GRADO EN INGENIERÍA DEL SOFTWARE

CIBERCANARIO: UN HONEYPOT BÁSICO, TANGIBLE Y USABLE PARA ENTORNOS IOT

CYBERCANARY: A BASIC, TANGIBLE AND USABLE HONEYPOT FOR IOT ENVIRONMENTS

Realizado por

JUAN GARCÍA RUIZ

Tutorizado por

RODRIGO ROMÁN CASTRO

Departamento

LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

MÁLAGA, (SEPTIEMBRE DE 2020)

Resumen

En este proyecto se ha desarrollado un dispositivo honeypot de bajo presupuesto y fácil de usar para proteger a una red doméstica compuesta de dispositivos IoT (Internet de las Cosas).

En primer lugar, se ha realizado un estudio de la composición y de las vulnerabilidades que suelen tener este tipo de entornos. Teniendo en cuanta esto, se ha realizado un análisis de las tecnologías a usar para el desarrollo del dispositivo siguiendo, además del estudio anterior, criterios económicos y de seguridad.

La segunda parte, más extensa, abarca todo el proceso de diseño y desarrollo del dispositivo honeypot. El sistema se ha desarrollado en una Raspberry Pi 3 Modelo B conectada a una pantalla. Con respecto al software, se ha usado un Ubuntu Server como sistema operativo y para el desarrollo de los componentes, Python. Estos componentes son, por un lado, el software de detección de amenazas y por otro lado un señuelo que finja ser un dispositivo y sea detectado por el controlador Mozilla WebThings Gateway.

Palabras claves:

- Honeypot
- IoT
- Seguridad
- Mozilla WebThings Gateway

Abstract

The main goal of this project is to develop a low-cost and easy to use honeypot device to protect a domestic network compound of IoT (Internet of Things) devices.

Firstly, an evaluation about the composition and vulnerabilities that often appears in these environments was done. Taking this into consideration, it was done an analysis of the technologies that were going to be use for the development of the device following, apart from the previous evaluation, economic and security policies.

The second part, more extensive than the previous one, covers the whole design and development process of the honeypot device. The system has been developed in a Raspberry Pi 3 Model B connected to a screen. Regarding the software, Ubuntu Server has been used as the operating system and for the component development, Python. These components are, partly, the threats detection software and, on the other hand, a decoy that simulates being a device connected to the Mozilla WebThings Gateway controller.

Keywords:

- Honeypot
- IoT
- Security
- Mozilla WebThings Gateway

Acrónimos

6LoWPAN	IPv6 over Low power Wireless Personal Area Network
API	Application Programming Interface
ARM	Advanced RISC Machine
BLE	Bluetooth Low Energy
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
FTP	File Transfer Protocol
GPS	Global Positioning System
HTTP	HyperText Transfer Protocol
ICMP	Internet Control Message Protocol
INCIBE	Instituto Nacional de Ciberseguridad
IoT	Internet of Things
IP	Internet Protocol
JTAG	Joint Test Action Group
LoRa	Long Range
MIPS	Microprocessor without Interlocked Pipelined Stages
MQTT	Message Queuing Telemetry Transport
RAM	Random Access Memory

REST	Representational State Transfer
SDK	Software Development Kit
SNMP	Simple Network Management Protocol
SO	Sistema Operativo
SOAP	Simple Object Access Protocol
SSH	Secure Shell
TCP	Transmission Control Protocol
UDP	User Data Protocol

Contenido

1

Introducción

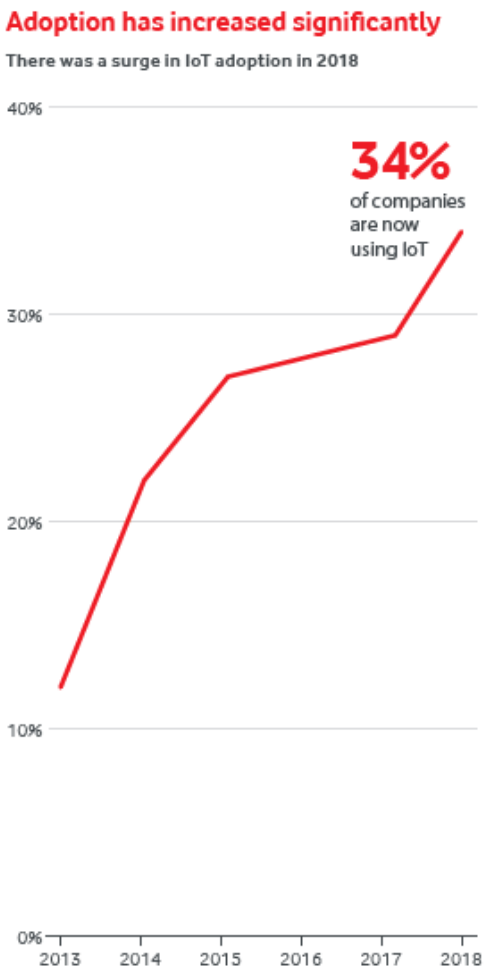

FIGURA 1: AUMENTO DEL USO DE LAS IOT EN LAS EMPRESAS

1. Motivaciones

El uso de las IoT, que había ido en aumento progresivamente durante la última década, tuvo un considerable repunte en 2018. Según el *2019 Vodafone IoT Barometer* **[1]**, "*más de un tercio (34%) de las empresas están usando actualmente IoT, frente al 29% del previo IoT Barometer*". Y de esas empresas, "*el 95% ya han detectado beneficios procedentes de los IoT. Y más de la mitad afirman que esos beneficios son significantes*". Además, "*el 55% de los innovadores afirman que las IoT ya han alterado su industria. Y el 74% que en 5 años las compañías que no estén usando aun IoT se quedarán atrasadas con respecto a sus competidores como consecuencia*".

Pero los IoT no solo se encuentran en las empresas, este aumento del uso de las IoT aplicadas a la industria también se ven reflejadas en aquellas usadas por los

consumidores (*consumer IoT*). Las IoT en ambas vertientes difieren en el tipo de dispositivo/aplicación y sus objetivos; sin embargo, algunas de las tecnologías utilizadas (p.ej. MQTT, servicios REST) son iguales, y los casos de uso guardan ciertas similitudes. Por ejemplo, un consumidor podría querer tener localizada a su mascota en todo momento de la manera en la que una compañía de taxis querría hacerlo con sus vehículos.

Este caso concreto puede explicar lo que se ha comentado con anterioridad al ejemplo. Son distintos tantos los dispositivos (necesitarán ser más preciso en el caso de los taxis para que a altas velocidades no se pierda la señal) como la aplicación (mientras que, para el escenario de la mascota, el número de localizadores a rastrear será bajo, en el de los taxis lo más seguro es que haya decenas como mínimo). De la misma manera los objetivos son distintos, en el primero se trata de controlar la ubicación de un animal para que no se pierda, mientras que en el segundo se realiza para tener controlados a los empleados de la compañía. Sin embargo, a pesar de estas diferencias, gran parte de las tecnologías usadas son iguales (GPS, por ejemplo) y, aunque los objetivos sean distintos, los casos de uso (cómo se localiza) son muy similares.

Y esto es solo un ejemplo, ya que las funcionalidades y posibilidades para los usuarios son infinitas, desde la domótica hasta atención sanitaria por medio de los relojes inteligentes. Y es, además, una propuesta que atrae mucho a la gente, quizás por la idea de que, con esta tecnología, poco a poco esas casas inteligentes que aparecían en las películas de ciencia ficción estén cada vez más cerca de ser una realidad.

Pero todos los innegables beneficios y las promesas aportadas por estos dispositivos tienen sus costes. La búsqueda del beneficio a toda costa, la necesidad de desarrollar nuevos productos rápidamente debido a que, al ser una tecnología joven, se encuentra en constante cambio y la escasa legislación existente sobre este tipo de productos son los principales causantes de una subestimación de la seguridad y del diseño de dispositivos IoT, que los vuelve potencialmente vulnerables permitiendo el acceso no autorizado con poco o ningún esfuerzo. Esto provoca, inevitablemente, la exposición de información personal sensible, desde el video de un monitor para bebés hasta contraseñas, emails y audios de juguetes IoT.

Con estas vulnerabilidades presentadas, esos dispositivos pensados en un inicio para hacer la vida más sencilla para el usuario final pueden convertirse en su adversario, lo que provoca una desconfianza general hacia ellos.

No ha sido hasta recientemente cuando se ha notado un considerable ascenso del número de trabajos en el campo de la seguridad para la IoT y, estos, suelen estar enfocados a las empresas y no al consumidor. Es por ello por lo que es necesario buscar soluciones que permitan a estos últimos detectar y, a ser posible, evitar esos problemas, permitiéndoles tener una barrera de defensa en sus hogares frente a ataques del exterior.

2. Estado del arte

Según el NIST **[2]**, el término IoT se refiere a los sistemas que involucran la computación, los sensores, la comunicación y la actuación. Los IoT buscan la conexión entre los humanos, objetos físicos y objetos electrónicos, permitiendo la monitorización, la automatización y la toma de decisiones.

No fue hasta 1999 cuando fue propuesto este término por Kevin Ashton, aunque se encuentran algunos ejemplos de este tipo de dispositivos con anterioridad a esa fecha. Por ejemplo, a principios de los años 80 un programador conectó a Internet un dispositivo refrigerador de la marca Coca Cola que se encontraba en la Universidad Carnegie Mellon para poder comprobar si había bebida disponible y, si era así, si estaba fría **[3]**. La aplicación fue un éxito entre los estudiantes de dicha universidad.

A pesar de este experimento llevado a cabo hace varias décadas, se trata de una tecnología relativamente novedosa ya que (como se puede observar en la *Figura 1*) no fue hasta 2013 cuando comenzó a usarse para aplicaciones industriales y desde entonces no ha dejado de crecer cada año con más rapidez que el anterior.

Y es esta *relativa novedad* uno de los culpables de que la seguridad se haya descuidado tanto. Este tipo de dispositivos son fácilmente accesible desde el exterior de la red y es por ello por lo que se precisan de soluciones para defender el entorno.

Un ejemplo de esta clase de soluciones son los honeypots. Un honeypot, o un sistema señuelo, es una herramienta útil tanto para detectar y analizar amenazas como para, incluso, mitigarlas. Los honeypots se muestran como víctimas reales para el atacante y, al recibir el correspondiente ataque, no solo son mínimas las consecuencias de este (al no haber ninguna infraestructura real tras ellos en la mayoría de los casos), sino que además puede recopilar información sobre el propio atacante como los comandos usados o, incluso su localización.

Sin embargo, casi todas las soluciones de este tipo están pensadas para la industria, al tratarse de dispositivos extremadamente complejos en algunos casos que simulan el funcionamiento de varios dispositivos virtuales detrás de una red virtual y que están económicamente fuera del alcance de un usuario final.

Hacen falta, por lo tanto, soluciones para los consumidores. Sería interesante disponer de un dispositivo honeypot básico, portable, tangible, barato y usable que permitiese a los usuarios finales descubrir que hay una potencial amenaza dentro de su red local.

3. Solución propuesta

Así pues, la aportación principal de este proyecto sería, siguiendo la estructura explicada arriba, crear un software *open source* sencillo y diseñado específicamente para este proyecto, el cual proporcione la funcionalidad de un honeypot básico, tangible y usable para un entorno IoT. Este software funcionaría sobre un hardware de bajo presupuesto (p.ej. ESP32, Raspberry Pi), y nos permitiría avisar al usuario de la existencia de una situación anómala en la red a la que está conectado el dispositivo.

El software de este dispositivo (que será implementado expresamente para este proyecto) se centrará en detectar un conjunto básico de ataques que representen a un adversario tratando de atacar una red IoT. Más concretamente, el software permitirá reconocer por un lado el escáner de puertos dentro de una red local, que se encarga de detectar el estado de los puertos (abierto, cerrado o protegido por firewall); y, por otro

lado, avisar cuando una IP sospechosa del exterior envíe paquetes que podrían resultar peligrosos.

4. Estructura del documento

Esta memoria está organizada en 5 capítulos quedando divididos de la siguiente manera:

- **Capítulo 1 - Introducción:** En este capítulo se realiza una introducción del documento, centrándose, especialmente, en las motivaciones y los objetivos del proyecto propuesto.
- **Capítulo 2 - Estudio preliminar:** En este capítulo se profundizará en conceptos relacionados con el dispositivo que se quiere desarrollar, centrándose especialmente en los honeypots y en los entornos IoT (funcionamiento y vulnerabilidades comunes en estos tipos de dispositivos).
- **Capítulo 3 - Tecnologías utilizadas:** En este capítulo se realizará la elección de las tecnologías que se usarán en este proyecto (hardware y software).
- **Capítulo 4 - Desarrollo:** En este capítulo se expone tanto la metodología usada como el propio proceso de desarrollo del producto con las decisiones tomadas durante el mismo, sus fallos y, consecuentemente, rectificaciones.
- **Capítulo 5 - Conclusiones y líneas futuras:** En este capítulo final se recopilarán las conclusiones alcanzadas a lo largo de todo el proceso del proyecto y una serie de posibles líneas para trabajos futuros.

Además de estos capítulos, al final de la memoria se incluye un manual de despliegue y de usuario en los apéndices, junto con un informe de pruebas.

2

Estudio preliminar

Como se ha adelantado en el capítulo anterior, se comenzará por realizar una investigación en profundidad de algunos aspectos de importancia que ayuden a comprender, no solo que son los honeypots, sino también cómo de imprescindibles son en los entornos IoT en la actualidad.

1. Honeypots. Definición, clasificación y solución propuesta

Para la realización de este estudio se comenzará primero por la base, por lo más básico, la definición de "honeypot". Según el INCIBE **[4]**, "un honeypot es una herramienta especialmente diseñada para servir como trampa contra posibles atacantes. Los honeypot tienen la capacidad de simular un servicio o un dispositivo con el objetivo de atraer hacia él acciones que posteriormente serán analizadas". Por lo tanto, el objetivo de estos dispositivos es el de engañan a los atacantes para que los ciberataques se realizan en vano y, de esta manera, impedir o mitigar ataques informáticos. Además de esto, el dispositivo podría ser capaz también de alertar ataques, obtener información de los atacantes o ralentizar ataques (según la complejidad del honeypot, algunos dispositivos pueden implementar uno o más de uno de estos objetivos).

Estos dispositivos se pueden clasificar además en base a diferentes criterios, los cuales nos permiten conocer detalles sobre su comportamiento o estructura real. Por ejemplo, se pueden clasificar en base a la interacción que tengan con las conexiones entrantes, dividiéndolos en:

- **Baja interacción.** Son honeypots que tienen respuesta básica a ciertos comandos, pero no profundizan más allá. Reproducen una parte limitada del comportamiento real del sistema. Un ejemplo sería un honeypot que simplemente presente un formulario de inicio de sesión para registrar qué usuarios y contraseñas prueban los atacantes como entrada. Su configuración es sencilla y su consumo de recursos bajo, pero es sencillo notar que no se trata de un dispositivo real.
- **Media interacción.** En estos honeypots crece la relación entre el atacante y el sistema incluyendo recursos falsos, como sistemas de ficheros o servidores de FTP o SSH. Pueden resultar útiles para capturar comandos que puedan ejecutar los atacantes, ver qué vulnerabilidades intentan explotar o qué malware intentan ejecutar.
- **Alta interacción.** Ofrecen un sistema con las mismas características que un dispositivo real, con acceso total al atacante, ya que las vulnerabilidades son completamente explotables. Se pueden recopilar más datos, pero también suponen un riesgo mayor, debido a que el sistema se puede ver comprometido.

También se pueden clasificar según la manera en la que esté implementado:

- **Simulados**. Son aquellos honeypots cuyo comportamiento imita al de un dispositivo real mediante recursos falsos, salidas preprogramadas o patrones definidos.
- **Emulados.** Cuentan con una plataforma que les proporciona una infraestructura virtual para que el honeypot se ejecute sobre ella, la cual es igual que la del dispositivo real. En ocasiones se usan como *sandbox*, pudiendo

ser fácilmente reseteados a un estado anterior si un atacante logra modificar partes importantes del sistema.

- **Dispositivos reales.** Se usa el hardware real como honeypot, estando disponible para el atacante el entorno al completo.

Como se puede observar por lo descrito hasta ahora, los honeypots pueden llegar a ser muy complejos. Se podría desarrollar para este proyecto un honeypot de alta iteración que no solo fuera capaz de avisar cuando se esté realizando un ataque, sino que incluso recopile información sobre el atacante. Pero aún con las posibilidades que nos podría proporcionar un dispositivo así, ¿por qué limitarse a ofrecer una solución sencilla en lugar de proponer una que, aunque más compleja, sea más completa?

Existen dos motivos principalmente, el primero de ellos es que una solución poco compleja permite poder desplegarla, con una configuración mínima, en un dispositivo con pocos recursos (incluso una Raspberry Pi conectada a una pantalla e-ink, o una ESP32) abaratando de esta manera el dispositivo en gran medida para que pueda estar al alcance económico de todos. El segundo motivo viene dado por el destinatario de esta solución. Al ser un dispositivo pensado para que sirva como primera línea de defensa para gente con escasos conocimientos de seguridad informática (o incluso ninguno), sería importante que cumpliera su función mientras fuera lo más simple posible para ser fácilmente configurable, desplegable, y usable.

Existe un tercer motivo de peso que engloba los dos anteriores. Existen muchas soluciones honeypot complejas para las empresas que ofrecen buenos resultados, pero no existe ninguna solución conocida que esté pensada y destinada para ese consumidor final que quiera tener, por ejemplo, el entorno IoT de su casa protegido de intrusiones no deseadas. Y es que, aunque el dispositivo esté pensado para entornos de este tipo, también estará pensado para poder integrarse en otras soluciones.

2. Entorno IoT

Para que un honeypot cumpla su objetivo de proteger otros dispositivos se debe encontrar en el mismo entorno que el resto. En el caso del *Cibercanario*, este se encontrará instalado en un entorno IoT, concretamente en un entorno a nivel doméstico.

El dispositivo, en esta red, tendrá que coexistir con otros muy distintos entre sí (cámaras, routers, wearables…) y que tendrán, por lo tanto, diversas respuestas a comandos de ataques. Esta heterogeneidad del escenario IoT es una de las primeras complicaciones que se encuentran y no controlarla correctamente podría desencadenar en el riesgo de ser una solución demasiado específica, o quedar rápidamente obsoleta. Es por ello por lo que, para simular la pertenencia a esta red, la arquitectura software del *Cibercanario* incluirá una serie de componentes sencillos (desarrollados en C, Python…) los cuales se encargarán de simular pertenecer a la red de dispositivos IoT.

También debemos tener en cuenta de la posibilidad de la existencia de una plataforma que coordine los elementos de la red local, un ejemplo de esto es *OpenHAB*, que haría de controlador central.

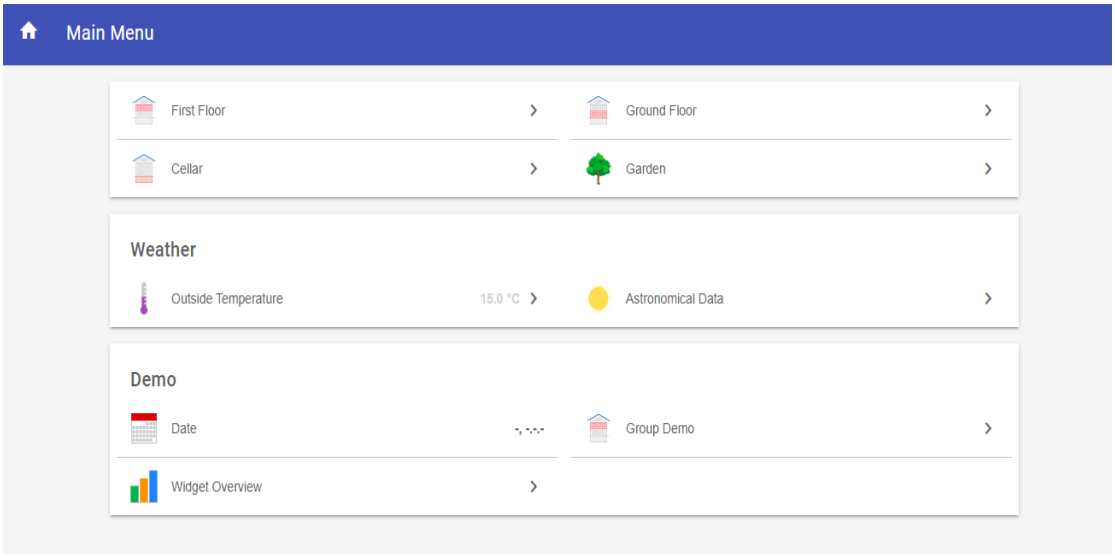

FIGURA 2: INTERFAZ PLATAFORMA OPENHAB

OpenHAB está especializado en sistemas domóticos y es por ello por lo que quizás sea la plataforma más común en los sistemas a los que enfocamos nuestro producto. Aun así, existen otras plataformas más generalistas y que, por lo tanto, muestran una información más amplia y detallada sobre los distintos dispositivos conectados en el entorno IoT.

