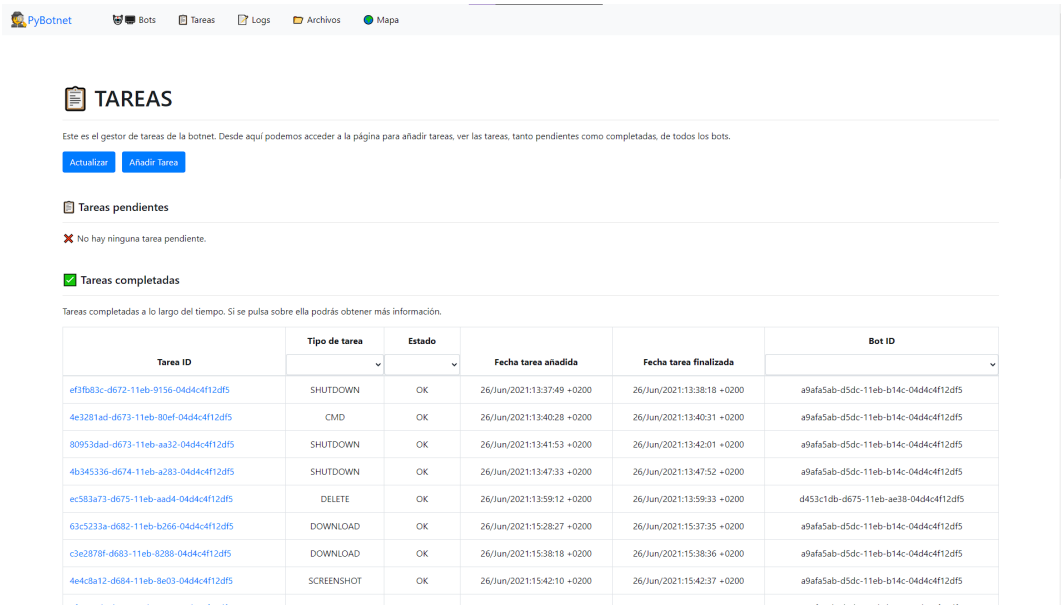

Figura A.5: Apariencia final de la página de las tareas.

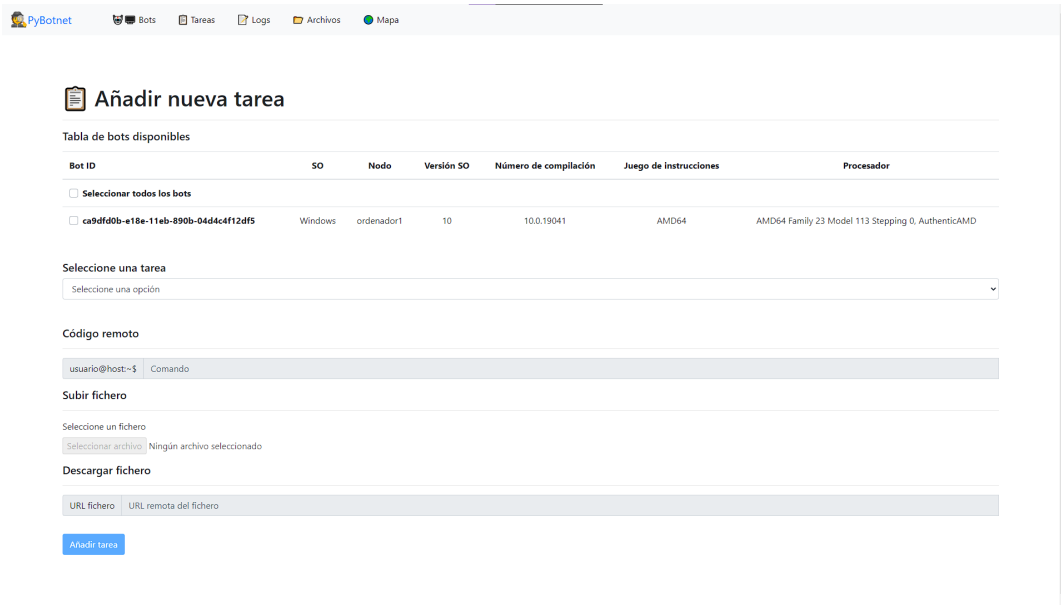

Figura A.6: Apariencia final de la página del formulario para añadir una nueva tarea.