Entre las plataformas de este tipo más destacables podemos encontrar:

- **Azure IoT Edge**

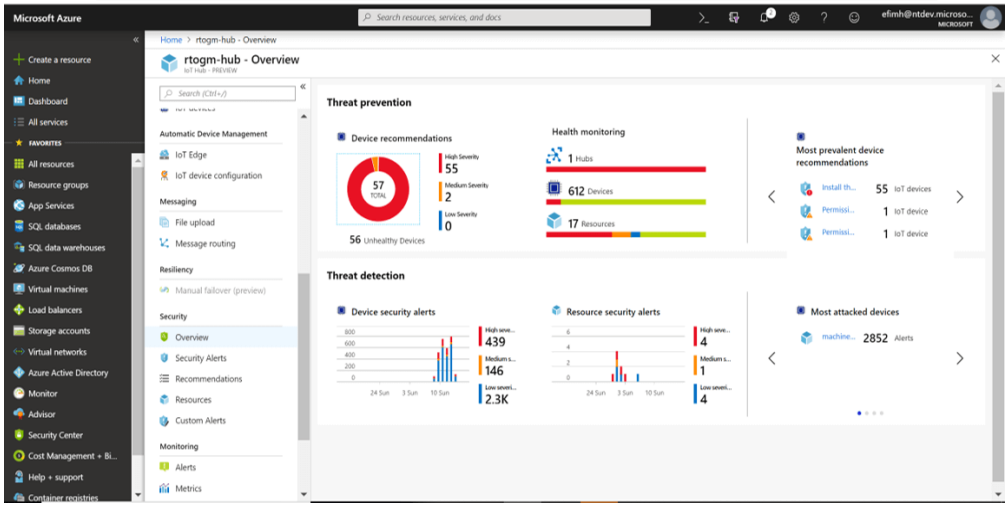

FIGURA 3: INTERFAZ PLATAFORMA AZURE IOT EDGE

- **Mainflux**

FIGURA 4: INTERFAZ PLATAFORMA MAINFLUX

- **Mozilla Webthings Gateway**

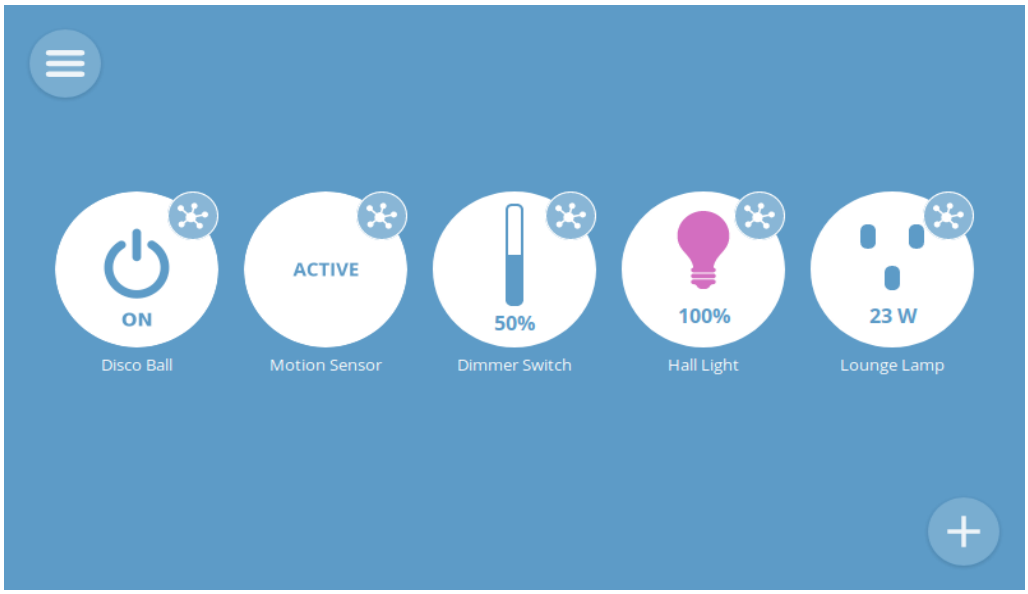

FIGURA 5: INTERFAZ PLATAFORMA MOZILLA WEBTHINGS

La heterogeneidad del escenario IoT expuesto con anterioridad implicará que se usarán varios protocolos durante las comunicaciones entre los dispositivos. Según el INCIBE **[5]**, el aumento del uso de dispositivos IoT en los hogares lleva a que más empresas desarrollen este tipo de productos (cada uno de ellos con sus propios protocolos) y aunque no se suele facilitar mucha información sobre este apartado, podemos clasificar los más importantes en dos apartados **[6]**:

- **Protocolos de comunicación:** Estos son los protocolos usados para conectar los dispositivos a través de la red. Existen varios protocolos como Bluetooth, ZigBee o HTTP. Este último es el más usado al ser el que mejor funciona cuando se quieren enviar grandes cantidades de datos; sin embargo, no es el más recomendable al consumir una gran cantidad de energía.
- **Protocolos de datos:** Se usan para conectar dispositivos IoT que no empleen mucha potencia. Estos protocolos proporcionan una conexión entre dispositivos sin necesidad de conectarse a internet, siendo MQTT el más conocido de este grupo. Está basado en la pila TCP/IP como base para su

comunicación y uno de los puntos positivos es que no se crea una conexión por cada comunicación, sino que se reutiliza la misma en todas. Al estar destinado para dispositivos que requieran de poca energía es el protocolo que usaremos para las comunicaciones con el controlador central.

Además de estas dos categorías existen otros protocolos más complejos que, en este caso, no se usarán en el dispositivo a desarrollar pero que son importantes conocer **[5]**:

- **AllJoyn** se trata de un estándar de código abierto y, por lo tanto, facilita en gran medida para todo tipo de protocolos localizados en la capa de transporte la comunicación entre dispositivos y aplicaciones.
- **HomePlug** y **HomeGrid** son protocolos que hacen uso de la red eléctrica para su comunicación.
- **MFi** (Made For iPhone/iPod/iPad) es un protocolo de comunicaciones de la propia compañía de Apple diseñado con el único objetivo de interactuar con los dispositivos de esta marca.
- **OCF** (Open Connectivity Foundation) es un proyecto de código abierto que pretende garantizar la interoperabilidad de millones de dispositivos, por medio de una implementación de referencia (IoTivity) y un programa de certificación.
- **Thread** (network protocol) es una tecnología basada en las comunicaciones por red mediante IPv6 usando el cifrado AES. Debido a esto y por la flexibilidad que ofrece, es un protocolo muy seguro y está preparado para el futuro.

3. Anomalía

Se ha definido lo que es un honeypot y explicado cómo será y funcionará el entorno en el que se encuentre y deba proteger. Sin embargo, ¿de qué lo tiene que proteger? ¿cómo es esta amenaza?

Cuando un elemento externo/interno ejecuta un ataque sobre un entorno IoT surgirá una anomalía. Para poder anticiparse a los ataques que se van a realizar, es esencial ponerse en el lugar del atacante: saber cómo trabajan, qué pasos siguen y, sobre todo, sería correcto conocer de antemano los ataques más usados en este tipo de entornos.

Se pueden encontrar diferentes tipos de ataques (que se estudiarán más adelante), pero los primeros pasos son comunes en todos ellos. Según el libro *The IoT Hacker's Handbook* **[7]**, lo primero que deberíamos tener en cuenta es obtener toda la información necesaria sobre el elemento objetivo ya que "*comenzar una evaluación con información incompleta o parcial es uno de los mayores errores que un pentester puede cometer*" (esto se puede realizar por medio de manuales de usuarios, documentación, recursos online...) y, tras esto, realizar un "*test de penetración*". Este estaría compuesto por dos partes: "*hacer un mapa de toda la superficie de ataque*" e "*identificar vulnerabilidades*". Este paso es muy importante ya que permitirá entender la arquitectura del elemento y, al mismo tiempo, priorizar los ataques con más probabilidades de éxito.

Antes de continuar, hay que fijarse en la arquitectura de los dispositivos. Aunque los dispositivos IoT son muy diversos suelen presentar una arquitectura bastante parecida la cual puede ser dividida, según el libro *The IoT Hacker's Handbook* **[7]**, en tres partes fácilmente diferenciables, cada una con sus vulnerabilidades características:

Dispositivo empotrado

Este dispositivo puede tener diferentes finalidades dependiendo del escenario del caso de uso. Se puede usar como un hub para toda la arquitectura IoT del dispositivo, puede servir como el sensor que recoge datos de su entorno físico o como un medio para mostrar los datos o realizar acciones planeadas por el usuario.

Aun con los diferentes propósitos que puede tener este elemento, la aproximación para atacar estos dispositivos suele ser bastante parecida. Sin embargo, lo que sí

depende de dichos propósitos es cómo de sensible es la información que contiene y, por lo tanto, como de crítico sería que se comprometa.

Algunas de las vulnerabilidades encontradas en estos dispositivos son:

- *Puertos en serie expuestos.*
- *Mecanismos de autenticación inseguros usados en los puertos en serie.*
- *La habilidad de copiar el firmware por medio del JTAG o los chips Flash.*
- *Ataques externos "media-based".*
- *Análisis de potencia y ataques de canal lateral.*

Firmware, software y aplicaciones

Tras observar la posible explotación del hardware, hay que centrarse ahora en lo relacionado con la parte software de un IoT. Esta sección es muy amplia y abarca desde el firmware que ejecuta el dispositivo, hasta las aplicaciones móviles que se usan para controlarlo, los componentes en la nube conectados al dispositivo…

Estos son los componentes de los IoT en los que se pueden aplicar los métodos tradicionales de pentesting, incluyendo puntos como la ingeniería inversa de diferentes arquitecturas como ARM y MIPS o de las aplicaciones móviles.

También sería interesante centrarse en la ingeniería inversa de la API de comunicación que permita entender cómo interaccionan los diferentes componentes IoT entre ellos (y así descubrir qué protocolos se usan).

Pero, como se ha comentado con anterioridad, esta sección es muy amplia y, por ello, es importante observar más en profundidad qué componentes de los dispositivos IoT se incluyen:

- **Aplicación móvil:** Esta es la que permite controlar los dispositivos inteligentes. Este puede ser un punto de acceso para atacar el componente web por medio de ingeniería inversa al código binario de la aplicación o a las APIs de comunicación.

- **Panel de control web:** Permite al usuario monitorizar el dispositivo, ver análisis e información de uso, controlar permisos… La mayoría de los dispositivos dispondrán de una interfaz web y si esta es vulnerable, podría permitir el acceso a datos no autorizados (esto ha sido un problema muy importante con los monitores de bebés).
- **Interfaces de red no seguras:** Estos son los componentes del dispositivo IoT que están expuestos en la red y pueden ser comprometidos debido a las vulnerabilidades en la expuesta interfaz de red. Esto puede incluir desde un puerto expuesto que acepta alguna conexión sin ningún tipo de autenticación o algún servicio que esté funcionando con una versión desactualizada (y, con mucha seguridad, vulnerable) de algún componente como SNMP, FTP…
- **Firmware:** Este se encarga de controlar los diversos componentes del dispositivo y es el responsable de toda acción en él. El firmware es una pieza binaria de datos la cual, abierta en un visor hexadecimal, puede mostrar las secciones de este. El sistema de archivos, probablemente la parte que más nos interesa en el firmware, puede ser de distintos tipos según los requerimientos del fabricante y la funcionalidad del dispositivo. Esto implica que cada tipo de sistema de archivos tiene su propia cabecera que se usa después para identificar dónde comienza el sistema de archivos en el código binario del firmware.

Las aplicaciones móviles, aplicaciones web y servicios empotrados suelen comunicarse con otros componentes por diversos mecanismos de comunicación como REST, SOAP, MQTT, CoAP…

Estos componentes pueden presentar ciertas vulnerabilidades. En el firmware, por ejemplo, encontramos:

- *Posibilidad de modificar el firmware.*
- *Firmas no seguras y verificación de integridad.*
- *Hardcode con valores sensibles en el firmware (API keys, contraseñas...).*

- *Certificados privados.*
- *Entender todo el funcionamiento del dispositivo por medio del firmware.*
- *Extracción de archivos desde el firmware.*
- *Componentes antiguos con vulnerabilidades conocidas.*

En las aplicaciones móviles, por otro lado, encontramos:

- *Ingeniería inversa de la aplicación móvil.*
- *Copia del código fuente de la aplicación móvil.*
- *Autenticaciones no seguras y comprobaciones de autenticación.*
- *Fallas empresariales y lógicas.*
- *Filtraciones de datos de canal lateral.*
- *Ataques de manipulación del tiempo de ejecución.*
- *Comunicaciones no seguras por red.*
- *Bibliotecas y SDKs de terceros desactualizadas.*

Por último, en las aplicaciones web:

- *Inyecciones desde el lado del cliente.*
- *Referencias a objeto directo no seguras.*
- *Autenticación y autorización no seguras.*
- *Pérdida de datos sensibles*
- *Fallas empresariales lógicas.*
- *Falsificación de peticiones cross-site.*
- *Cross-site scripting*

Radiocomunicación

La radiocomunicación proporciona un método para diferentes dispositivos para comunicarse entre sí. Estos protocolos de comunicaciones no suelen ser considerados

por las compañías cuando se piensa sobre seguridad y, por lo tanto, es un punto prometedor para los pentesters a la hora de buscar vulnerabilidades en este tipo de dispositivos.

Los protocolos de radiocomunicación más usados por los dispositivos IoT son Wi-Fi, BLE, ZigBee, Wave, 6LoWPAN, LoRa...

Dependiendo del componente radio, tendrá diversas vulnerabilidades. Sin embargo, podemos destacar unas vulnerabilidades comunes en los diversos protocolos de radiocomunicación:

- *Ataques Man-in-the-middle.*
- *Ataques basados en repeticiones.*
- *Verificación CRC.*
- *Ataques basados en embotellamiento.*
- *Denegación de servicio (DoS).*
- *Falta de encriptación.*
- *Extracción de información sensible de paquetes de radio.*
- *Intercepción y modificación de comunicaciones por radio en tiempo real.*

3

Tecnologías utilizadas

Teniendo en cuenta, en parte, lo expuesto en el anterior capítulo, en este se tratará de escoger las tecnologías que más se adapten al producto a desarrollar. Se centrará tanto en el apartado del hardware como en el software.

1. Hardware

Como se ha comentado, se necesitará de una solución sencilla y destinada a un público que busque no gastar una gran cantidad de dinero en ella. Por lo tanto, el hardware necesitará ser lo más barato posible. Se puede encontrar varios candidatos que cumplan con estos requisitos, pero se compararán los más destacados dentro de este grupo: *ESP32*, *Raspberry*, *Rock64 Media Board*, *Orange Pi*, *BeagleBone* y *Onion Omega*. La diferencia entre los distintos periféricos se puede observar mejor en la siguiente tabla **[8]**:

	ESP32	Raspberry Pi 4	ROCK64	BeagleBone Black	Orange Pi 4B	Onion Omega2+
CPU	Dual-Core Hasta 240 MHz Hasta 600 MIPS	Quad-Core 1.5GHz	Quad-Core 1.5GHz	1GHz 2000 MIPS	6-Core Hasta 2.0GHz	580MHz
RAM	520 KiB (SRAM)	4GB	4GB	512MB	4GB	128MB
Precio	10€	60€	45€	50€	65€	13€

FIGURA 6: COMPARACIÓN HARDWARE

Como se puede observar, son dispositivos muy diferentes tanto en tecnología como en precio. Sin embargo, para realizar una elección lo más correcta posible se debería buscar la opción que mejor se adapte a lo que se quiere desarrollar y, al mismo tiempo, sea lo más barato y sencillo posible.

El *Cibercanario* debería comportarse como un IoT y, por lo tanto, su hardware debería también ser similar en potencia. A la hora de elegir el dispositivo que usar habría que centrarse en los siguientes puntos:

- **Simple:** Concretamente el dispositivo a desarrollar no necesita ser muy complejo, ya que trabajará como un sensor (no tiene que reproducir audio ni video en alta calidad, por ejemplo). Por lo que no nos haría falta un dispositivo con una gran CPU o RAM.
- **Bajo consumo de energía:** Además del gasto al comprar el dispositivo, es importante también asegurarse de que el gasto sea también el menor posible mientras esté funcionando y esto pasa por un bajo consumo energético. De primeras, cuanto menor sean las especificaciones, menor será el consumo. Además de esto, algunos dispositivos presentan ciertas especificaciones con este punto en mente como es el caso del ESP32 que

tiene un *ultra low power co-processor* que permite realizar ciertas acciones mientras se encuentra suspendido.

- **Seguro:** Es importante asegurarse de que el dispositivo hardware no presente vulnerabilidades conocidas. Debido a que el dispositivo está pensado para servir como una primera barrera contra ataques, es importante que, al menos, no sea fácilmente comprometible.

Por lo tanto, observando esto, se puede llegar a la conclusión de que no necesitamos tampoco un dispositivo muy potente y es probable que el *ESP32* o el *Onion Omega2+* sean las mejores opciones para llevar a cabo este proyecto.

Sin embargo, y haciendo referencia al último punto de los expuestos arriba, elegir la *ESP32* no sería la mejor opción al haberse encontrado a principios de este año una vulnerabilidad en este tipo de dispositivos **[9]**.

Aun así, debido a las limitaciones causadas por el confinamiento a causa del COVID-19, para el proyecto usaremos una *Raspberry Pi 3B* (que es el dispositivo que se encontraba disponible durante el desarrollo).

2. Software

Elegido ya el hardware que usaremos como dispositivo que alojará el *Cibercanario*, es el momento de centrarse en el apartado del software. Para esta sección, primero se estudiará que sistema operativo se adapta más al dispositivo a desarrollar y, posteriormente, en cómo se desarrollará el software de la aplicación que se encargará principalmente de, por un lado, detectar los ataques provenientes del exterior y, por otro, simular ser un dispositivo más del entorno IoT en el que se encuentra instalado.

a. Sistema Operativo

Como ya se ha comentado anteriormente se debe tratar de un dispositivo simple, pero esta palabra es muy genérica y sería necesario especificar con más detalle. Principalmente para que un dispositivo como el de este proyecto se considere simple debe ser capaz de durar mucho tiempo, siendo lo menos vulnerable posible y evitando por todos los medios que quede rápidamente obsoleto. Es decir, se puede decir que para que el Cibercanario sea considerado simple debe cumplir dos requisitos: robustez frente a ataques y la capacidad de funcionar a pesar de las actualizaciones del SO.

El SO que tenga el dispositivo deberá actualizarse continuamente para resolver vulnerabilidades que puedan surgir en él (por el mismo motivo que el que se explicó en el anterior apartado del hardware) y para mantener las bibliotecas actualizadas. Muchos SO permiten al dispositivo actualizarse automáticamente sin que el cliente tenga que hacer nada.

Pero para ello, primero habría que elegir un adecuado sistema operativo para el dispositivo. Para reducir el número de opciones nos fijaremos en sistemas operativos de la familia Linux los cuales, al ser tan variados, será más fácil encontrar alguno que se adapte a las necesidades del dispositivo a desarrollar.

Por ejemplo, Kali Linux ofrece una opción fácil y rápida de convertir el dispositivo en el que esté instalado el SO en un honeypot. Sin embargo, esta opción no permite poder configurar un honeypot desde 0 usando esta opción (hay que recordar que debe ser lo más sencillo posible) y, además, Kali Linux es un SO bastante poco estable lo que podría afectar a su durabilidad.

También hay otras opciones más comunes como Ubuntu. Pero, este SO tiene de base muchas más funcionalidades instaladas de las que necesitamos. Quizás algo más personificable como Arch Linux podría ser la solución que más encajaría. Pero debería existir algo más acorde y menos complejo de instalar.

Finalmente encontramos tres opciones muy prometedoras. Por un lado, Raspbian, un sistema operativo basado en Debian creado para la Raspberry Pi. Aunque, como Raspberry es un dispositivo pensado para cumplir casi cualquier funcionalidad, de nuevo incluye una gran cantidad de software preinstalado que no necesitamos. La segunda es Ubuntu Server, la cual, aunque está pensada para albergar servidores, está adaptada para instalarse en una Raspberry y presta especial atención a la seguridad. La tercera de estas opciones es Ubuntu Core, un sistema basado en Ubuntu y pensado para el mundo IoT, centrándose especialmente en la seguridad (cumpliendo así la premisa de que el dispositivo fuera lo menos vulnerable posible) y, al estar pensado para este tipo de dispositivos, tendrá un SO acorde a lo que queremos desarrollar.

Esta última opción parece, con claridad, la mejor de las expuestas así y es la que, al menos en un inicio, se iba a usar para nuestro dispositivo. Sin embargo, a la hora de realizar la instalación ha presentado muchos problemas. Para poder instalar el Cibercanario en este SO era necesario subir la aplicación a la tienda de Snapcraft **[10]** y, posteriormente, crear una imagen de Ubuntu Core para instalarla en el dispositivo. Tras dedicarle varios días no se ha podido encontrar la manera de subirla a Snapcraft por lo que está solución, a pesar de ser muy prometedora, se ha descartado finalmente.

Por lo tanto, se usará la segunda opción de las tres últimas, Ubuntu Server.

b. Aplicación

Para el desarrollo de la aplicación se empleará Python. Más allá de la experiencia personal del alumno con este lenguaje de programación, Python es un lenguaje que aporta numerosos beneficios para el desarrollo de este tipo de aplicaciones:

Se trata de un lenguaje **multiplataforma** que nos permite usarlo en diferentes sistemas operativos, lo que ayuda a que el lenguaje no sea un requisito más para la elección realizada en la sección anterior.

Tiene un estilo **flexible** al aportar una gran cantidad de herramientas que permite esa flexibilidad en el código.

Existe una **gran base de usuarios** y, por lo tanto, una **gran abundancia de bibliotecas** que permite extender la funcionalidad sin importar el tipo de aplicación que se esté desarrollando.

Para poder realizar el desarrollo de los componentes se han empleado algunas bibliotecas de Python. Se ha intentado usar las menos posibles para impedir la existencia de vulnerabilidades por estas vías. En concreto se han empleado 4:

- **Tkinter [11]:** Se ha usado para el desarrollo de la interfaz gráfica de la aplicación.
- **IPy [12]:** Biblioteca que permite el manejo de direcciones IP.
- **psutil [13]:** Es el encargado de mostrar la información del sistema que se observa en la pantalla inicial del *Cibercanario*.
- **webthing [14]:** Paquete de Mozilla WebThings que permite el desarrollo de dispositivos IoT que puedan conectarse con esta plataforma.

Con respecto al controlador central, y de todas las opciones estudiadas en el segundo capítulo, se usará Mozilla WebThings Gateway **[15]**. Los motivos principales de esta elección es que se trata de un software open source que, además, está preparado para el consumer IoT resultando bastante sencillo de usar al tener una interfaz intuitiva y poco cargada información. También emplea por

debajo el protocolo abierto MQTT, que en el capítulo de estudio se comentó que sería el más indicado para el Cibercanario al consumir poco.

Además de lo anterior, se ha anunciado en septiembre de 2020 **[16]** que WebThings pasará a ser un proyecto open source independiente, cediendo Mozilla tanto el control de la plataforma como su responsabilidad a la comunidad. Esto nos garantiza que el proyecto tendrá continuidad durante los próximos años y seguirá actualizándose.

c. Detección

La aplicación estará compuesta por una serie de componentes. En este apartado se profundizará en los detectores y se comentará un detalle del señuelo, mientras que el resto de ellos se analizarán en los capítulos siguientes.

Estos detectores se encargarán de detectar ataques comunes en este tipo de dispositivos. Uno de ellos detectará escaneos de puertos que se produzcan dentro de la red local. El escaneo de puertos suelen ser siempre el primero paso en un ataque **[17]**, es por ello por lo que es esencial que sean detectados por el dispositivo para poder controlar la intrusión en el entorno desde el primer momento.

Por otro lado, el Cibercanario será capaz también de detectar paquetes TCP, UDP e ICMP que puedan suponer una amenaza para el entorno. Este detector, a diferencia del anterior, estará pensado para detectar los paquetes que puedan suponer una amenaza y que provengan del exterior de la red local. Esto ayuda a detectar ataques a nivel de aplicación, por medio de paquetes a este nivel (p.ej. MQTT).

Para controlar, además, que el dispositivo pueda obviar los paquetes provenientes de ciertas dirección IP (al considerarse que no son paquetes que supongan una amenaza), se ha introducido una lista blanca. Esta lista en un inicio contendrá ya algunas direcciones IP (sobre todo de compañías telefónicas

españolas y de Ubuntu) que suelen enviar paquetes periódicamente con el fin, casi en todos los casos, de comprobar si el dispositivo está conectado a la red.

Además, y esto se comentará con más detalle en los próximos capítulos, se añadirá una funcionalidad para que el propio usuario pueda añadir nuevas direcciones IP a la lista si así lo ve conveniente.

Hay que indicar también que, al dedicarse el Cibercanario de detectar ataques en remoto o en local a través de las comunicaciones, los ataques a nivel de hardware no se aplican.

Finalmente, con respecto al señuelo, este será un elemento que servirá como distracción para el atacante para que no detecte que el Cibercanario es un honeypot. Tal y como se ha comentado se profundizará más en el capítulo siguiente. Sin embargo, para poder usar esta funcionalidad hará falta emplear un controlador central.

4

Desarrollo

En este cuarto capítulo se va a describir, inicialmente, la metodología que se ha seguido durante el proceso de desarrollo y los motivos de su elección. Posteriormente, se expondrá el propio proceso de desarrollo del producto con las decisiones tomadas durante el mismo, sus fallos y, consecuentemente, rectificaciones.

1. Metodología

Para el desarrollo de este proyecto se ha decidido seguir la metodología SCRUM. Se dividirá el desarrollo en iteraciones en las que se seleccionarán que requisitos implementar, se planificará la iteración y, finalmente, se realizará la implementación de lo planificado.

SCRUM es una metodología ágil; es decir, son metodologías que buscan adaptar la forma en la que se desarrolla un proyecto para conseguir una mayor flexibilidad. Esta flexibilidad se consigue, en primer lugar, dejando a un lado los métodos tradicionales de cascada los cuales conllevan muchos problemas. Al tratarse de un modelo lineal (en el que no se inicia una nueva etapa hasta que se ha terminado al completo la anterior), cualquier cambio durante el proceso o se descubre algún error en las fases tempranas del desarrollo, por pequeño que sea, puede causar una gran cantidad de problemas.

Ese es su principal problema, la falta de flexibilidad que esa imposibilidad de volver atrás provoca. Esta metodología funciona bien para proyectos que impliquen fabricar coches o edificios, por ejemplo, pero no es adecuada para el desarrollo de software, a no ser que el proyecto propuesto sea breve, simple y el equipo tenga experiencia en desarrollos similares.

Por lo tanto, es necesario que, para un proyecto como este, se emplee una metodología ágil. Existen varios motivos por los que se ha escogido SCRUM por encima de otras metodologías de este tipo, pero todas giran en torno a uno, la innovación. Y es que el dispositivo a desarrollar se trata de una solución novedosa para un público localizado y pensado para funcionar en un entorno de dispositivos IoT, considerado una tecnología bastante joven.

Esta innovación provoca que la flexibilidad sea una obligación en la metodología usada ya que los requisitos pueden ser cambiantes o estar poco definidos y el riesgo de cometer errores es considerablemente alto, por lo que es necesario que las consecuencias de estos fallos sea lo menor posible.

Esta metodología:

a. Determinar la lista de prioridades (**Product Backlog**).
b. Planificar la iteración (**Sprint Planning**) en la que se definen los objetivos que hay que alcanzar al final de esta, la fecha límite y el reparto de actividades más pequeñas entre los miembros del equipo, también conocido como **Sprint Backlog** (que en este caso no se realiza al ser individual).
c. Tras esto es el momento de realizar la implementación de los requisitos propuestos en la planificación de la iteración (**Sprint**).
d. Al acabar la iteración se realiza un análisis de esta (**Sprint Review**) en la que se comprueba si se han cumplido los objetivos. Esta es una reunión importante

porque según los resultados se puede cancelar la iteración si los resultados no son satisfactorios o actualizar la versión del producto.

e. Para cualquiera de los dos casos anteriores, se realiza tras eso una retrospectiva de la iteración (**Sprint Retrospective**) en la que se identifican elementos que fallaron durante la iteración junto con potenciales soluciones para esos problemas.

En el diagrama a continuación se puede observar ordenadamente los pasos explicados.

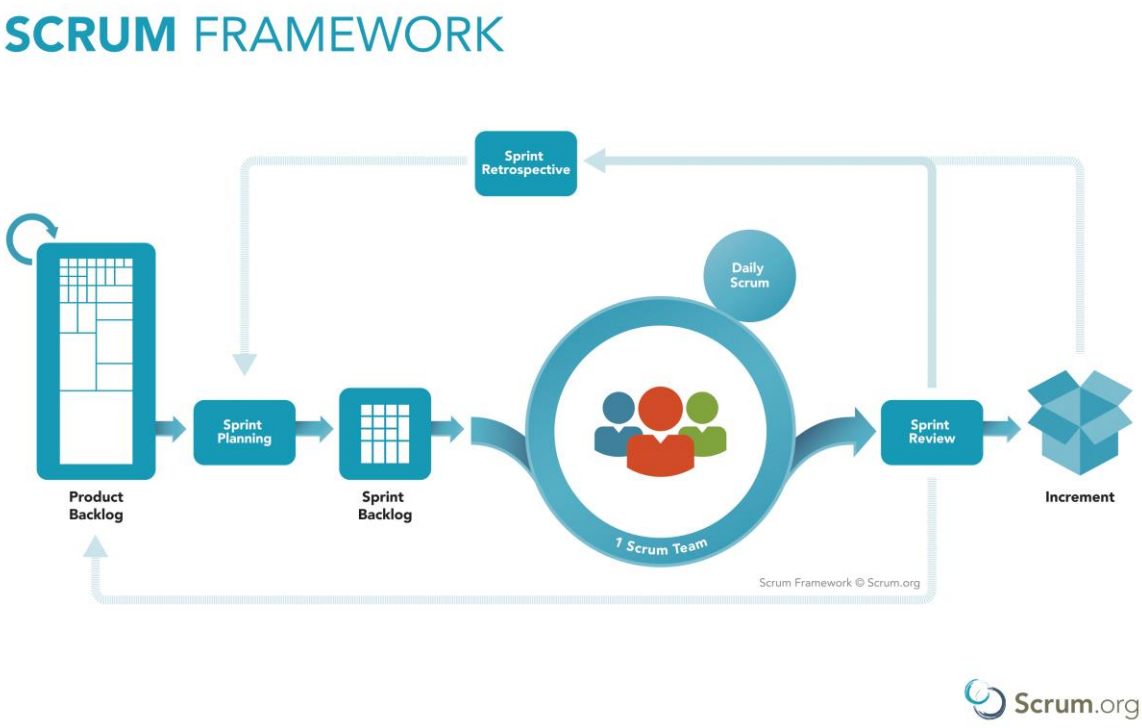

FIGURA 7: FASES SCRUM

2. Creación del *Cibercanario*

En este apartado se expondrán los pasos seguidos para el desarrollo del *Cibercanario* explicando el funcionamiento de este entre otros detalles del desarrollo.

a. Estudio

Es importante para proyectos en los que la innovación está presente que la primera fase sea de estudio. En el caso de este proyecto, esa fase, en un sentido más general, se ha desarrollado a lo largo del capítulo 2, en el que se ha profundizado en el concepto de honeypot, los diferentes tipos que existen y el motivo por el cual se ha decidido desarrollar un honeypot sencillo para usuarios con poco (o ningún) conocimiento en informática. Además de esto, se ha descrito la naturaleza de los entornos IoT, con especial atención a las diferentes piezas que componen estos dispositivos y a las vulnerabilidades que puedan estar presentes en ellos.

Por otro lado, en el capítulo 3 se ha hecho, apoyándose en las conclusiones obtenidas del estudio del capítulo 2, un análisis más específico en el que se ha tenido en cuenta el dispositivo a desarrollar, centrándose en qué tecnologías son las más apropiadas y en que bibliotecas se necesitan.

Teniendo esto en cuenta, no se va a realizar más comentarios al respecto de este apartado al ya haber sido analizado en profundidad en los capítulos anteriores.

b. Análisis de requisitos

Los requisitos a lo largo del proceso de desarrollo del dispositivo han cambiado, tal y como se esperaba desde el principio. Elegir una metodología ágil como SCRUM ha permitido que añadir o modificar requisitos ya existentes no hayan supuesto muchos contratiempos.

De hecho, al inicio del desarrollo los requisitos estaban muy poco definidos y eran muy generales. La mayoría, de hecho, estaban más centrados con el apartado no funcional que con el funcional: *el dispositivo debía ser robusto ante ataques, debía ser capaz de funcionar a pesar de las actualizaciones y debía simular su pertenencia al entorno en el que se encontrara instalado*. Estos

requisitos se han mantenido durante todo el proceso y han permanecido hasta el final.

Otro requisito que se tuvo en cuenta desde el inicio, aunque este no relacionado con el software sino con el montaje por parte del usuario final, es que debía ser fácil de instalar por parte del usuario final. Un usuario que lo más probable es que no tuviera muchos conocimientos en informática. Aunque este requisito pueda parecer algo que considerar para el final del desarrollo, lo cierto es que es importante tenerlo en cuenta desde el principio para evitar cualquier modificación considerable al final de la implementación por incumplirla.

Con respecto a los requisitos funcionales, en un inicio el único que existía era que el usuario tenía que ser capaz de visualizar algún tipo de aviso por pantalla cuando el *Cibercanario* recibiera algún tipo de ataque. A media que el proyecto ha ido creándose y tomando forma (ya que antes de la interfaz se implementó los detectores, pero de esto ya se hablará en su correspondiente sección) fueron apareciendo más requisitos funcionales, quedando el listado final de la siguiente manera:

- *El usuario podrá ver un aviso por pantalla cuando el dispositivo sea objetivo de un escaneo de puertos.*
- *El usuario podrá ver un aviso por pantalla cuando el dispositivo reciba paquetes de tipo TCP, UDP o ICMP proveniente de direcciones IP no conocidas.*
- *El usuario podrá ver la hora a la que se ha recibido el ataque.*
- *El usuario podrá observar el estado de las características del sistema (porcentaje de RAM y CPU usados).*
- *El usuario podrá elegir si cerrar las conexiones del dispositivo automáticamente después de recibir un ataque o mantenerlos abiertos.*

- *El usuario podrá añadir IPs que no supongan un riesgo a la lista blanca (se explicará el significado de este término en el apartado de diseño) del Cibercanario.*
- *El usuario podrá eliminar IPs de la lista blanca del Cibercanario.*

Además, y volviendo al tema de que los usuarios probablemente tengan escasos conocimientos de informática, estas acciones deben ser fáciles e intuitivas; es decir, la interfaz debe ser lo más simple posible. Y esto es algo que, al igual que pasaba con tener en cuenta que debía ser fácil de instalar para el usuario final, se debe tener en cuenta durante todo el proceso de desarrollo, especialmente en la parte de diseño e implementación.

Por ejemplo, para los requisitos de insertar o eliminar IPs de la *lista blanca*, en un principio se pensó en que el usuario podría hacerlo editando un archivo *txt*. Sin embargo, finalmente se decidió, para hacerlo más simple, añadir esta funcionalidad dentro de la propia aplicación para simplificar el proceso lo máximo posible.

3. Diseño

Esta sección se centrará en el diseño del dispositivo, concretamente en el diseño de la interfaz de la aplicación y en como esta se ha simplificado para que resulte más sencilla e intuitiva de utilizar para el usuario final.

A continuación, se muestra un boceto de la interfaz con transiciones entre los distintos estados:

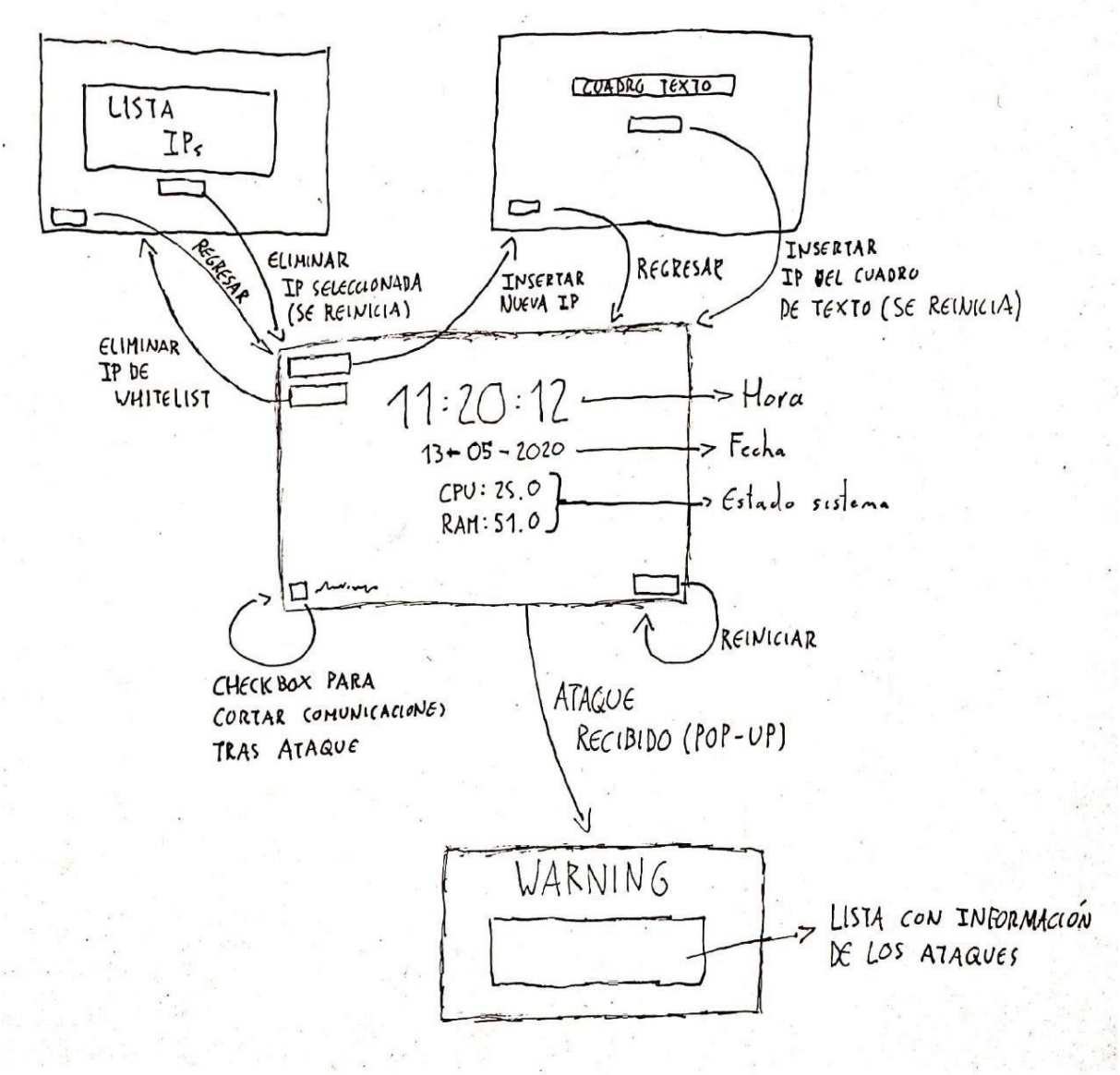

FIGURA 8: DISEÑO INTERFAZ *CIBERCANARIO*

Como se puede observar, la aplicación contará con una pantalla inicial en la que aparecerá información sobre la hora, fecha y datos del estado del sistema (CPU y RAM) junto con una serie de botones.

En la esquina superior izquierda, por ejemplo, estarán los botones para añadir y eliminar IPs de la *lista blanca* (funcionalidad que ya se nombró en el punto anterior). La *lista blanca* consiste en una lista de IPs "seguras" para que el dispositivo no las

considere como amenazas. De inicio esta lista ya incluye algunas direcciones relacionadas en su mayoría con compañías telefónicas españolas y direcciones que Ubuntu (como sistema operativo) usa para comprobar si el dispositivo sigue conectado a Internet. Aun así, estas funciones permiten al usuario añadir nuevas IPs, si así lo ve conveniente, o eliminar alguna ya existente en la lista. Después de cualquiera de las dos acciones, la aplicación se reiniciará para establecer los cambios. También se añade la opción de retroceder en caso de no querer hacer ninguna modificación.

En la esquina inferior izquierda, existe una casilla de verificación para indicar si se desea que se cierren todas las comunicaciones después de una ataque o si, por el contrario, se desea que continúen. En el primer caso y tras recibir el ataque, el dispositivo dejará de recibir comunicaciones del exterior y habría que reiniciarlo manualmente para volver a su estado anterior.

Por último, en la esquina inferior derecha encontramos el botón de reinicio. Este, como su nombre indica, reinicia la aplicación cerrando las comunicaciones (si no estaban cerradas ya) y abriéndolas de nuevo. La *lista blanca*, por supuesto, no se modificará por este reinicio.

Dejando a un lado el panel principal, en caso de recibir un ataque, aparecerá un pop-up. En éste se avisará que se ha recibido un ataque y se mostrará, además de la hora a la que se ha recibido, un listado con las amenazas detectadas e información sobre ellas. También, y solo en caso de haberse marcado la casilla de verificación antes mencionada, aparecerá un mensaje informando de que se han cerrado las comunicaciones.

4. Implementación

En este apartado, se profundizará en los distintos componentes que se han desarrollado para componer el software del dispositivo. Como en la sección anterior ya se ha hablado de la interfaz, en esta se intentará centrar especialmente en el programa detrás de ella, en los detectores, en el señuelo y en aquello que los conecta.

a. Detectores

Los detectores son aquello elementos que, por medio de los sockets, esperan y recogen cualquier mensaje que llegue del exterior (o desde dentro de la misma red) y los analiza para comprobar si son amenazas.

En este caso se desarrollará dos detectores que estarán funcionando en un mismo proceso y analizará cada uno los paquetes que el *Cibercanario* reciba que les corresponda según su procedencia. En el capítulo anterior se han comentado el funcionamiento de ambos, pero en este apartado se estudiará como se han desarrollado:

- **Escaneos de puertos dentro de la red local:** Para detectarlos habría que ser capaz de recoger una conexión que llegara a cualquiera de los puertos. En este escenario se podría detectar esas conexiones empleando ***socket.bind()***. Sin embargo, no se puede emplear esta función a varios puertos al mismo tiempo, por lo que (y aprovechando que se está utilizando una distribución de Linux) lo mejor para resolver este problema sería emplear ***iptables*** para redireccionar todos los puertos al que estará escuchando nuestro detector.

  Esta fue la opción definitiva desde el principio, sin embargo, a una semana de la entrega final dejó de funcionar. Al no poder detectarse el motivo, se decidió resolver de otra manera parecida a la que se explicará en el siguiente punto, esta consistía en detectar paquetes TCP provenientes de la red local que tuvieran la *Flag SYN* activada. El detector cuando recibe el primer paquete guarda la dirección en un listado de sospechosos junto al número de veces que ha enviado dicho paquete. Si en un corto espacio de tiempo recibe una gran cantidad de paquetes de estas características de la misma dirección, avisará de que se está recibiendo un escaneo de puertos.

- **Paquetes TCP/UDP/ICMP provenientes de fuera de la red local:** Muchos de estos paquetes que llegan al dispositivo no realizan una conexión completa, sino que es parcial, quedándose a nivel de *kernel* y no pudiéndose detectar con una función como ***socket.bind()***. Se puede, para esta situación emplear raw sockets, un tipo especial de sockets los cuales en lugar de escuchar paquetes de un protocolo concreto (TCP o UDP), lo que escucha son tramas IP. De esta manera una vez recibido el paquete, se puede comprobar que tipo de protocolo tiene, observar a que puerto estaba destinado y estudiar si puede tratarse de una amenaza.

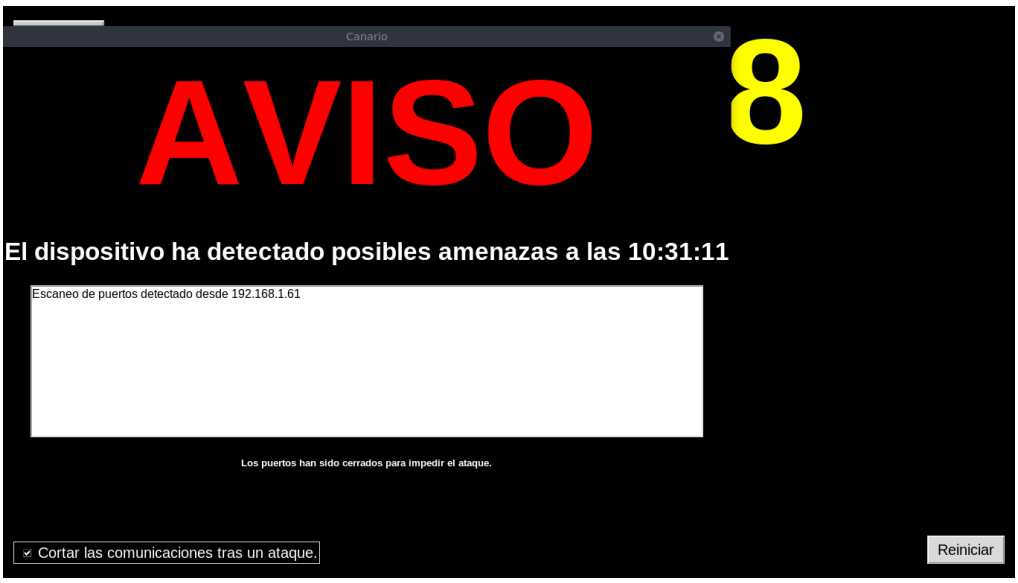

FIGURA 9: EJEMPLO DETECCIÓN AMENAZA

El empleo de la estrategia de los raw sockets permite expandir esta función y no limitarse a los paquetes TCP. Además de estos, sería interesante detectar paquetes sospechosos de tipo UDP e ICMP.

b. Señuelo

Una parte esencial para que un honeypot sea eficiente es que el atacante no sea capaz de detectar que lo es. Por ello es importante que el dispositivo finja comportarse como un elemento más del entorno en el que se encuentra.