Figura A.7: Apariencia final de la página de información de una tarea.

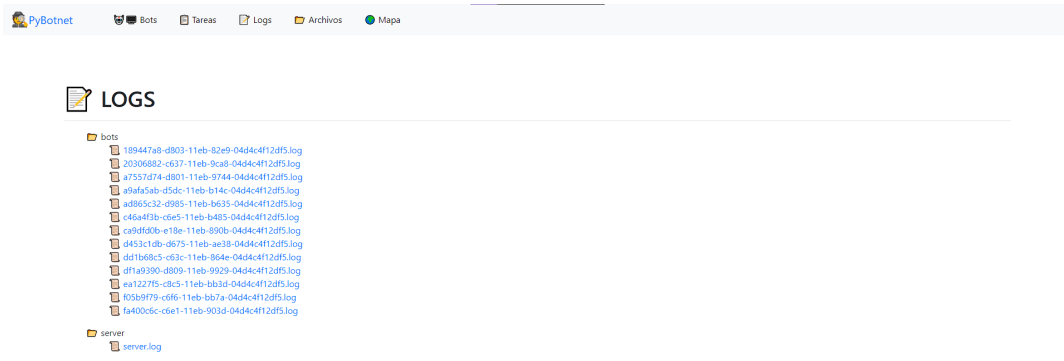

Figura A.8: Apariencia final de la página de *logs*.

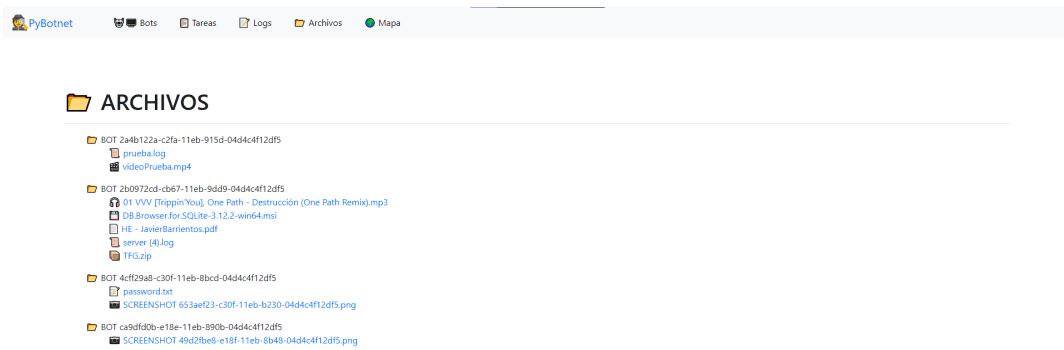

Figura A.9: Apariencia final de la página de archivos.

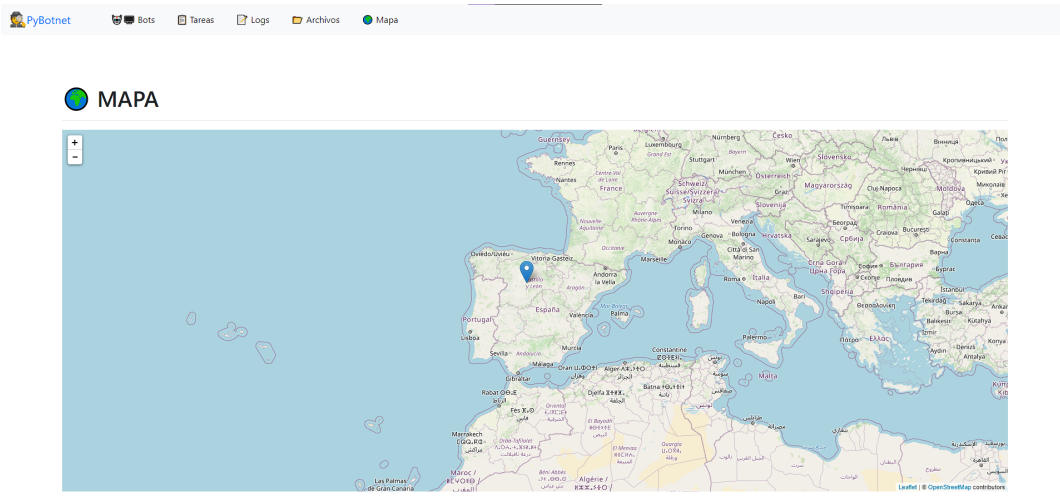

Figura A.10: Apariencia final de la página del mapa.

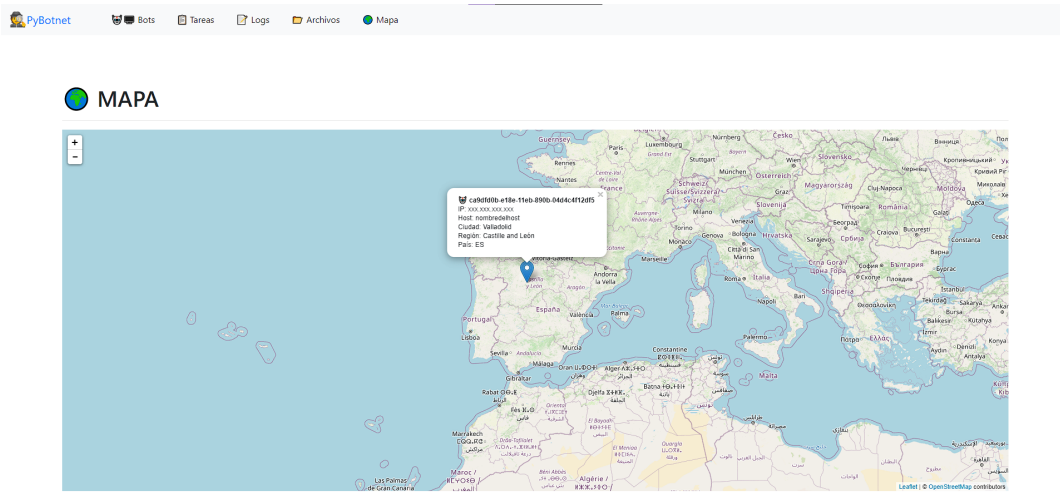

Figura A.11: Apariencia final de la página del mapa con el diálogo del *bot* seleccionado.

Apéndice B

Código del fichero .proto

```
syntax = "proto3";

enum Type {
    UNKNOWN = 0;
    CONNECTION = 1;
    ADMIN = 2;
    CMD = 3;
    UPLOAD = 4;
    DOWNLOAD = 5;
    SCREENSHOT = 6;
    SHUTDOWN = 7;
    DELETE = 8;
}

enum Status {
    MISSING = 0;
    OK = 1;
    ERROR = 2;
    WAITING = 3;
}

enum StatusBot {
    CONNECTED = 0;
    DISCONNECTED = 1;
}

message Empty{}

message Geo {
    string bot_id = 1;
    string ip = 2;
    string hostname = 3;
```

```
    string city = 4;
    string region = 5;
    string country = 6;
    string loc = 7;
}

message Bot {
    string bot_id = 1;
    StatusBot status = 2;
    string system = 3;
    string node = 4;
    string release = 5;
    string version = 6;
    string machine = 7;
    string processor = 8;
    optional Geo geo = 9;
}

message TaskId {
    string task_id = 1;
}

message Task {
    string task_id = 1;
    string bot_id = 2;
    Type type = 3;
    Status status = 4;
    string date_start = 5;
    string date_finish = 6;
    optional string command = 7;
    optional string file = 8;
    optional string response = 9;
}

message Response {
    Status status = 1;
    Type type = 2;
}

message BotId {
    string bot_id = 1;
}

message Chunk {
    string bot_id = 1;
    Status status = 2;
    string filename = 3;
    bytes data = 4;
}

message File {
```

```
    string bot_id =1;
    string file = 2;
}

service Botnet {
    rpc SendFile(stream Chunk) returns (Response);
    rpc GetFile(File) returns (stream Chunk);
    rpc GetBots(Empty) returns (stream Bot);
    rpc GetActiveBots(Empty) returns (stream Bot);
    rpc GetBot(BotId) returns (Bot);
    rpc SetConnection(Bot) returns (Response);
    rpc SendStreamComplete(stream Task) returns (Response);
    rpc GetTasksBot(BotId) returns (stream Task);
    rpc GetCompleteBot(BotId) returns (stream Task);
    rpc GetTasks(Empty) returns (stream Task);
    rpc GetComplete(Empty) returns (stream Task);
    rpc GetTask(TaskId) returns (Task);
    rpc AddTask(stream Task) returns (Response);
    rpc GetGlobalBots(Empty) returns (stream Geo);
}
```

Bibliografía

[1] AEAT (Agencia Estatal de Administración Tributaria). *Tabla de coeficientes de amortización lineal*. URL: https://www.agenciatributaria.es/AEAT.internet/Inicio/_Segmentos_/Empresas_y_profesionales/Empresas/Impuesto_sobre_Sociedades/Periodos_impositivos_a_partir_de_1_1_2015/Base_imponible/Amortizacion/Tabla_de_coeficientes_de_amortizacion_lineal_.shtml (visitado 12-03-2021).