Como el dispositivo se va a encontrar en un entorno IoT deberá fingir ser un dispositivo de este tipo. Por ello, se ha desarrollado, usando como controlador central a Mozilla Webthings Gateway **[15]** y haciendo uso del Mozilla WebThings Framework **[18]** y la WebThings API **[19]**, un programa que finja que el *Cibercanario* es una bombilla inteligente llamada *SLamp*. Cuando se active el Gateway, este lo detectará de la siguiente manera:

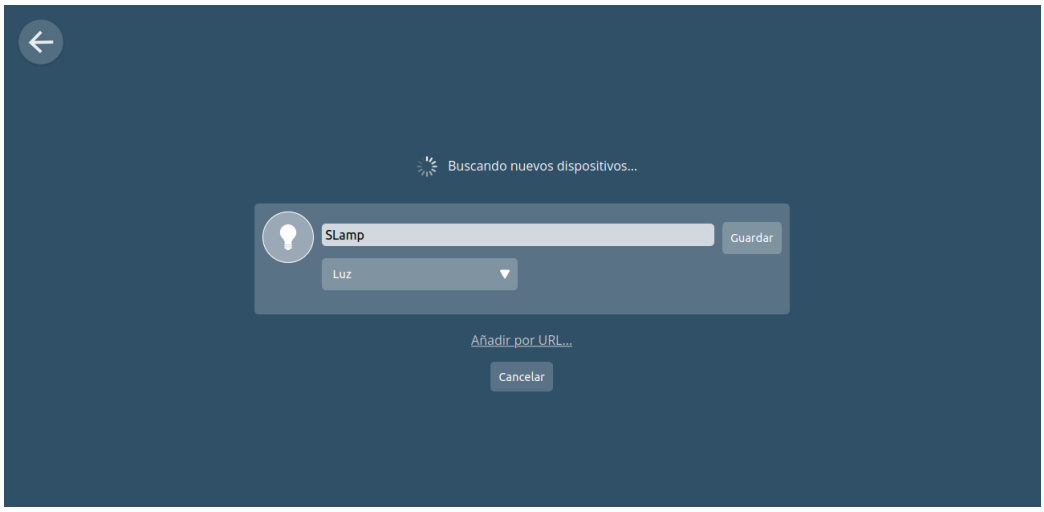

FIGURA 10: GATEWAY DETECTA EL *SEÑUELO*

No solo detectará el dispositivo y permitirá conectarse con él, sino que además permitirá realizar acciones que finjan encender, apagar o, incluso modular la intensidad de la luz (**Figura 11**). El objetivo de este funcionalidad es impedir que el atacante descubra la verdadera naturaleza del *Cibercanario*.

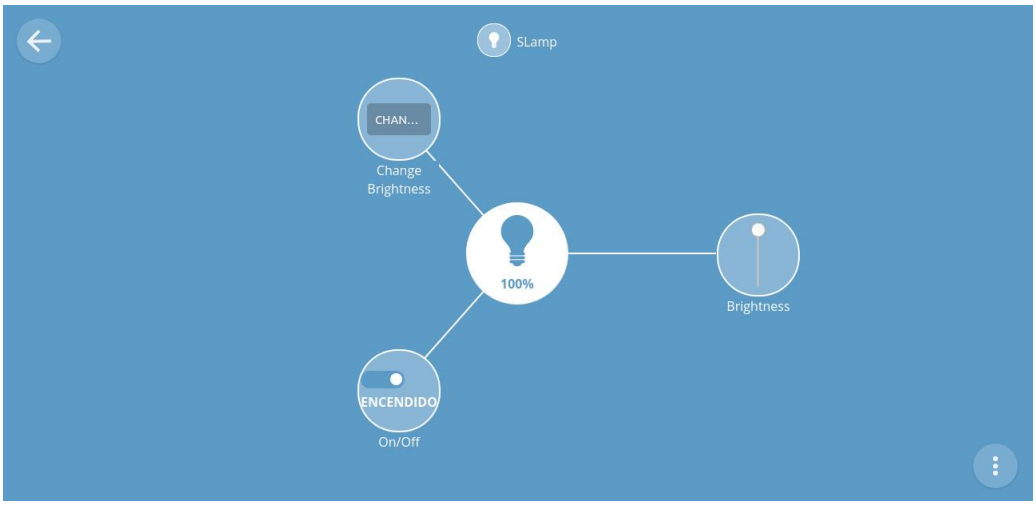

FIGURA 11: ACCIONES PERMITIDAS CON EL *SEÑUELO*

De esta manera, aún si el controlador acaba siendo comprometido, el dispositivo seguirá apareciendo como una inofensiva bombilla y podrá continuar defendiendo el entorno sin ser descubierto.

c. Manejador

Cada una de estas funcionalidades, tanto los detectores como el señuelo, serán un proceso distinto. El manejador, haciendo uso del paquete *multiprocessing* se encargará de coordinar los distintos procesos. Y es que la coordinación es muy importante en este escenario, cuando un detector encuentra alguna amenaza deberá avisar tanto a los otros detectores y al propio señuelo (para cortar cualquier tipo de comunicación si procede); y a la interfaz para que muestre el aviso correspondiente.

El manejador será, por lo tanto, el encargado de iniciar los distintos procesos y otorgarles las correspondientes herramientas para que puedan ser capaces de realizar sus funciones y comunicarse entre ellos si fuera necesario.

5. Pruebas

Para esta última parte del desarrollo se realizarán una serie de pruebas para verificar el correcto funcionamiento del *Cibercanario*.

Se realizarán dos tipos de pruebas. Por un lado, aquellas que prueban las funcionalidades de la interfaz, esto es, las funciones de insertar y eliminar IPs de la *lista blanca*. Para estas solo nos hará falta el dispositivo con el *Cibercanario* instalado y desplegado.

Por otro lado, las pruebas del funcionamiento de los detectores y que comprueban, sobre todo, si la comunicación tras un ataque se cierra realmente. Para estas pruebas si harán falta, además del dispositivo con el *Cibercanario* instalado y desplegado, un

equipo a parte (en este caso se ha usado una máquina virtual con Kali Linux) con *NMAP*, *tcpdump* y *Wireshark* instalados.

Debido a la longitud de la documentación de estas pruebas, éstas se encuentran detalladas en el *Apéndice C*. Como se puede observar en dicho apéndice, el *Cibercanario* ha conseguido superar todas las pruebas mencionadas anteriormente (funcionalidad y detectores).

5

Conclusiones y líneas futuras

En esta última sección se recogerán las conclusiones que se han alcanzado a lo largo del desarrollo del dispositivo, además de algunas líneas para posibles trabajos futuros.

Tal y como se ha comentado en varias ocasiones a lo largo de esta memoria, el terreno de las IoT se encuentra en constante evolución. Cada vez se utilizan en mayor cantidad y ya no solo en las grandes empresas, también en pequeñas e, incluso, en los hogares. Estos últimos lo emplean con un objetivo en mente, desde facilitar tareas (por medio de, por ejemplo, asistentes virtuales) hasta reducir el consumo de electricidad del hogar (como el uso de las bombillas y los enchufes inteligentes).

En este proyecto, nos hemos querido centrar en este último grupo de personas. Ya que entendemos que existen una gran cantidad de soluciones complejas y de elevado coste para grandes empresas, pero ninguna para el consumidor final. Con tal fin, se ha intentado diseñar y desarrollar un dispositivo honeypot de bajo presupuesto con el objetivo de poder desplegarse fácilmente en un entorno IoT y proteger el mismo.

El uso en el párrafo anterior de la palabra *intentado* no es casual. Y es que, aunque nos hemos podido acercar mucho al objetivo que se perseguía, la situación de cuarentena con motivo del COVID-19 no nos ha permitido poder desarrollar el dispositivo con algunas características (especialmente centradas en la seguridad) que teníamos planteadas desde un inicio.

Una de ellas, y quizás una de las que más ha condicionado el desarrollo, está relacionada con la elección del hardware. Ante la situación a la que nos encontrábamos solo tuvimos la elección de poder emplear una *Raspberry Pi 3 B*. De primeras el uso de este hardware aumentaba el precio base del dispositivo por encima de lo que esperábamos desde un inicio.

Pero, además de lo comentado arriba, el uso de una Raspberry nos ha obligado a usar un SO. En un inicio se planteó la posibilidad de no usar uno, lo que elimina algunos problemas, como el de la actualización de paquetes y 0-days. Además, el uso de uno tiene implicaciones como las vulnerabilidades asociadas a un SO complejo y un gasto excesivo de energía (que es, de hecho, uno de los requisitos que tuvimos en cuenta para la elección del hardware).

Sin embargo, dejando esto a un lado, lo cierto es que, gracias a la elaboración de un proyecto de estas características, se han adquirido muchos conocimientos tanto de áreas en las que se han profundizado bastante en el Grado de Ingeniería del Software, como en otras que no se tratan de manera tan extensa. Entre las primera destaca principalmente el proceso de desarrollo del software siguiendo la metodología SCRUM, una metodología que se ha empleado en diversas asignaturas del grado. Se ha empleado, además, algunas aplicaciones que nos ha permitido controlar el proceso desarrollo del dispositivo, tanto la organización de tareas por medio de *Trello* (que, aun siendo un proyecto desarrollado por una sola persona, la organización mejora considerablemente usando este tipo de plataformas), como mantener un control de las actualizaciones del código a través de *Github*.

También se ha tomado consciencia del desarrollo de proyectos de grandes dimensiones. Aunque es cierto que a lo largo del grado se han desarrollado proyectos que han englobado un cuatrimestre entero, nunca habían sido de carácter individual.

Pero además de esto, se ha profundizado en el funcionamiento de las redes y de la seguridad de estas. Para estos dos puntos ha sido esencial los conocimientos adquiridos en las asignaturas de Redes y Sistemas Distribuidos; y Seguridad en Servicios y Aplicaciones respectivamente.

Con respecto a las posibles líneas futuras del proyecto, existen varios caminos abiertos:

- **Empleo de un hardware más acorde:** Como se ha comentado unos párrafos más arriba, el dispositivo debería instalarse en un hardware no solo más barato, sino que además nos permitiera poder hacerlo sin emplear un SO.
- **Expandir detectores:** Aunque el dispositivo es capaz de detectar tanto escaneos de puertos dentro de la red local, como paquetes del exterior que puedan suponer una amenaza, se pueden añadir otras características. Por ejemplo, simular un inicio de sesión para detectar intentos de intrusión o reaccionar también ante mensajes a nivel de aplicación, dentro de la red local, que no provengan del Gateway.

APÉNDICES

APÉNDICE A: MANUAL DE DESPLIEGUE

En este apartado se mostrará una explicación para el correcto despliegue del sistema. Para ello se necesitará de los siguiente:

- Una Raspberry (o similar) que sirva de base para la instalación. En nuestro caso usaremos una Raspberry Pi 3B.
- Una microSD (nosotros usaremos una de 32Gb).
- Un adaptador microSD (si fuera necesario para conectarlo con el ordenador).
- Un teclado y un ratón USB.
- Ordenador (ya sea sobremesa o portátil) con lector de tarjetas SD.

Se describirán ahora los pasos a seguir para la instalación del *Cibercanario*.

1. Instalación Ubuntu Server

Para la ejecución del Cibercanario se ha decidido usar como SO base un *Ubuntu Server*. Instalar el programa *Raspberry Pi Imager* **[20]**. Una vez hecho, insertar la microSD con el adaptador (si fuera necesario) en el ordenador y abrir *Raspberry Pi Imager*.

Nos aparecerá una pantalla en la que elegir el SO y el dispositivo donde queremos instalarlo (**Figura A.1**). Elegimos el Ubuntu Server (**Figura A.2**) y en el segundo apartado nuestra microSD y pulsamos en *Write*.

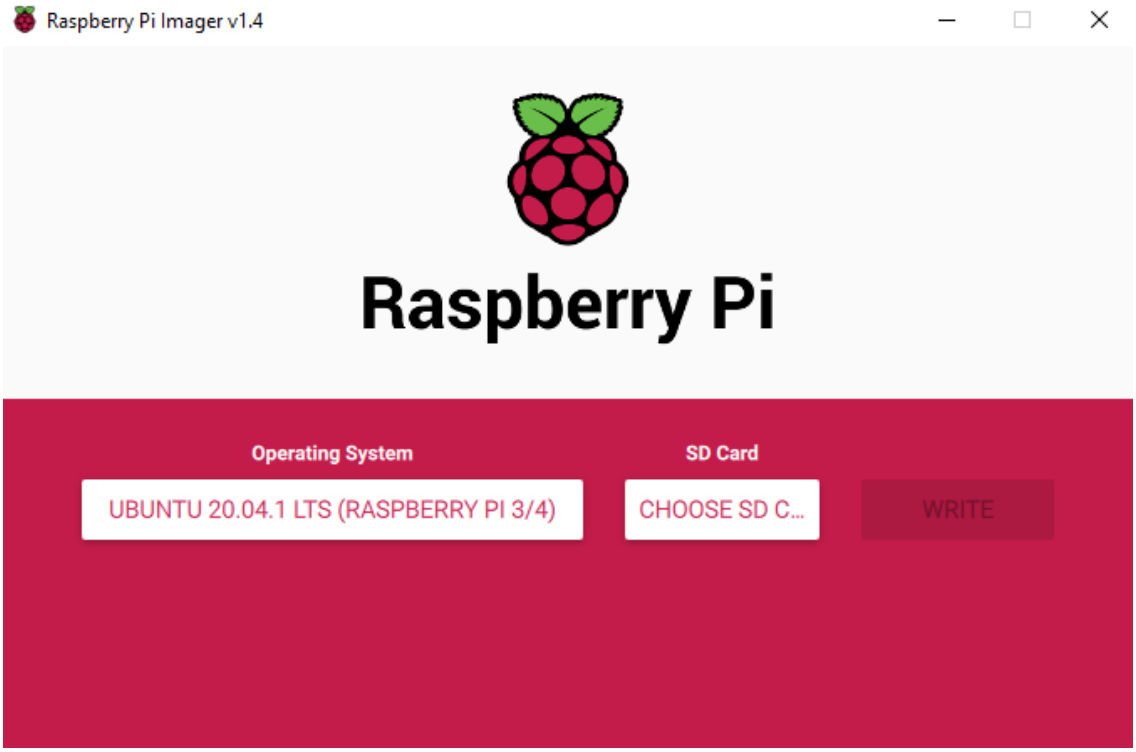

FIGURA A.1: RASPBERRY PI IMAGER

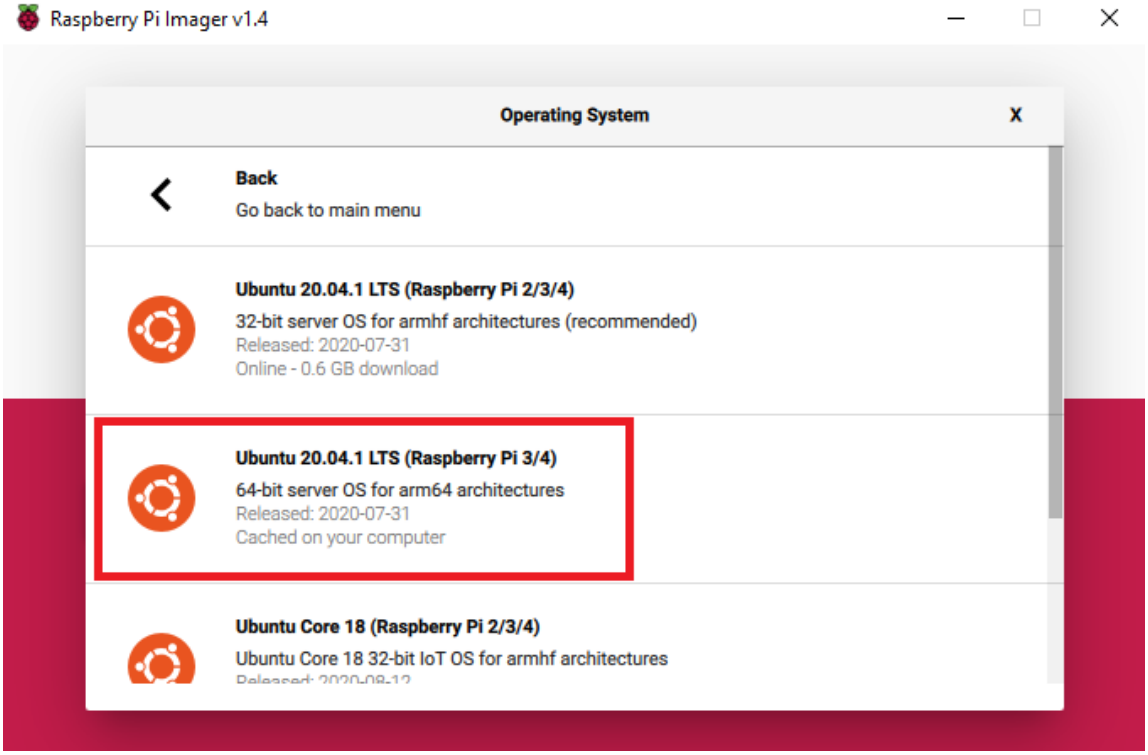

FIGURA A.2: SELECCIONAR SO

Una vez instalado, el sistema expulsará la microSD automáticamente. Volver a conectarla al ordenador y abrirla con el *Explorador de archivos* (o cualquier gestor de este tipo).

Buscar el archivo *network_config* y abrirlo con un editor de texto. Este archivo nos permite conectarle la red WIFI antes de ejecutarlo con la Raspberry. También se puede realizar la instalación conectándole un cable Ethernet a la Raspberry si así se prefiere y, en este último caso, no será necesario hacer estos pasos.

En caso de elegir la opción del WIFI, habrá que indicar de la siguiente manera el nombre de la red y la contraseña de esta (hay que recordar quitar los corchetes del principio de las líneas para que funcione):

```
# This file contains a netplan-compatible configuration which cloud-init
# will apply on first-boot. Please refer to the cloud-init documentation and
# the netplan reference for full details:
#
# https://cloudinit.readthedocs.io/
# https://netplan.io/reference
#
# Some additional examples are commented out below

version: 2
ethernets:
  eth0:
    dhcp4: true
    optional: true
wifis:
  wlan0:
    dhcp4: true
    optional: true
    access-points:
      My_WIFI:
        password: "papopepo"
```

FIGURA A.3: MODIFICACIÓN DEL NETWORK_CONFIG

Tras esto, guardar el archivo, desconectar del ordenador, conectar a la Raspberry Pi e iniciarla. Lo primero que solicitará Ubuntu Server al iniciarlo será un usuario y contraseña que son los siguientes:

- Usuario: **ubuntu**
- Contraseña: **ubuntu**

Es probable que al introducir los valores falle, a veces tiene que aparecer el mensaje de SSH primero para aceptar la conexión. Tras entrar, se solicitará un cambio de contraseña.

En caso de usar WIFI, para que este funcione correctamente, hay que ejecutar los siguientes comandos:

```
sudo netplan generate
sudo netplan apply
```

Sería interesante ahora actualizar el sistema antes de instalar cualquier aplicación/paquete por medio de los comandos:

```
sudo apt update
sudo apt upgrade
```

Automáticamente el teclado se configura al inglés, se puede cambiar el idioma de este con el siguiente comando y reiniciar el sistema:

```
sudo dpkg-reconfigure keyboard-configure
sudo reboot
```

Pasando ahora con el escritorio, deberemos instalar primero *tasksel*:

```
sudo apt install tasksel
```

Tras esto, elegiremos un *Display Manager*. Existen dos opciones bastante ligeras, SLiM y LightDM. Nosotros elegiremos la primera, pero ambas son buenas para tener un escritorio lo más simple posible.

Lo instalamos con el siguiente comando:

```
sudo apt install slim
```

Por último, corremos *taskel* con el siguiente comando, elegimos de las opciones que nos aparecer *Lubuntu Desktop* y reiniciamos:

```
taskel
reboot
```

Al arrancar el sistema ya se iniciará el escritorio (el usuario y contraseña serán los mismos de antes tras cambiar la contraseña).

2. Instalación paquetes para *Cibercanario*

Para instalar los paquetes, tendremos que instalar primero pip3 con el siguiente comando:

```
sudo apt install python3-pip
```

Para la interfaz usaremos Tkinter, por lo que necesitaremos instalarlo con el siguiente comando:

```
sudo apt install python3-tk
```

Necesitaremos, también, instalar los siguientes paquetes que necesita el Cibercanario para su funcionamiento:

```
sudo pip3 install IPy
sudo pip3 install psutil
```

```
sudo pip3 install webthing
```

Finalmente, conectar un USB con el código del *Cibercanario*, pasarlo al dispositivo, y correrlo usando:

```
sudo python3 canary.py
```

Para que funcione la funcionalidad del señuelo, será necesario instalar en otro dispositivo Webthings Gateway. En este manual no se explicará el proceso de instalación, pero se puede encontrar en su web oficial **[15]**.

APÉNDICE B: MANUAL DE USO

En este apartado se mostrará, además del proceso de instalación del dispositivo, una explicación para el correcto uso del sistema y una guía por las distintas funcionalidades que se ofrecen al usuario.

1. Instalación *Cibercanario*

La instalación del Cibercanario es un proceso simple, en pocos pasos estará totalmente configurado y funcionando.

Antes de introducir la microSD en el dispositivo, será necesario conectarlo al ordenador (que deberá tener un lector de tarjetas SD) con un adaptador microSD. Tras eso habrá que acceder al contenido de la tarjeta microSD con el *Explorador de archivos* (o cualquier gestor de este tipo).

Buscar el archivo *network_config* y abrirlo con un editor de texto. Este archivo nos permite conectarle la red WIFI antes de ejecutarlo con la Raspberry.

```
# This file contains a netplan-compatible configuration which cloud-init
# will apply on first-boot. Please refer to the cloud-init documentation and
# the netplan reference for full details:
#
# https://cloudinit.readthedocs.io/
# https://netplan.io/reference
#
# Some additional examples are commented out below

version: 2
ethernets:
  eth0:
    dhcp4: true
    optional: true
wifis:
  wlan0:
    dhcp4: true
    optional: true
    access-points:
      My_WIFI:
        password: "papopepo"
```

FIGURA B.1: CONFIGURACIÓN WIFI

Habrá que indicar como se indica en la **Figura B.1**, el nombre de la red y la contraseña de esta. No será necesario modificar ningún elemento del resto del documento.

Tras esto, guardar el archivo, desconectar del ordenador, conectar al dispositivo e iniciarla. Lo primero que solicitará al iniciarlo será un usuario y contraseña que son los siguientes:

- Nombre de usuario: **ubuntu**
- Contraseña: **ubuntu**

Se solicitará modificar la contraseña (la cual se pedirá introducir junto al nombre de usuario cada vez que se encienda el *Cibercanario*) y tras hacerlo, el sistema se iniciará y la aplicación empezará a funcionar.

2. Detección de amenazas

Una vez que el *Cibercanario* es iniciado y se muestra la pantalla inicial (**Figura B.2**), comenzará automáticamente a detectar cualquier amenaza que pueda aparecer.

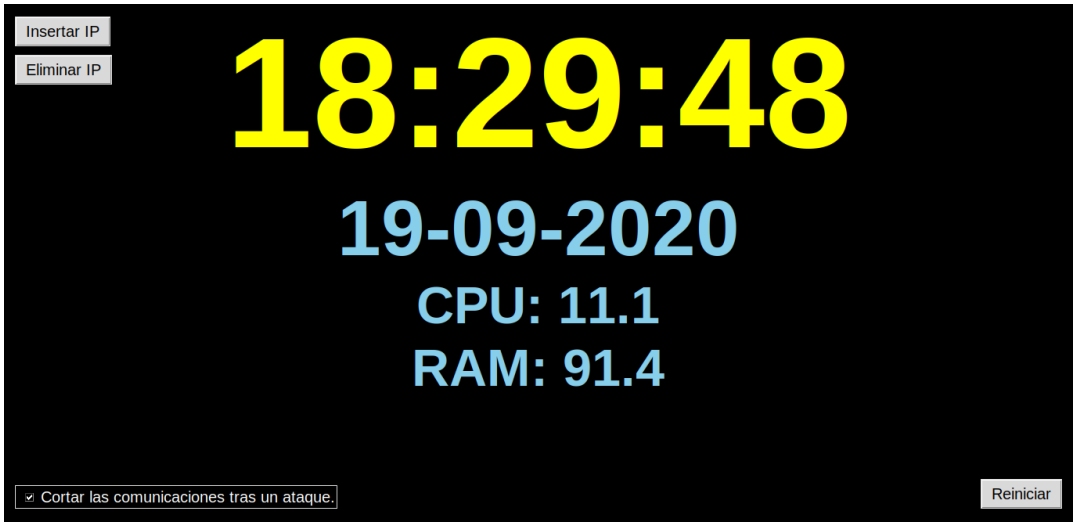

FIGURA B.2: PANTALLA INICIAL *CIBERCANARIO*

Cuando la amenaza aparezca, se mostrará una pantalla en la que se advierte de la misma, junto con la hora a la que se ha detectado y un listado en el que se detalla brevemente que se ha detectado. Si se ha indicado que se corten las comunicaciones tras detectar un ataque (ver apartado 4 de este manual), el dispositivo no detectará más amenazas hasta que no se reinicie (ver apartado 5 de este manual).

3. Insertar nueva IP

Para insertar una nueva IP a la *lista blanca* (listado de IPs que el *Cibercanario* no detectará como amenazas), se deberá, estando en la pantalla inicial, clicar en el botón de *Insertar IP*.

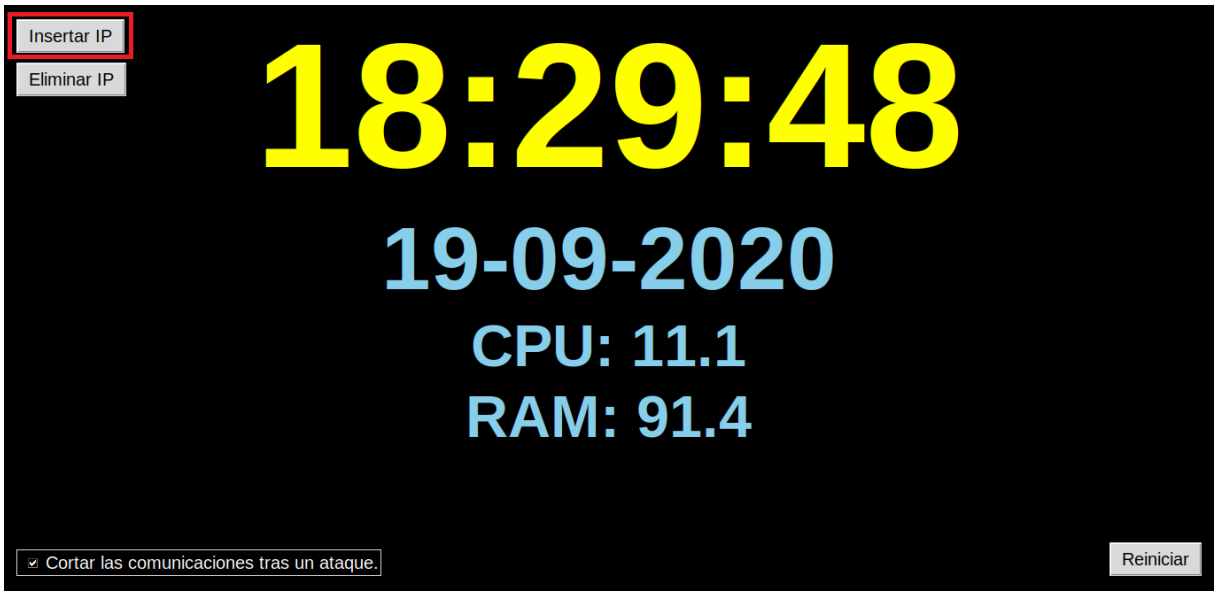

FIGURA B.3: BOTÓN PARA INSERTAR NUEVA IP

Tras esto aparecerá una pantalla (**Figura B.3**) en la que el usuario puede introducir una determinada IP en el cuadro de texto que aparece. Debe tenerse en cuenta que el cuadro de texto no deberá estar vacío, que la IP no debe estar ya incluida en la *lista blanca* y que la IP introducida deberá mantener la estructura de este tipo de direcciones: XXX.XXX.XXX.XXX (p.e. 192.168.1.72). Si estas tres condiciones se

cumplen, al clicar en el botón que se encuentra justo debajo del cuadro de texto (*Insertar*) se guardará la IP, en caso contrario aparecerá un mensaje de error.

En caso de no querer hacer ninguna operación, se puede regresar a la pantalla inicial (**Figura B.2**) clicando en el botón de *Regresar* que se encuentra en la esquina inferior izquierda.

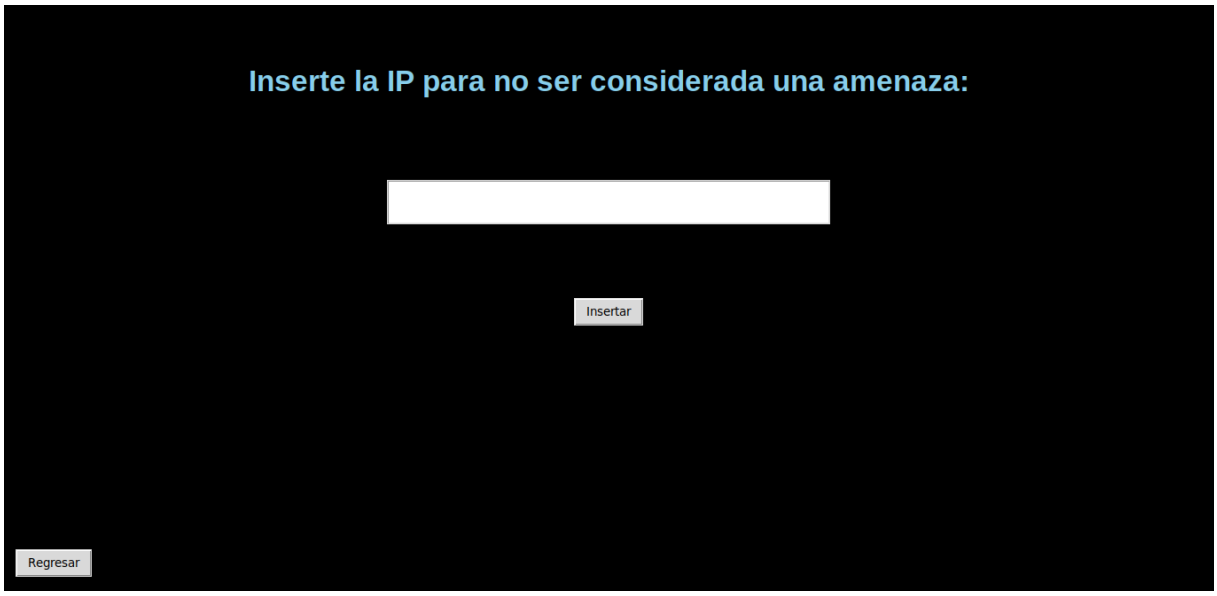

FIGURA B.4: INSERTAR NUEVA IP

4. Borrar IP existente

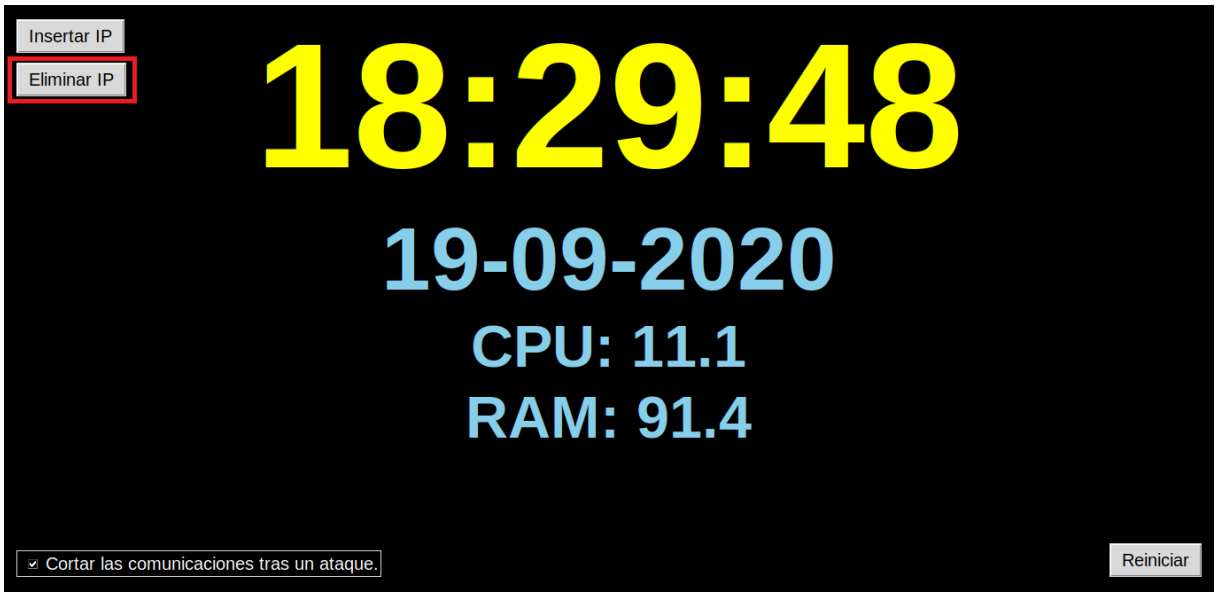

FIGURA B.5: BOTÓN BORRAR IP EXISTENTE

Para borrar una nueva IP ya existente de la *lista blanca*, se deberá, estando en la pantalla inicial, clicar en el botón de Eliminar IP.

Tras esto aparecerá una pantalla (**Figura B.6**) en la que el usuario puede señalar de la lista que se le muestra una IP. Debe tenerse en cuenta que se deberá realizar una selección antes de clicar en el botón *Eliminar*. En caso contrario se mostrará un mensaje de error. Si se encuentra seleccionada, al clicar en dicho botón que se encuentra justo debajo de la lista se eliminará la IP.

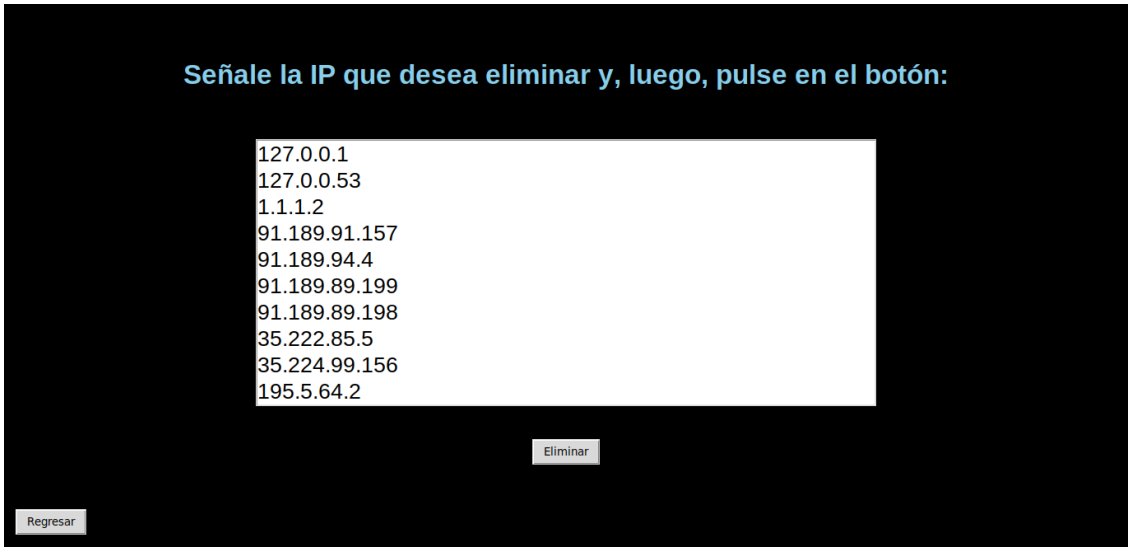

FIGURA B.6: BOTÓN BORRAR IP EXISTENTE

En caso de no querer hacer ninguna operación, se puede regresar a la pantalla inicial (**Figura A.2**) clicando en el botón de *Regresar* que se encuentra en la esquina inferior izquierda.

5. Cortar comunicaciones tras detectar una amenaza

Si se desea que, tras detectar un ataque, el *Cibercanario* cierre las comunicaciones se deberá señalar el recuadro que aparece en página inicial (**Figura B.7**). En caso contrario, se deberá dejar sin señalar (por defecto estará seleccionado).

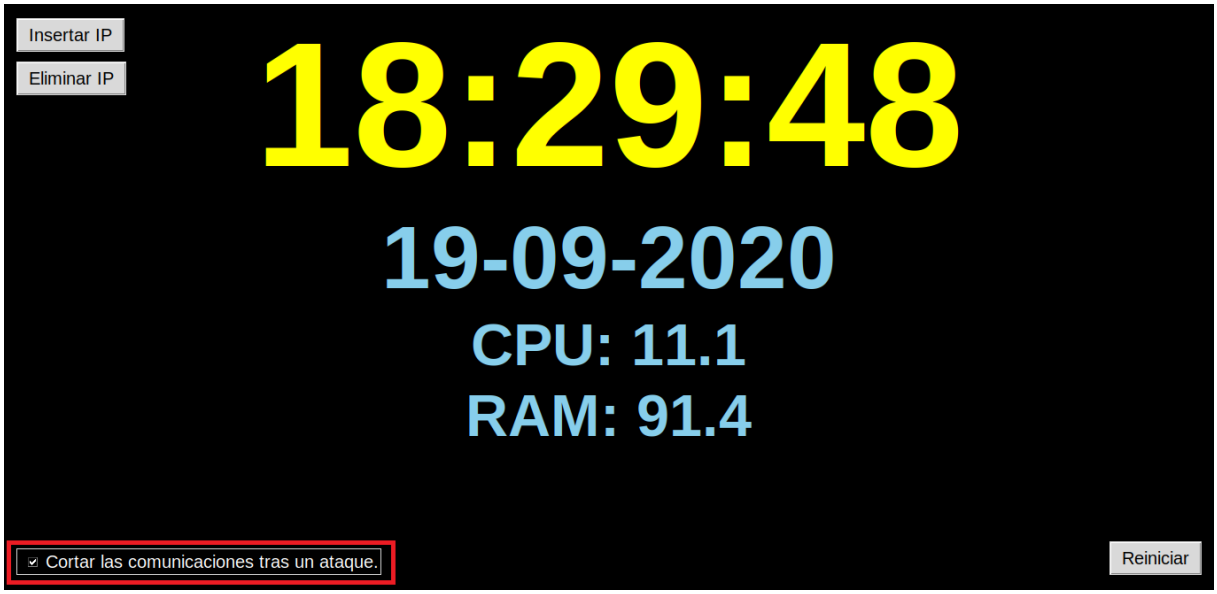

FIGURA B.7: BOTÓN BORRAR IP EXISTENTE

6. Reiniciar el *Cibercanario*

Si se desea que el *Cibercanario* se reinicie y vuelva a abrir las comunicaciones (si se han cerrado), deberá clicar en el botón de *Reiniciar* en la pantalla inicial.

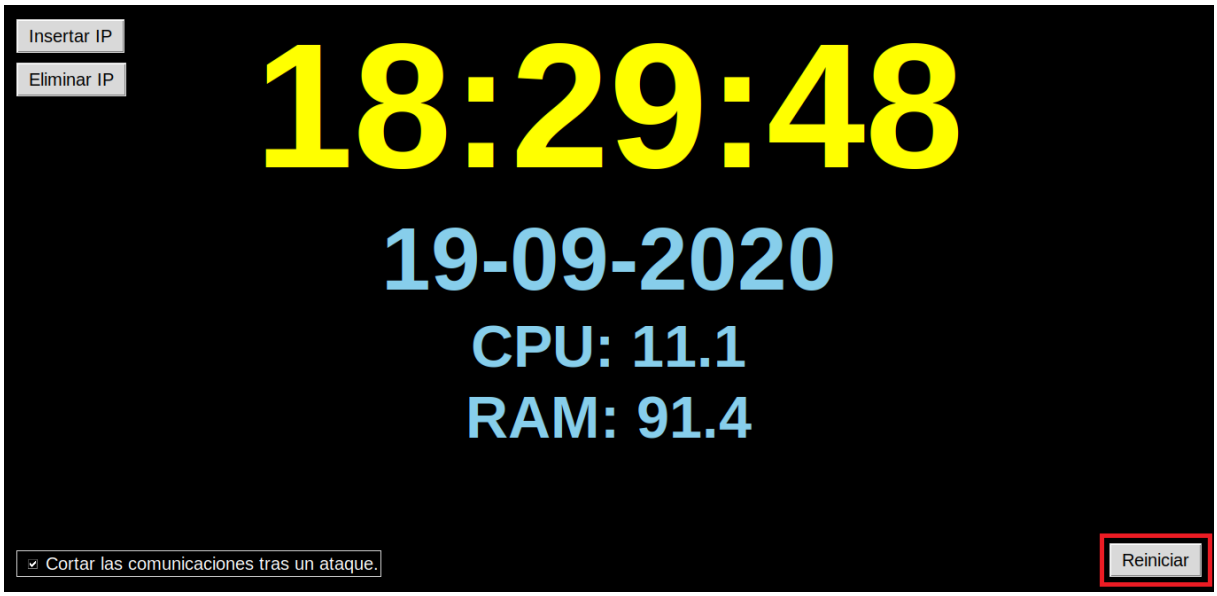

FIGURA B.8: BOTÓN REINICIAR *CIBERCANARIO*

APÉNDICE C: INFORME DE PRUEBAS

En este apartado, se mostrarán las pruebas realizadas sobre el dispositivo *Cibercanario* para comprobar el correcto funcionamiento de todas sus funcionalidades.

1. Insertar una IP correctamente

Resumen

Comprobar que al insertar una dirección IP correcta, el *Cibercanario* la introduce en la *lista blanca* satisfactoriamente.

Requisitos

- Dispositivo *Cibercanario* correctamente instalado y desplegado.

Procedimiento

a. Estando en la pantalla inicial, clicar en el botón *Insertar IP*.

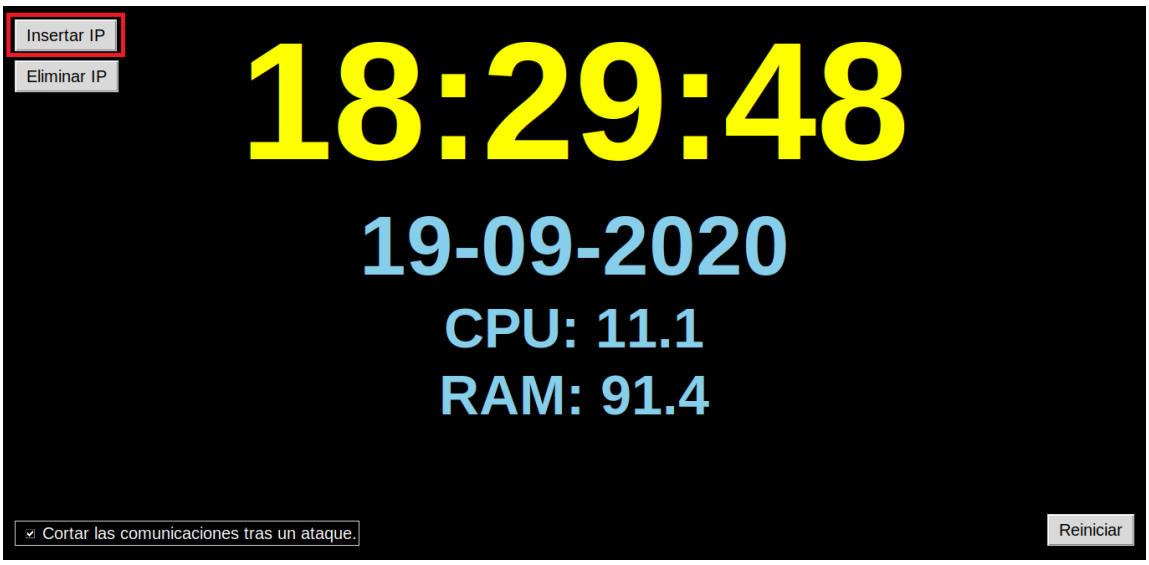

FIGURA C.1.1

b. En el cuadro de texto, introducir la IP 81.25.5.213.