[2] Vladimir Agafonkin. *Leaflet*. URL: https://leafletjs.com (visitado 14-06-2021).

[3] Astah. *Astah*. URL: https://astah.net (visitado 01-03-2021).

[4] gRPC Authors. *gRPC*. URL: https://grpc.io (visitado 30-03-2021).

[5] gRPC Authors. *Python—gRPC*. URL: https://grpc.io/docs/languages/python (visitado 12-04-2021).

[6] Stuart Bishop. *pytz*. URL: https://pypi.org/project/pytz (visitado 18-04-2021).

[7] INCIBE (Instituto Nacional de Ciberseguridad). *Emotet: características y funcionamiento*. URL: https://www.incibe-cert.es/blog/emotet-caracteristicas-y-funcionamiento (visitado 05-03-2021).

[8] INCIBE (Instituto Nacional de Ciberseguridad). *Las 7 fases de un ciberataque. ¿Las conoces?* URL: https://www.incibe.es/protege-tu-empresa/blog/las-7-fases-ciberataque-las-conoces (visitado 16-03-2021).

[9] INCIBE (Instituto Nacional de Ciberseguridad). *Methbot*. URL: https://www.incibe-cert.es/alerta-temprana/bitacora-ciberseguridad/methbot-mayor-y-mas-rentable-operacion-fraude-anuncios (visitado 04-03-2021).

[10] INCIBE (Instituto Nacional de Ciberseguridad). *Qué es una botnet y cómo saber si tu empresa forma parte de ella*. URL: https://www.incibe.es/protege-tu-empresa/blog/botnet-y-saber-si-tu-empresa-forma-parte-ella (visitado 04-03-2021).

[11] Alex Clark. *Pillow*. URL: https://pypi.org/project/Pillow (visitado 14-04-2021).

[12] Steve Coast. *OpenStreetMap*. URL: https://www.openstreetmap.org/about (visitado 14-06-2021).

[13] The MITRE Corporation. *MITRE*. URL: https://www.mitre.org (visitado 17-03-2021).

[14] The MITRE Corporation. *MITRE ATT&CK®*. URL: https://attack.mitre.org/ (visitado 17-03-2021).

[15] The MITRE Corporation. *MITRE ATT&CK® Navigator*. URL: https://mitre-attack.github.io/attack-navigator (visitado 17-03-2021).

[16] Exceltic. *Introducción a Scrum... en menos de 5 minutos*. URL: https://www.youtube.com/watch?v=P25JP0u6UKw (visitado 09-03-2021).

[17] F-Secure. *PrettyPark*. URL: https://www.f-secure.com/v-descs/prettyp.shtml (visitado 04-03-2021).

[18] F-Secure. *SubSeven*. URL: https://www.f-secure.com/v-descs/subseven.shtml (visitado 04-03-2021).

[19] F-Secure. *Zbot*. URL: https://www.f-secure.com/v-descs/trojan-spy_w32_zbot.shtml (visitado 04-03-2021).

[20] Ed. G. Camarillo. *Peer-to-Peer (P2P) Architecture*. URL: https://datatracker.ietf.org/doc/html/rfc5694 (visitado 04-03-2021).

[21] GitHub. *MIT License*. URL: https://choosealicense.com/licenses/mit (visitado 05-07-2021).

[22] Hartmut Goebel y col. *PyInstaller*. URL: https://pypi.org/project/pyinstaller (visitado 12-04-2021).

[23] Google. *Protocol Buffers*. URL: https://developers.google.com/protocol-buffers (visitado 12-04-2021).

[24] Junio Hamano y Linus Torvalds. *Git*. URL: https://git-scm.com (visitado 01-03-2021).

[25] John Hammersley y John Lees-Miller. *Overleaf*. URL: https://www.overleaf.com (visitado 01-03-2021).

[26] D. Richard Hipp. *SQLite*. URL: https://www.sqlite.org (visitado 07-06-2021).