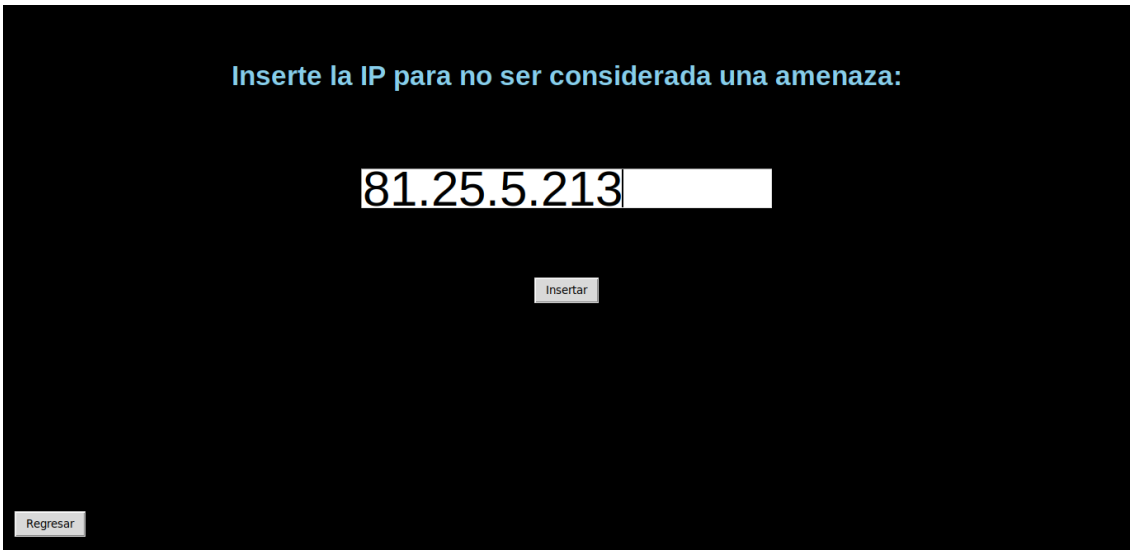

FIGURA C.1.2

c. Clicar en *Insertar*.

d. Estando de nuevo en la pantalla inicial, clicar en el botón *Eliminar IP*.

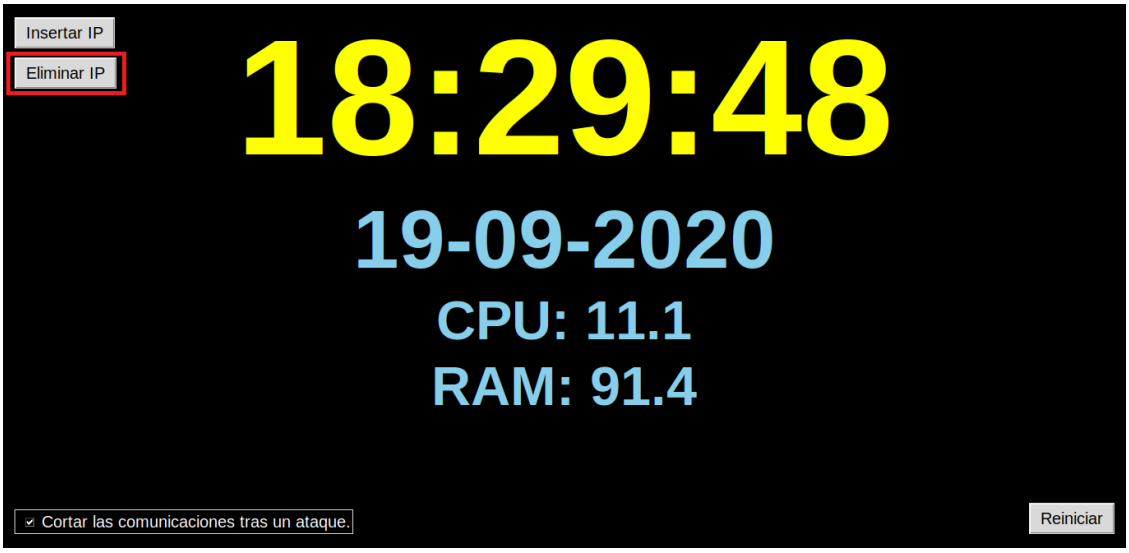

FIGURA C.1.3

e. En la lista buscar la IP 81.25.5.213.

f. Verificar que la IP aparece en la lista y se ha introducido correctamente.

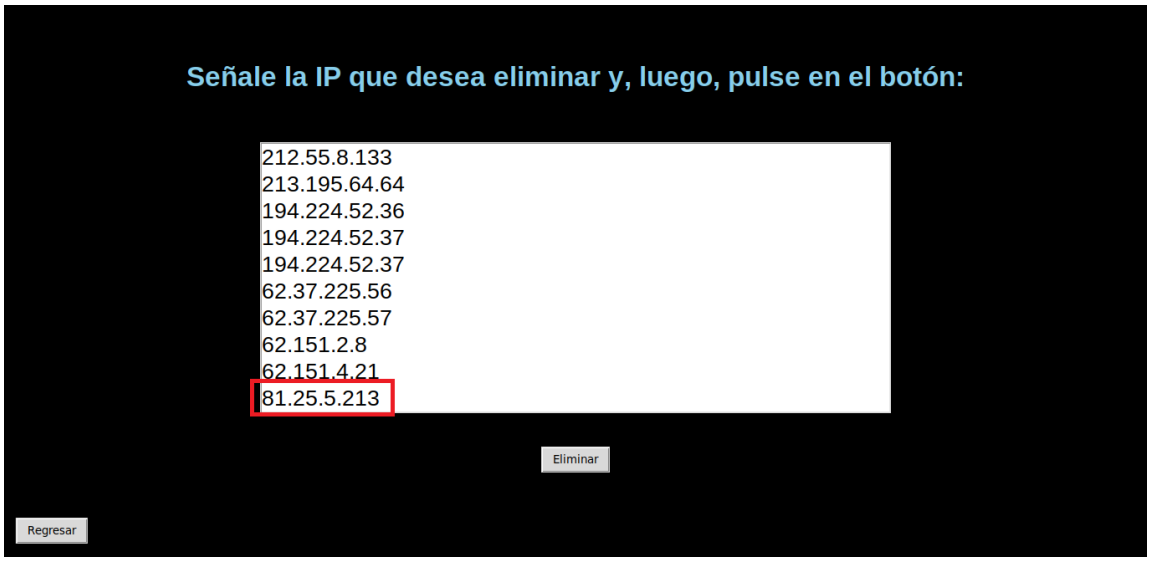

FIGURA C.1.4

2. Insertar una IP vacía

Resumen

Comprobar que al insertar una dirección IP vacía, el *Cibercanario* muestra un mensaje de error.

Requisitos

- Dispositivo *Cibercanario* correctamente instalado y desplegado.

Procedimiento

a. Estando en la pantalla inicial, clicar en el botón *Insertar IP*.

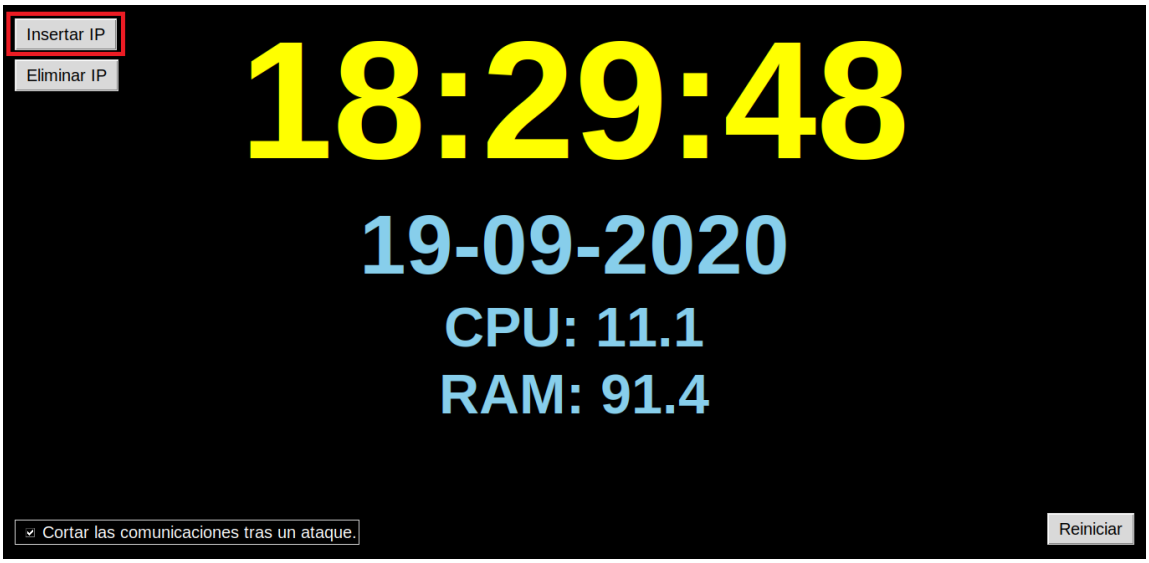

FIGURA C.2.1

b. En el cuadro de texto, introducir ningún valor y clicar directamente en *Insertar*.

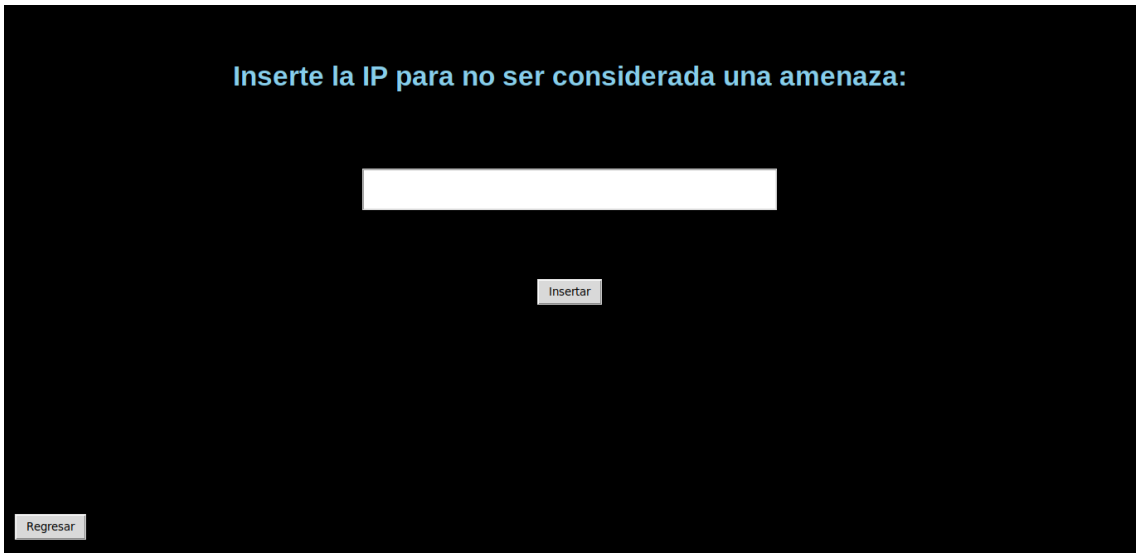

FIGURA C.2.2

c. Verificar que el *Cibercanario* muestra un mensaje de error.

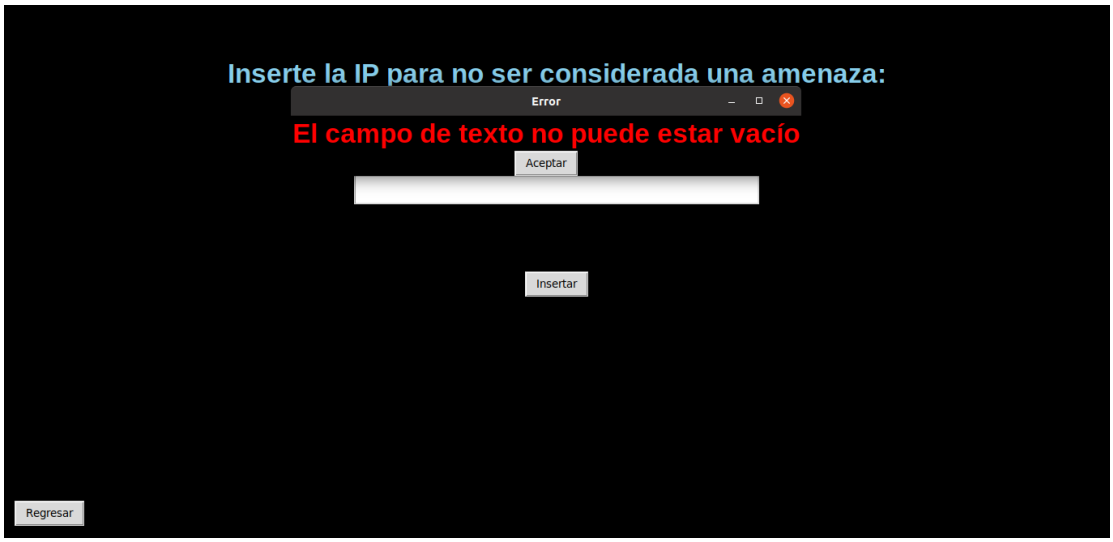

FIGURA C.2.3

d. Pulsar el botón *Aceptar* en el mensaje de error.

e. Pulsar el botón de *Regresar*.

f. Estando de nuevo en la pantalla inicial, clicar en el botón *Eliminar IP*.

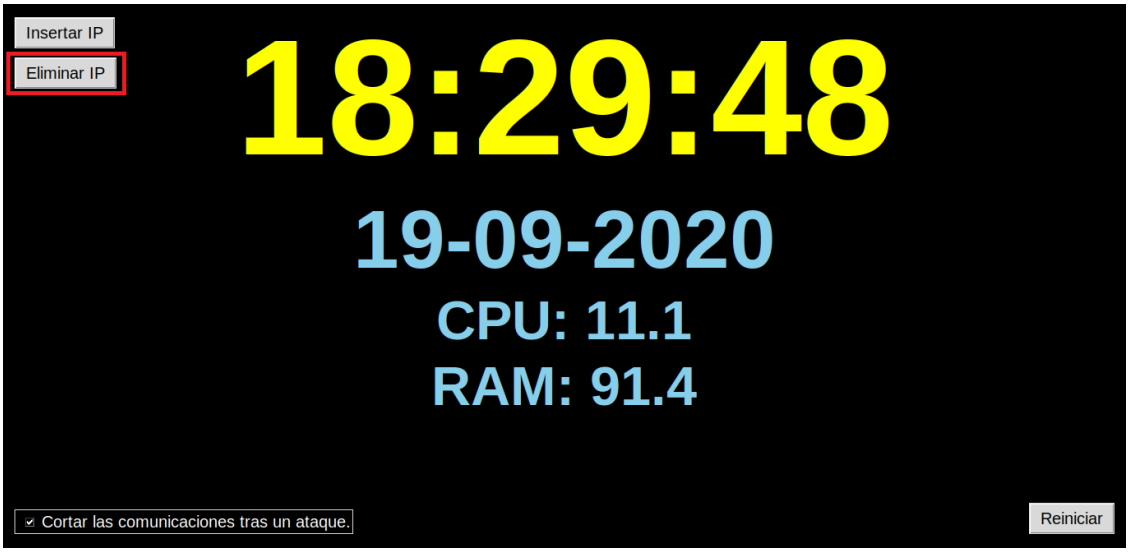

FIGURA C.2.4

g. En la lista buscar si existe algún valor en blanco.

h. Verificar que no aparece el valor introducido.

3. Insertar una IP no válida

Resumen

Comprobar que al insertar una dirección IP no válida, el *Cibercanario* muestra un mensaje de error.

Requisitos

- Dispositivo *Cibercanario* correctamente instalado y desplegado.

Procedimiento

a. Estando en la pantalla inicial, clicar en el botón *Insertar IP*.

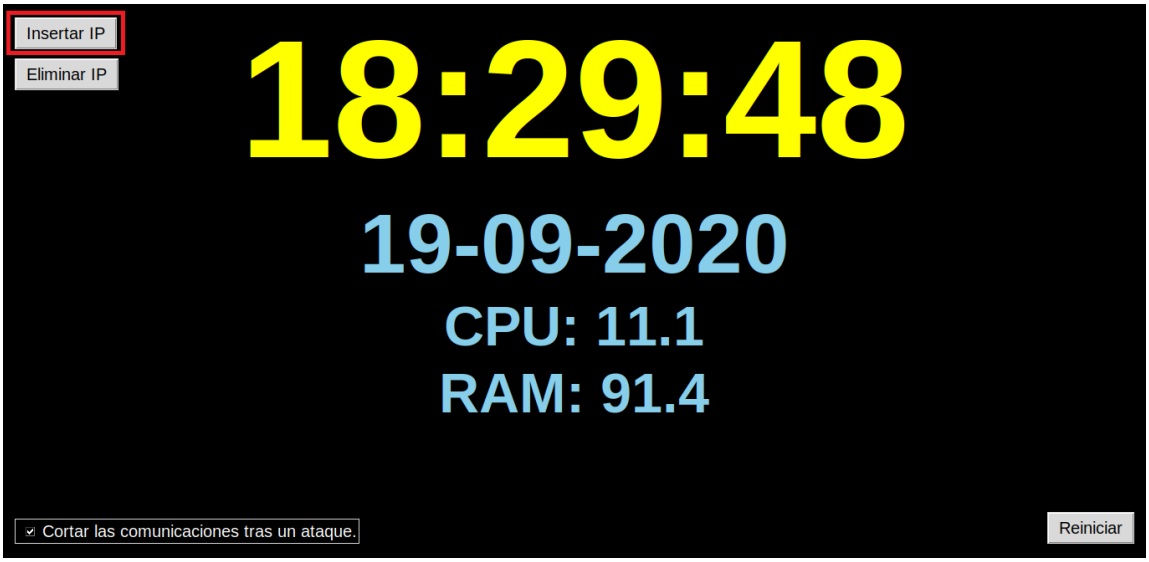

FIGURA C.3.1

b. En el cuadro de texto, introducir "papopepo" y clicar en *Insertar*.

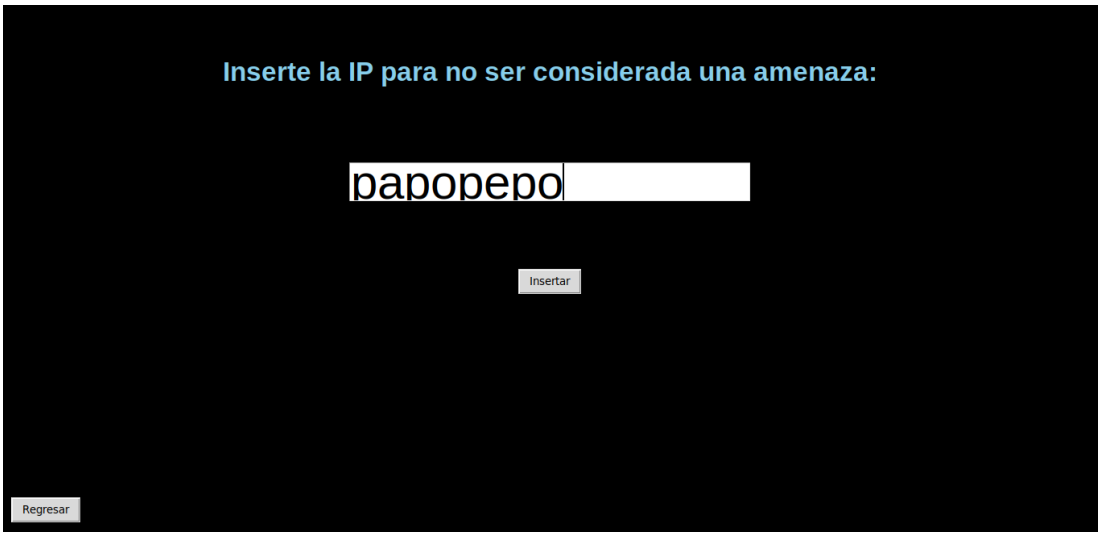

FIGURA C.3.2

c. Verificar que el *Cibercanario* muestra un mensaje de error.

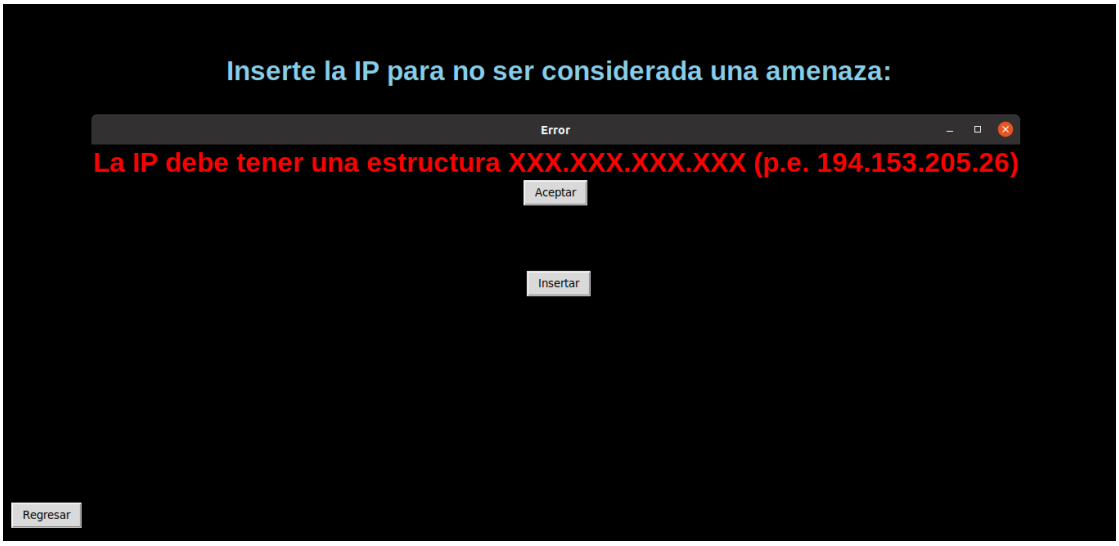

FIGURA C.3.3

d. Pulsar el botón *Aceptar* en el mensaje de error.

e. Pulsar el botón de *Regresar*.

f. Estando de nuevo en la pantalla inicial, clicar en el botón *Eliminar IP*.

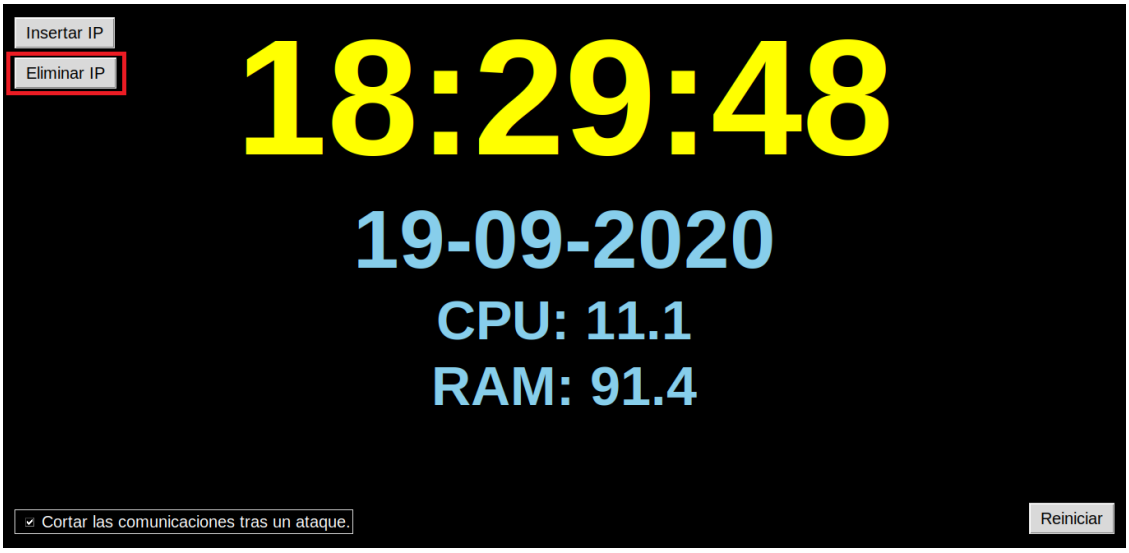

FIGURA C.2.4

g. En la lista buscar si existe el valor "papopepo".

h. Verificar que no aparece el valor introducido.

4. Insertar una IP con una estructura no válida

Resumen

Comprobar que al insertar una dirección IP con una estructura no válida, el *Cibercanario* muestra un mensaje de error.

Requisitos

- Dispositivo *Cibercanario* correctamente instalado y desplegado.

Procedimiento

a. Estando en la pantalla inicial, clicar en el botón *Insertar IP*.

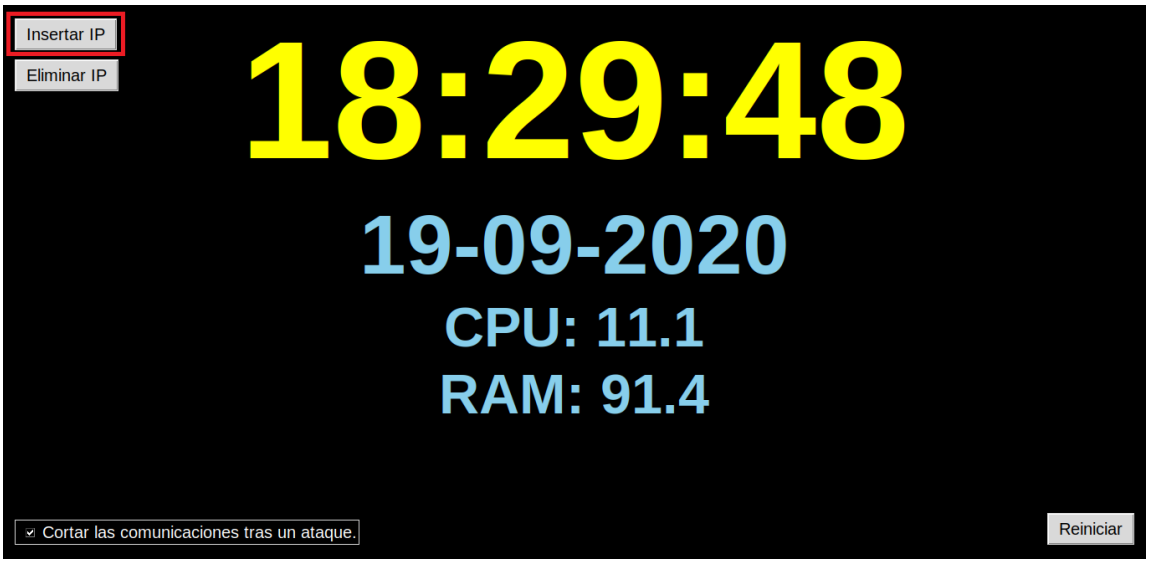

FIGURA C.4.1

b. En el cuadro de texto, introducir "168.520.56.23" y clicar en *Insertar*.

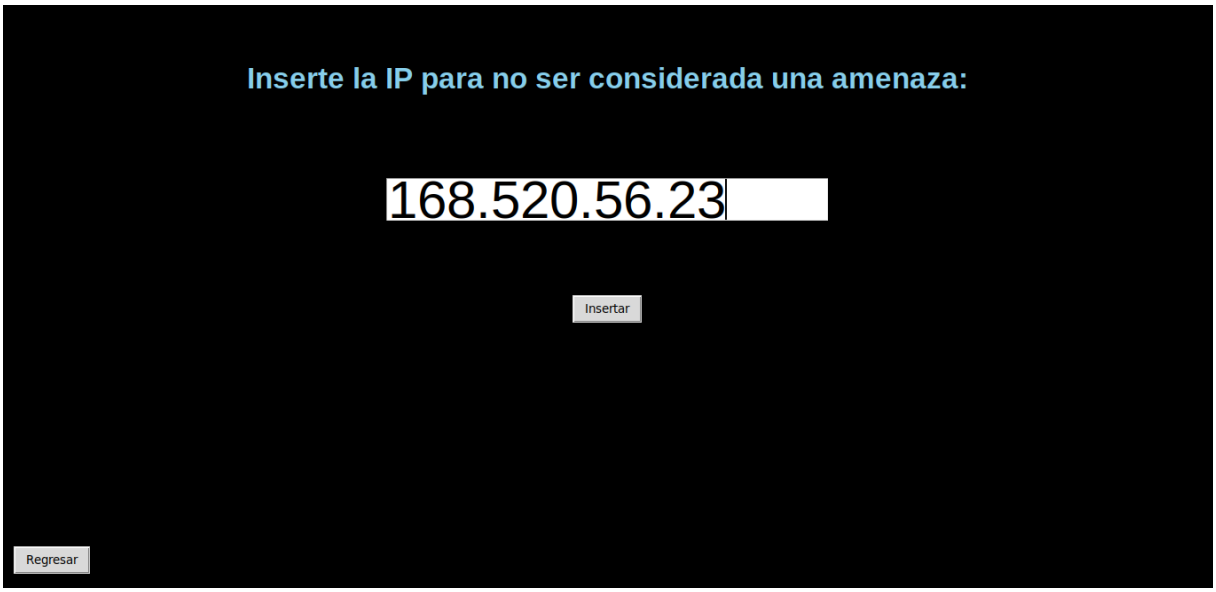

FIGURA C.4.2

c. Verificar que el *Cibercanario* muestra un mensaje de error.

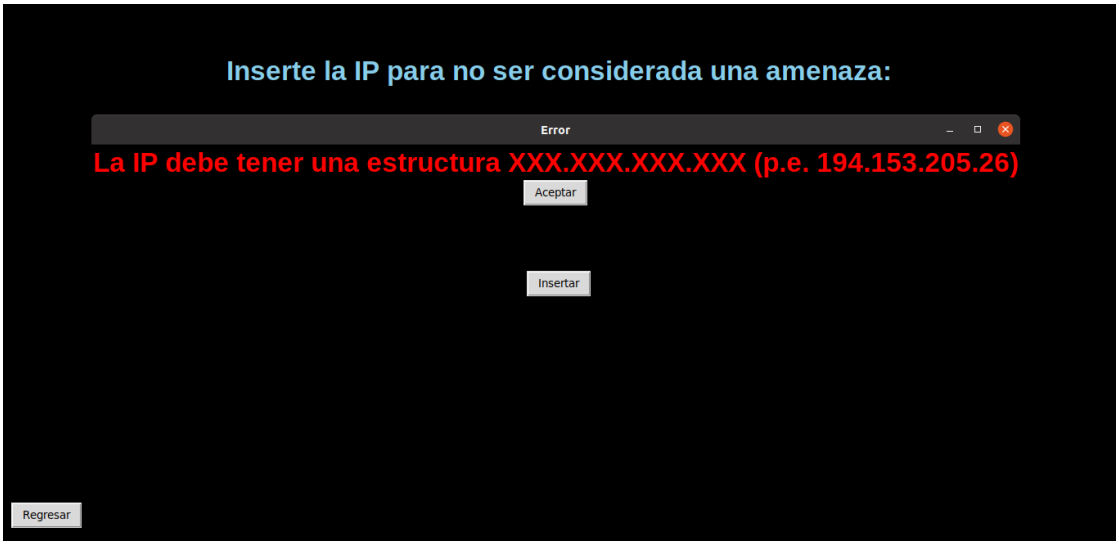

FIGURA C.4.3

d. Pulsar el botón *Aceptar* en el mensaje de error.

e. Pulsar el botón de *Regresar*.

f. Estando de nuevo en la pantalla inicial, clicar en el botón *Eliminar IP*.

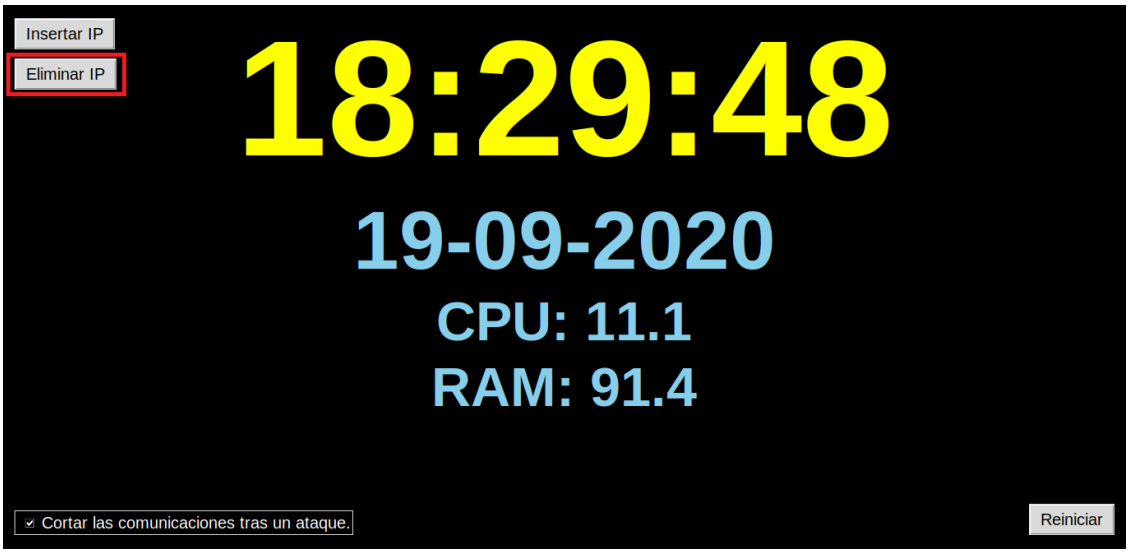

FIGURA C.4.4

g. En la lista buscar si existe el valor 168.520.56.23.

h. Verificar que no aparece el valor introducido.

5. Insertar una IP ya existente

Resumen

Comprobar que al insertar una dirección IP que ya está en la *lista blanca*, el *Cibercanario* muestra un mensaje de error.

Requisitos

- Dispositivo *Cibercanario* correctamente instalado y desplegado.

Procedimiento

a. Estando en la pantalla inicial, clicar en el botón *Insertar IP*.

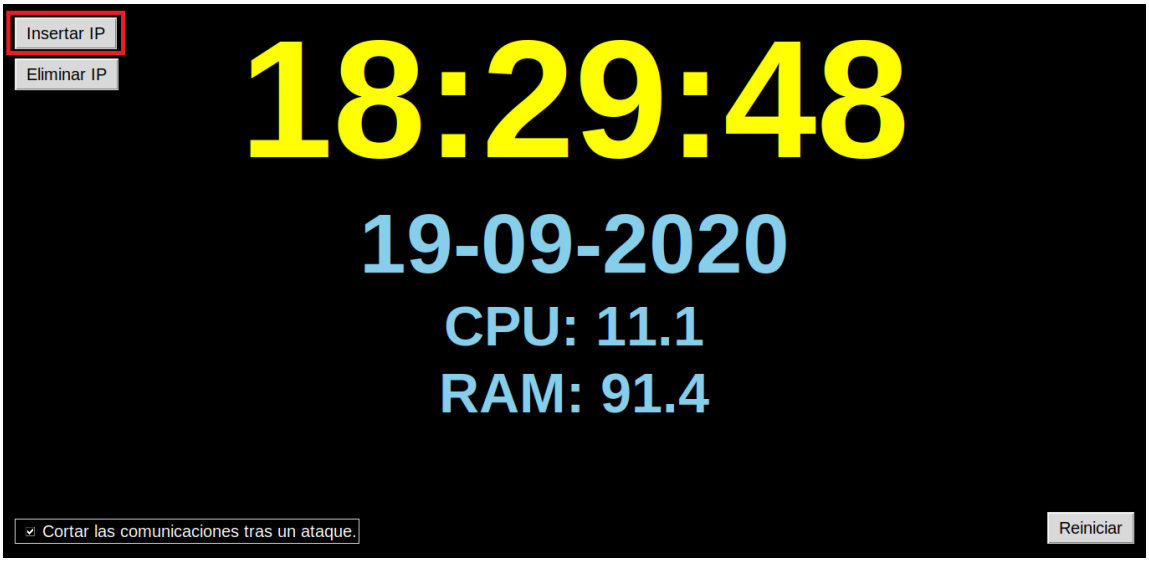

FIGURA C.5.1

b. En el cuadro de texto, introducir una IP ya existente como, por ejemplo, "127.0.0.1" y clicar en *Insertar*.

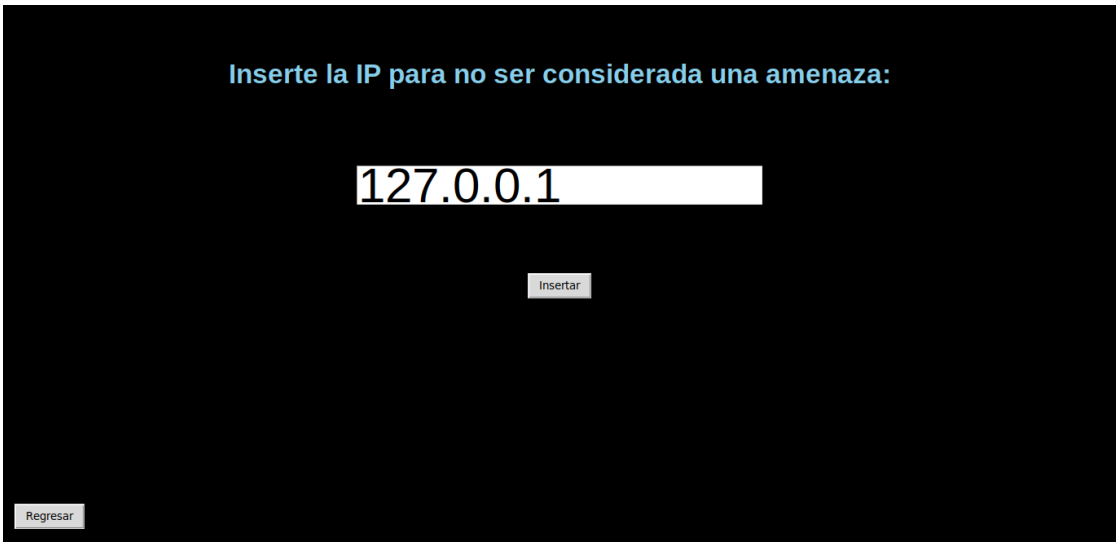

FIGURA C.5.2

c. Verificar que el *Cibercanario* muestra un mensaje de error.

FIGURA C.5.3

d. Pulsar el botón *Aceptar* en el mensaje de error.

e. Pulsar el botón de *Regresar*.

f. Estando de nuevo en la pantalla inicial, clicar en el botón *Eliminar IP*.

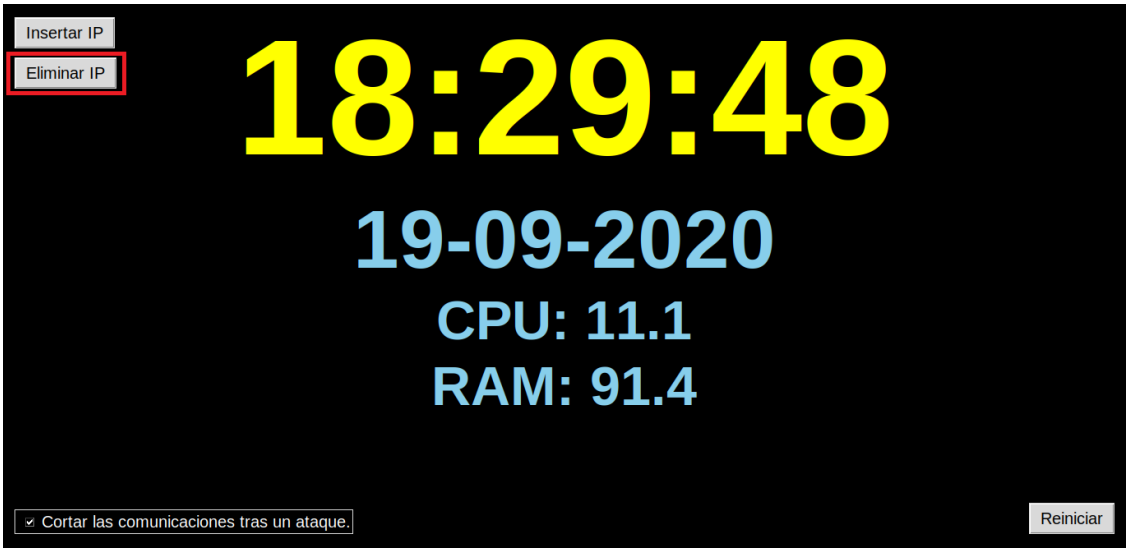

FIGURA C.5.4

g. En la lista buscar si existe el valor 127.0.0.1 duplicado.

h. Verificar que no aparece el valor introducido.

6. Eliminar una IP correctamente

Resumen

Comprobar que al eliminar una dirección IP existente, el *Cibercanario* la borra de la *lista blanca* satisfactoriamente.

Requisitos

- Dispositivo *Cibercanario* correctamente instalado y desplegado.

Procedimiento

a. Estando en la pantalla inicial, clicar en el botón *Eliminar IP*.

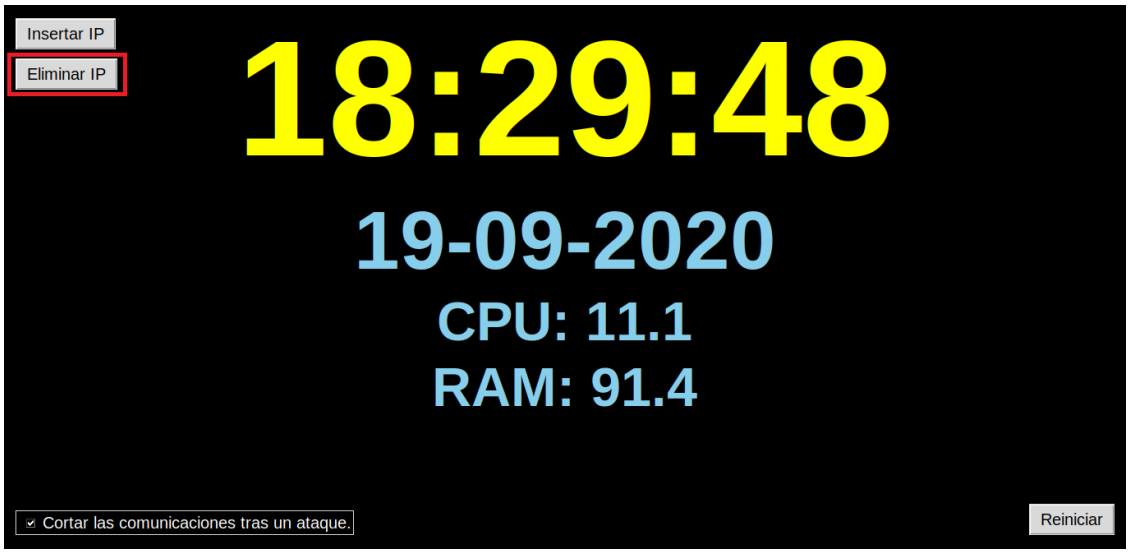

FIGURA C.6.1

b. En la lista, seleccionar la IP 81.25.5.213 introducida anteriormente.