[27] Bob Hughes y Mike Cotterell. *Software Project Management*. 5a edición. Europa, Oriente Medio y Africa: McGraw-Hill Education, 2009. ISBN: 9780077122799. (Visitado 14-03-2021).

[28] IANA. *Time Zone Database*. URL: https://www.iana.org/time-zones (visitado 18-04-2021).

[29] GitLab Inc. *GitLab*. URL: https://about.gitlab.com (visitado 01-03-2021).

[30] INDEED. *Salario de un programador junior*. URL: https://es.indeed.com/career/programador-junior/salaries (visitado 12-03-2021).

[31] Escuela de Ingeniería Informática - Universidad de Valladolid. *GitLab Escuela de Ingeniería Informática*. URL: https://gitlab.inf.uva.es (visitado 01-03-2021).

[32] Escuela de Ingeniería Informática - Universidad de Valladolid. *Guía docente de la asignatura - TRABAJO DE FIN DE GRADO (MENCIÓN TECNOLOGÍAS DE LA INFORMACIÓN)*. URL: https://alberguweb1.uva.es/guia_docente/uploads/2021/545/46977/1/Documento.pdf (visitado 07-03-2021).

[33] KAFEINE. *Smominru Monero mining botnet making millions for operators*. URL: https://www.proofpoint.com/us/threat-insight/post/smominru-monero-mining-botnet-making-millions-operators (visitado 04-03-2021).

[34] Leslie Lamport. *LaTeX*. URL: https://www.latex-project.org (visitado 01-03-2021).

[35] Knud Lasse Lueth. *State of the IoT 2020*. URL: https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time (visitado 04-03-2021).

[36] Santiago Millán y Marimar Jiménez. *Telefónica sufre un 'hackeo' masivo y el CNI hace saltar las alarmas: cómo y a quién afecta*. URL: https://cincodias.elpais.com/cincodias/2017/05/12/companias/1494585502_908236.html (visitado 05-03-2021).

[37] Natalia Andrea Molano. *Claves para entender la tecnología 'blockchain'*. URL: https://www.bbva.com/es/claves-para-entender-la-tecnologia-blockchain (visitado 16-03-2021).

[38] Jarkko Oikarinen. *Internet Relay Chat Protocol*. URL: https://datatracker.ietf.org/doc/html/rfc1459 (visitado 04-03-2021).

[39] Mauricio Piacentini. *DB Browser*. URL: https://sqlitebrowser.org (visitado 07-06-2021).

[40] Kenneth Reitz. *Requests*. URL: https://pypi.org/project/requests (visitado 10-05-2021).

[41] Armin Ronacher. *Flask*. URL: https://flask.palletsprojects.com (visitado 10-05-2021).

[42] Armin Ronacher. *Jinja2*. URL: https://jinja.palletsprojects.com (visitado 10-05-2021).

[43] Guido van Rossum. *Python*. URL: https://www.python.org (visitado 01-03-2021).

[44] Evyatar Saias. *Bitcoins blockchains and botnets*. URL: https://blogs.akamai.com/sitr/2021/02/bitcoins-blockchains-and-botnets.html (visitado 16-03-2021).

[45] Ken Schwaber y Jeff Sutherland. *La Guía de Scrum*. URL: https://scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-Spanish.pdf (visitado 09-03-2021).

[46] E. U. Informática en Segovia - Departamento de Informática - Universidad de Valladolid. *Tema 2: El modelo cliente/servidor*. URL: https://www.infor.uva.es/~fdiaz/sd/2005_06/doc/SD_TE02_20060305.pdf (visitado 02-03-2021).

[47] OSI (Oficina de Seguridad del Internauta). *Botnet: Mirai*. URL: https://www.osi.es/es/servicio-antibotnet/info/mirai (visitado 04-03-2021).

[48] Al Sweigart. *PyScreeze*. URL: https://pypi.org/project/PyScreeze (visitado 14-04-2021).

[49] Trello. *Trello*. URL: https://trello.com (visitado 01-03-2021).

[50] Twitter. *Bootstrap*. URL: https://getbootstrap.com (visitado 10-05-2021).

[51] wenzhixin. *Bootstrap Table*. URL: https://bootstrap-table.com (visitado 10-05-2021).

[52] Nicky Woolf. *DDoS attack that disrupted internet was largest of its kind in history, experts say*. URL: https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet (visitado 05-03-2021).