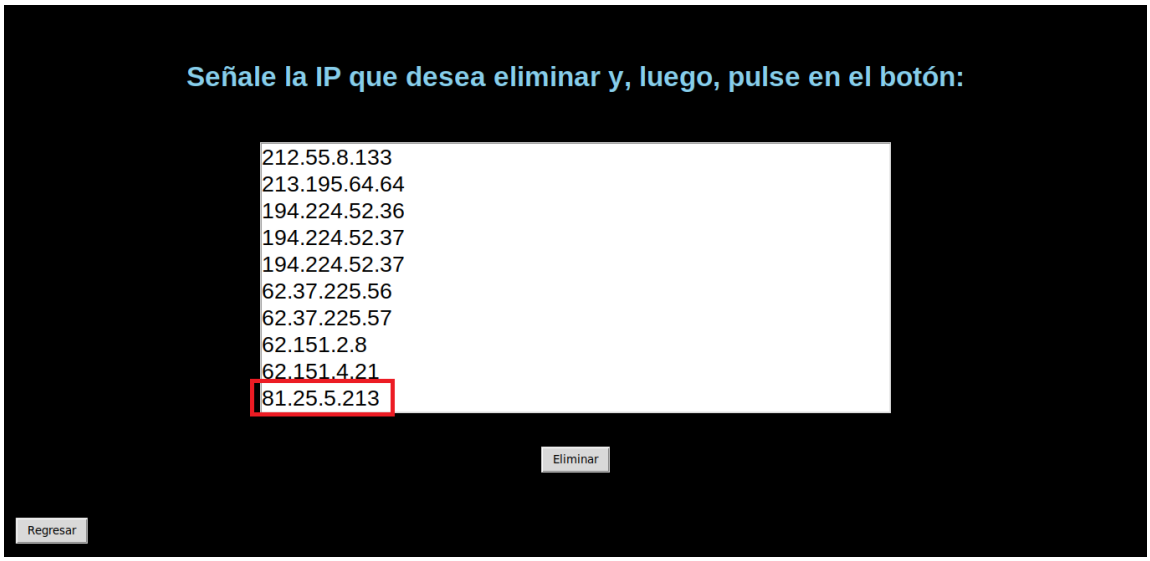

FIGURA C.6.2

c. Clicar en *Eliminar*.

d. Estando de nuevo en la pantalla inicial, clicar en el botón *Eliminar IP*.

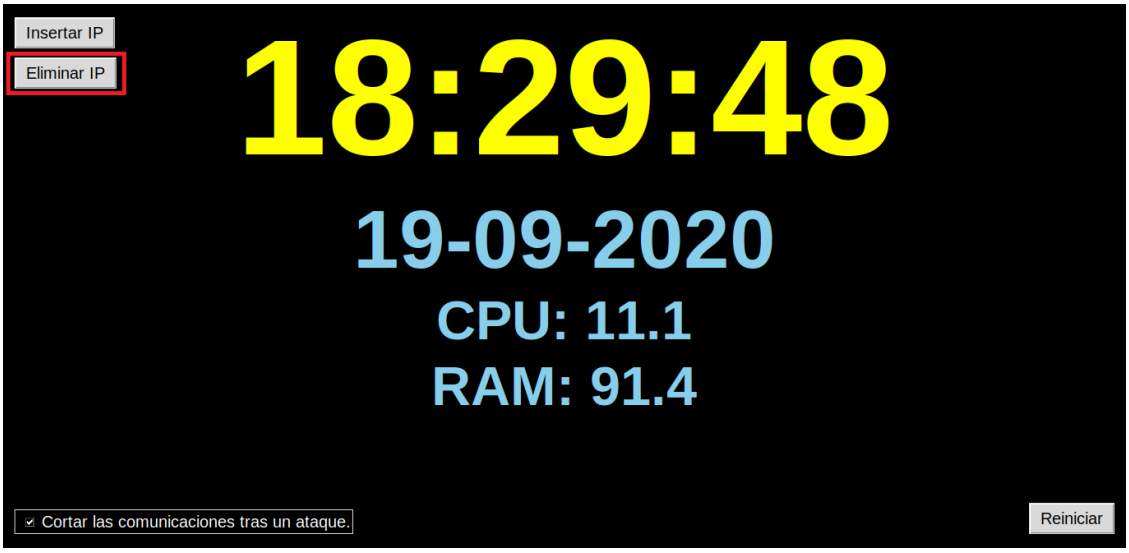

FIGURA C.6.3

e. En la lista buscar la IP 81.25.5.213.

f. Verificar que la IP no aparece en la lista y ha sido eliminado con éxito.

7. Intentar eliminar una IP sin seleccionar una de la lista

Resumen

Comprobar que al intentar eliminar una dirección IP sin elegir ninguna, el *Cibercanario* muestra un mensaje de error.

Requisitos

- Dispositivo *Cibercanario* correctamente instalado y desplegado.

Procedimiento

g. Estando en la pantalla inicial, clicar en el botón *Eliminar IP*.

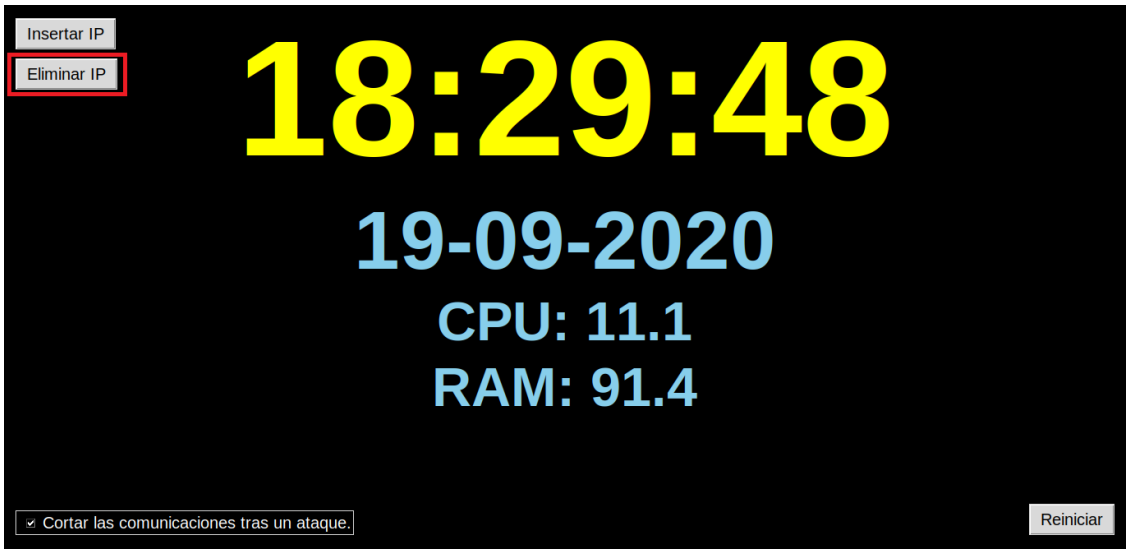

FIGURA C.7.1

h. No seleccionar ninguna IP de la lista y clicar directamente en *Eliminar*.

i. Verificar que aparece un mensaje de error.

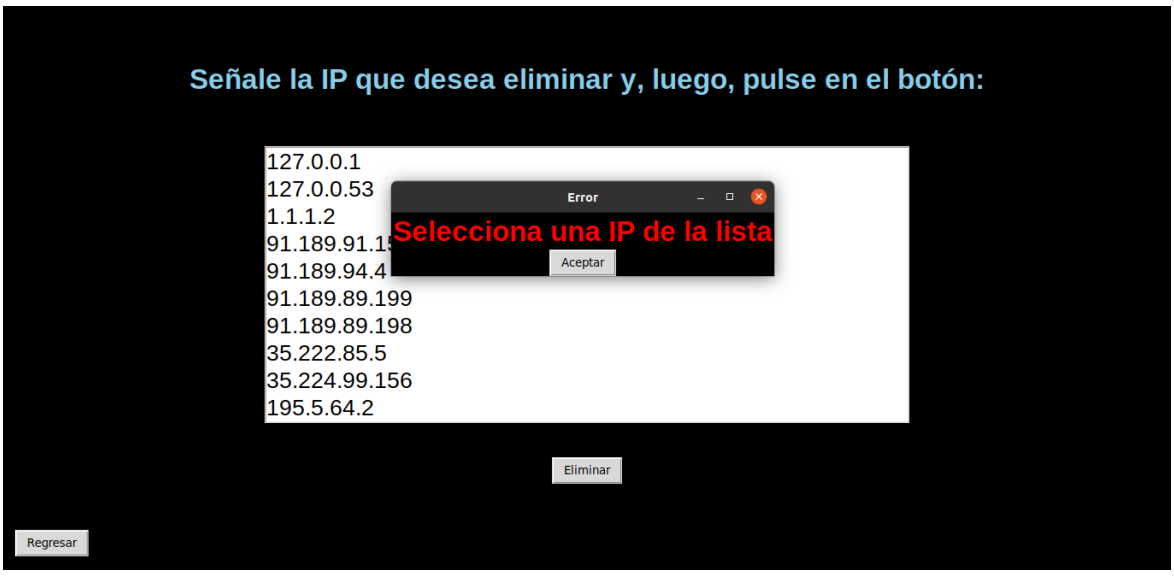

FIGURA C.7.2

8. El dispositivo recibe un ataque y cierra las comunicaciones

Resumen

Comprobar que al recibir el dispositivo un ataque de cualquier tipo, este corta las comunicaciones y rechaza cualquier paquete. Como todos los detectores funcionan de la misma manera se hará la prueba realizando un escaneo de puerto dentro de la red local.

Requisitos

- Dispositivo *Cibercanario* correctamente instalado y desplegado (IP 192.168.1.53).
- Sistema dentro de la red del *Cibercanario* con NMAP, tcpdump y Wireshark instalado a la que llamaremos *Evaluador* (se usará para la prueba una máquina Kali Linux con la IP 192.168.1.61)

Procedimiento

a. El *Cibercanario* deberá estar encendido y funcionando.

b. Desde la máquina *Evaluador*, abrir una terminal y lanzar el siguiente comando:

```
sudo tcpdump -ttvvX -i eth0 net 192.168.1.0/24 -w privateScn.pcap
```

c. Abrir una nueva terminal y lanzar el siguiente comando:

```
nmap -Pn 192.168.1.53
```

d. Verificar que realiza correctamente el escaneo y detecta los puertos abiertos.

```
           :~# nmap -Pn 192.168.1.53
Starting Nmap 7.70 ( https://nmap.org ) at 2020-09-20 13:36 CEST
Nmap scan report for 192.168.1.53
Host is up (0.033s latency).
Not shown: 998 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
8888/tcp open  sun-answerbook
```

FIGURA B.8.1

e. Verificar que el *Cibercanario* ha detectado el ataque.

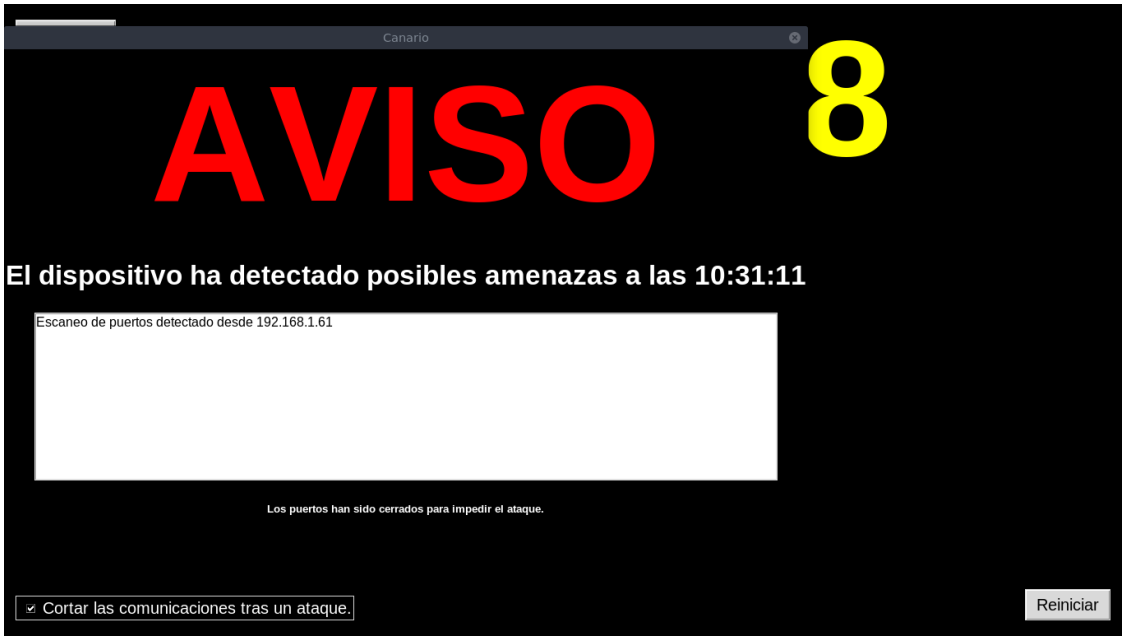

FIGURA C.8.2

f. De nuevo en la maquina *Evaluador*, lanzar nuevamente el comando del paso c.

g. Verificar que, en esta ocasión, el escaneo muestra que todos los puertos se encuentran filtrados.

```
          :~# nmap -Pn 192.168.1.53
Starting Nmap 7.70 ( https://nmap.org ) at 2020-09-20 13:36 CEST
Nmap scan report for 192.168.1.53
Host is up (0.13s latency).
All 1000 scanned ports on 192.168.1.53 are filtered
```

FIGURA C.8.3

h. Verificar que el *Cibercanario* no ha detectado un nuevo ataque.

i. Cortar la ejecución del *tcpdump* y abrir el archivo generado con Wireshark.

j. Verificar que, en el primer ataque, antes de cortar las comunicaciones, el *Cibercanario* responde correctamente al escaneo de puerto (en gris se observan los paquetes usados para escanear los puertos y en rojo los ACK con los que responde el *Cibercanario*).

ip.addr == 192.168.1.61 and ip.addr == 192.168.1.53 Expressior

No.	Time	Source	Destination	Protocol	Length	Info
22	10.151165	192.168.1.61	192.168.1.53	TCP	58	53650 → 135 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
23	10.151243	192.168.1.61	192.168.1.53	TCP	58	53650 → 1723 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
24	10.151279	192.168.1.61	192.168.1.53	TCP	58	53650 → 443 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
25	10.151321	192.168.1.61	192.168.1.53	TCP	58	53650 → 3389 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
26	10.151355	192.168.1.61	192.168.1.53	TCP	58	53650 → 21 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
27	10.151388	192.168.1.61	192.168.1.53	TCP	58	53650 → 8888 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
28	10.151426	192.168.1.61	192.168.1.53	TCP	58	53650 → 993 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
29	10.151461	192.168.1.61	192.168.1.53	TCP	58	53650 → 25 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
30	10.151511	192.168.1.61	192.168.1.53	TCP	58	53650 → 1025 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
31	10.152139	192.168.1.61	192.168.1.53	TCP	58	53650 → 143 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
32	10.160073	192.168.1.53	192.168.1.61	TCP	60	135 → 53650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
33	10.160103	192.168.1.53	192.168.1.61	TCP	60	1723 → 53650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
34	10.160116	192.168.1.53	192.168.1.61	TCP	60	443 → 53650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
35	10.160127	192.168.1.53	192.168.1.61	TCP	60	3389 → 53650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
36	10.161125	192.168.1.53	192.168.1.61	TCP	60	21 → 53650 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37	10.164881	192.168.1.53	192.168.1.61	TCP	60	8888 → 53650 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460

FIGURA C.8.4

k. Verificar que, en el segundo ataque, tras cortar las comunicaciones, los paquetes usados para escanear los puertos siguen llegando, pero el *Cibercanario* nunca responde a ellos.

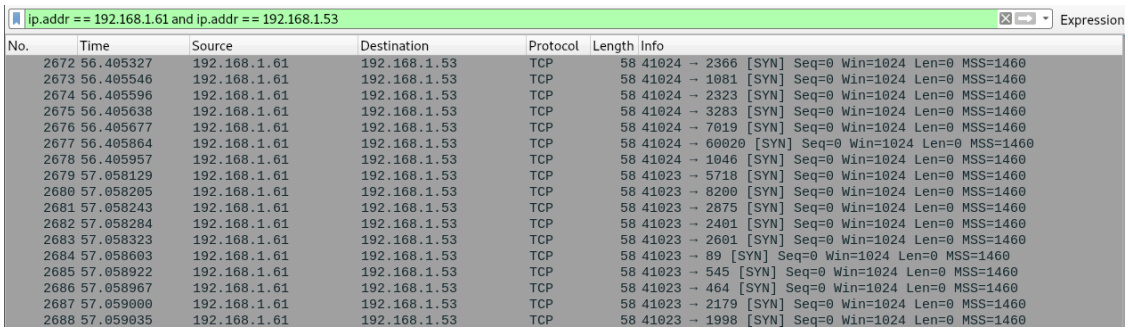
ip.addr == 192.168.1.61 and ip.addr == 192.168.1.53
Expression
No. Time Source Destination Protocol Length Info
2672 56.405327 192.168.1.61 192.168.1.53 TCP 58 41024 → 2366 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
2673 56.405546 192.168.1.61 192.168.1.53 TCP 58 41024 → 1081 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
2674 56.405596 192.168.1.61 192.168.1.53 TCP 58 41024 → 2323 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
2675 56.405638 192.168.1.61 192.168.1.53 TCP 58 41024 → 3283 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
2676 56.405677 192.168.1.61 192.168.1.53 TCP 58 41024 → 7019 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
2677 56.405864 192.168.1.61 192.168.1.53 TCP 58 41024 → 60020 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
2678 56.405957 192.168.1.61 192.168.1.53 TCP 58 41024 → 1046 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
2679 57.058129 192.168.1.61 192.168.1.53 TCP 58 41023 → 5718 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
2680 57.058205 192.168.1.61 192.168.1.53 TCP 58 41023 → 8200 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
2681 57.058243 192.168.1.61 192.168.1.53 TCP 58 41023 → 2875 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
2682 57.058284 192.168.1.61 192.168.1.53 TCP 58 41023 → 2401 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
2683 57.058323 192.168.1.61 192.168.1.53 TCP 58 41023 → 2601 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
2684 57.058603 192.168.1.61 192.168.1.53 TCP 58 41023 → 89 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
2685 57.058922 192.168.1.61 192.168.1.53 TCP 58 41023 → 545 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
2686 57.058967 192.168.1.61 192.168.1.53 TCP 58 41023 → 464 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
2687 57.059000 192.168.1.61 192.168.1.53 TCP 58 41023 → 2179 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
2688 57.059035 192.168.1.61 192.168.1.53 TCP 58 41023 → 1998 [SYN] Seq=0 Win=1024 Len=0 MSS=1460

FIGURA C.8.5

APÉNDICE D: REQUISITOS HARDWARE Y SOFTWARE

En los capítulos que recogen el estudio y desarrollo del proyecto se ha profundizado en los motivos por los cuales se han elegido los requisitos. En este apéndice se recopilarán los requisitos tanto del hardware como del software.

Con respecto al software, los requisitos en los que nos centraremos serán aquellos relacionados, principalmente, con la interfaz de usuario y, por lo tanto, son esencialmente funcionales:

- *El usuario podrá ver un aviso por pantalla cuando el dispositivo sea objetivo de un escaneo de puertos.*
- *El usuario podrá ver un aviso por pantalla cuando el dispositivo reciba paquetes de tipo TCP, UDP o ICMP proveniente de direcciones IP no conocidas.*
- *El usuario podrá ver la hora a la que se ha recibido el ataque.*
- *El usuario podrá observar el estado de las características del sistema (porcentaje de RAM y CPU usados).*
- *El usuario podrá elegir si cerrar las conexiones del dispositivo automáticamente después de recibir un ataque o mantenerlos abiertos.*
- *El usuario podrá añadir IPs que no supongan un riesgo a la lista blanca (se explicará el significado de este término en el apartado de diseño) del Cibercanario.*
- *El usuario podrá eliminar IPs de la lista blanca del Cibercanario.*

También existen algunos requisitos no funcionales, todos ellos relacionados con la seguridad del dispositivo:

- *El dispositivo debe ser robusto ante ataques.*
- *El dispositivo debe ser capaz de funcionar a pesar de las actualizaciones.*
- *El dispositivo debe simular su pertenencia al entorno en el que se encontrara instalado.*

Con respecto al hardware del dispositivo, también existen requisitos sobre las condiciones que debe cumplir este para ser un candidato para alojar nuestra aplicación:

- *El hardware debe ser lo más barato posible.*
- *El hardware debe consumir la menor cantidad de electricidad posible teniendo la potencia suficiente para mover la aplicación.*
- *El hardware debe presentar las mínimas (ninguna a poder ser) vulnerabilidades posibles.*

Bibliografía

[1] **2019 Vodafone IoT Barometer** (https://www.vodafone.com/business/news-and-insights/white-paper/vodafone-iot-barometer-2019)

[2] https://csrc.nist.gov/CSRC/media/Publications/white-paper/2018/10/17/iot-trust-concerns/draft/documents/iot-trust-concerns-draft.pdf

[3] https://www.ibm.com/blogs/industries/little-known-story-first-iot-device/

[4] https://www.incibe-cert.es/blog/honeypots-industriales

[5] https://www.incibe-cert.es/blog/iot-protocolos-comunicacion-ataques-y-recomendaciones

[6] https://www.kelltontech.com/kellton-tech-blog/internet-of-things-protocols-standards

[7] **The IoT Hacker's Handbook - Aditya Gupta** (https://www.oreilly.com/library/view/the-iot-hackers/9781484243008/)

[8] Especificaciones de la *Figura 6*:

- ESP32: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf
- Raspberry Pi 4: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/
- ROCK64: https://wiki.pine64.org/index.php/Rock64#CPU_Architecture
- BeagleBone Black: https://beagleboard.org/Products/BeagleBone%20Black
- Orange Pi 4B: http://www.orangepi.org/Orange%20Pi%204B/
- Onion Omega2+: https://docs.onion.io/omega2-docs/omega2p.html

[9] https://www.espressif.com/en/news/ESP32_FIA_Analysis

[10] https://snapcraft.io/

[11] https://docs.python.org/3/library/tkinter.html#module-tkinter

[12] https://pypi.org/project/IPy/

[13] https://pypi.org/project/psutil/

[14] https://pypi.org/project/webthing/

[15] https://iot.mozilla.org/gateway/

[16] https://discourse.mozilla.org/t/an-important-update-on-mozilla-webthings/67764

[17] https://www.nics.uma.es/pub/papers/RubioRomanAlcarazZhang2018.pdf **(Apartados 3.1 y 3.2)**

[18] https://iot.mozilla.org/framework/

[19] https://iot.mozilla.org/wot/

[20] https://www.raspberrypi.org/downloads/

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga