Facultad de Ciencias

APARATO DETECTOR DE DISPOSITIVOS USB MALICIOSOS
(Detector device for malicious USB devices)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERIA INFORMATICA

Autor: Pablo Izquierdo González

Director: Esteban Stafford

Junio 2022

Agradecimientos

Me gustaría agradecérselo, a mi familia por haberme apoyado tanto durante la realización de este trabajo, y en especial a mi madre. También me gustaría agradecerle a mi director el haberme guiado tan bien, y animado a realizar una idea propia.

Resumen

Desde hace años las tecnologías de información y comunicación ha revolucionado el mundo. Esto tiene una parte negativa, ya que antiguamente no nos podíamos ni imaginar a los peligros a los que estaríamos expuestos hoy en día debido a estas tecnologías. Hay un esfuerzo muy grande por prevenir y educar a la sociedad en algunos de estos peligros, pero hay un tipo de ataque que ha pasado desapercibido.

Los dispositivos USB son un medio de transmisión de código malicioso, que a pesar de no ser el mas común, es conveniente protegerse. Debido a esto, este trabajo de fin de grado pretende crear una herramienta capaz de detectar y prevenir al usuario de dispositivos USB que puedan vulnerar su ordenador.

El resultado del trabajo es un dispositivo que un usuario puede utilizar para evaluar la peligrosidad de dispositivos USB antes de enchufarlos a su equipo. El dispositivo detecta algunos ataques realizados a través de puertos USB. Consiguiendo proteger al usuario de estos dispositivos maliciosos.

Palabras clave: USBGuardian, Protección USB, Ataques USB, prevenir ataques por USB, dispositivos USB maliciosos, detectar ataques por USB, herramienta seguridad USB, securizar puertos USB, proteccion frente a Rubber Ducky, proteccion contra dispositivos USB maliciosos.

Abstract

For years, information and communication technologies have revolutionised the world. This has a negative side, as in the past we could not even imagine the dangers we would be exposed to today because of these technologies. There is a great effort to educate society on some of these dangers, but there is one type of attack that has gone unnoticed.

USB devices are a means of transmitting malicious code, which, although not the most common, is worth protecting against. Because of this, this final degree project aims to create a tool capable of detecting and preventing the user from USB devices that could breach their computer.

The result of the work is a device that a user can use to evaluate the danger of USB devices before plugging them into their computer. The device detects some attacks carried out through USB ports. It is able to protect the user from these malicious devices.

Keywords: USBGuardian, USB Protection, USB Attacks, prevent USB attacks, malicious USB devices, detect USB attacks, USB security tool, secure USB ports, protection against Rubber Ducky, protection against malicious USB devices.

Índice general

1. Introducción **1**

1.1. Motivación 1

1.2. Objetivos 2

1.3. Metodología 2

2. Tecnologías y Herramientas **7**

2.1. Hardware 7

2.2. Software 8

2.2.1. Linux 10

2.3. Expecificacion USB 12

3. Diseño **19**

3.1. Introduccion 19

3.2. Requisitos 24

4. Implementación **25**

4.1. Introduccion 25

4.2. Diferenciar entre los diferentes dispositivos conectados 25

4.3. Proteger el sistema y detectar la mayor cantidad de ataques 27

4.4. Minimizar el tiempo de arranque 31

4.5. Interfaz con luces 32

4.6. Facil instalacion 33

5. Pruebas **35**

5.1. Rubber Ducky 35

5.2. Dispositivos probados 36

6. Conclusiones **39**

6.1. Objetivos conseguidos 39

6.2. Trabajos futuros 39

6.3. Reflexión personal 40

Referencias bibliográficas **41**

Introducción

1.1 Motivación

Hoy en día todos podemos ser víctimas de algún tipo de estafa online, cosa que mediante los medios de comunicación se ha intentado advertir a la población acerca de correos electrónicos sospechosos, llamadas telefónicas falsas, etc. Por otro lado, también se han difundido técnicas para navegar por internet de forma segura, usar antivirus y tomar precauciones en la red. Sin embargo, los ciberdelincuentes también tienen otras maneras de llegar hasta ti, que aunque sean menos utilizadas, no hay que pasarlas por alto.

En algún momento todos hemos utilizado un pendrive. Un dispositivo muy útil para guardar archivos y transferirlos a otro ordenador, ya sea para imprimir algo en una imprenta o para pasar un archivo a un conocido. En las últimas dos décadas estas memorias USB han dominado el mercado, relegando a otras tecnologías como los disquetes al olvido. Con el tiempo fue mejorando la tecnología, permitiendo mayor capacidad a menor coste, lo que provocó que las marcas comprasen esos dispositivos muy baratos para emplearlos como publicidad.

El hecho de que sean tan usados y comunes hoy en día, permite a los ciberdelincuentes utilizarlos como medio de transmisión de malware. El malware es un termino genérico para definir al software con una intención maliciosa. Para ser más concretos, el ataque más simple que se realiza con estos dispositivos es dejar un pendrive "olvidado" en un sitio en que la víctima pueda encontrarlo. La vulnerabilidad de este ataque es principalmente el ser humano, ya que sea por maldad (para quedarse el pendrive) o bondad (intentar averiguar quién es el propietario) la víctima conecta el pendrive a su ordenador. Al hacer esto, se ejecuta el código malicioso y pone en riesgo la seguridad del ordenador. Si el ataque tiene éxito, el atacante puede ser capaz de espiar el contenido del ordenador, consolidar un acceso al mismo o en el peor de los casos encriptar el contenido y pedir un rescate económico.

1.2 Objetivos

Este Trabajo Fin de Grado propone el diseño e implementación de un aparato autónomo, que disponga de un puerto al que se le puedan enchufar dispositivos USB. Los cuales tendrán que superar una serie de filtros que aseguren que el dispositivo no contiene malware ni está preparado para transmitir código malicioso que vulnere la seguridad de un ordenador. Es importante que el aparato sea a su vez invulnerable a este tipo de riesgos de seguridad en puertos USB. Además, el aparato mostrará si el dispositivo introducido presenta alguna amenaza de seguridad al usuario a través de una interfaz sencilla con luces.

Dada la variedad de ataques posibles mediante este tipo de puerto, simulando dispositivos USB, se pretende crear unos filtros que permitan detectar, bloquear y notificar al usuario la mayor cantidad de estos ataques.

1.3 Metodología

Una vez ya establecido el conjunto de objetivos que buscará cumplir este trabajo, es necesario establecer el procedimiento que se seguirá para alcanzar los objetivos de manera satisfactoria. Por ello, he optado por un ciclo de vida de proyecto que sigue un modelo iterativo incremental. Éste es un enfoque intermedio que combina las ventajas de los modelos de cascada y de desarrollo evolutivo.

La metodología de cascada fue propuesta por Winston W. Royce en 1970 en su ensayo titulado *Managing the development of large software systems* [31]. Wintom W. Royce propone un modelo compuesto por siete fases que se pueden observar en la Figura §1.1. Este método es rígido y los cambios importantes son costosos.

En la práctica, a este modelo se le aplican algunas modificaciones para reducir el número de fases. En la publicación [34] mencionan solo 5 de estas fases. En la fase de análisis y definición de requerimientos, se está en contacto con los clientes y usuarios finales para determinar las metas de la aplicación, además de determinar los servicios que debe proporcionar el sistema y sus restricciones. La identificación y descripción de los subsistemas que constituyen el sistema, así como la relación entre estos, entra dentro de la fase de diseño del sistema y del software. Después de estas fases se pasa a codificar y probar los diferentes subsistemas por separado, lo que forma la fase de implementación y validación. La etapa de integración y validación del sistema consiste en anexar los programas o unidades individuales de programas y formar así un

sistema completo que cumpla los requerimientos del software. Una vez las pruebas sean satisfactorias, se entrega al cliente. Como última etapa, tenemos que instalar y poner en funcionamiento el sistema, además de realizar mantenimientos periódicos para garantizar la corrección de errores no encontrados en anteriores etapas y mejorar la implementación del sistema.

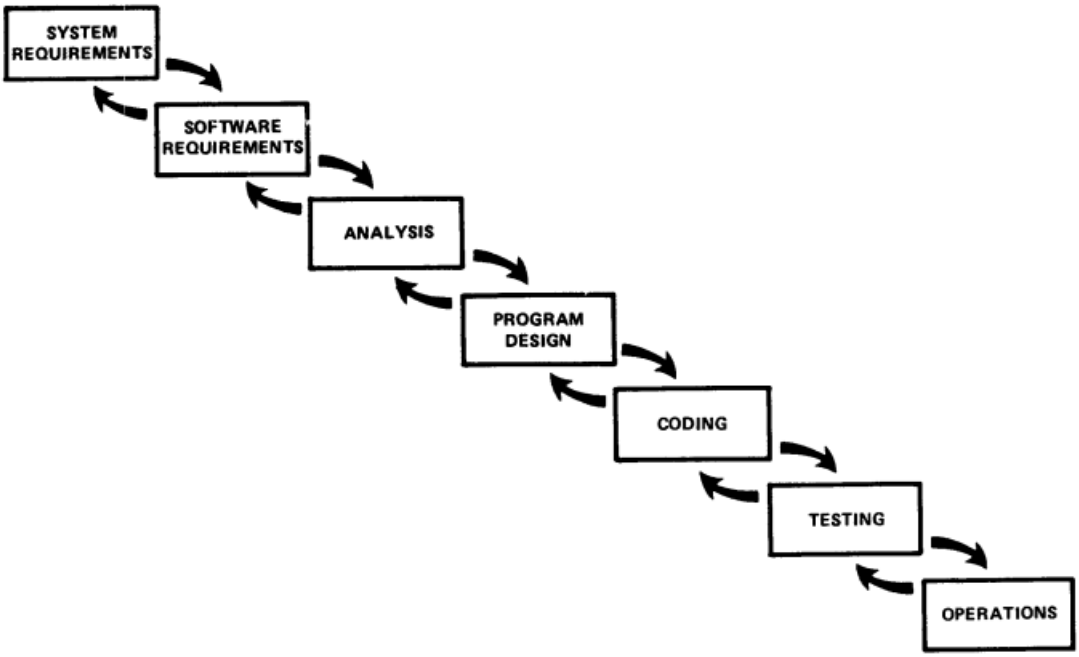

Figura 1.1: Representacion de la metodologia de cascada propuesta por Winston W. Royce [31]

El método evolutivo se caracteriza por permitir a los ingenieros de software desarrollar una implementación inicial, teniendo en cuenta los comentarios de los usuarios, y mejorar el sistema a través de diferentes versiones consiguiendo así cada vez un sistema más adecuado (Figura §1.2) [34].

El enfoque evolutivo suele ser más efectivo que el enfoque en cascada, porque satisface los requisitos inmediatos de los clientes. Un desarrollo de software basado en un enfoque evolutivo tiene la ventaja de que la especificación tiene la posibilidad de ampliarse de forma incremental. Sin embargo, el enfoque evolutivo tiene dos problemas, que el proceso no es visible y es necesario realizar entregas frecuentemente para medir el progreso, y que si el sistema se desarrolla rápidamente no es rentable detallar cada versión del mismo. Además, otro problema es que la estructura a menudo es deficiente, ya que los cambios continuos la perjudican y cada vez es más difícil y costoso realizar cambios y añadir características.

Figura 1.2: Representacion de la metodologia evolutiva [34]

La metodología en cascada mencionada anteriormente necesita que los clientes cumplan un conjunto de exigencias antes de entrar en la etapa de diseño y que el diseñador realice una serie de estrategias específicas para la realización del diseño, adaptándolo a las necesidades del cliente antes de la implementación. La modificación de los requerimientos conlleva rehacer el trabajo de captura de éstos para las etapas de diseño e implementación. Al ser etapas diferenciadas, deben dar lugar a sistemas bien documentados capaces de permitir cambios. Por otro lado, un enfoque de desarrollo evolutivo, mencionado anteriormente, posibilita que los requisitos y las decisiones de diseño se retrasen, pero dan lugar a un software que puede estar frágilmente estructurado lo que complica comprenderlo y mantenerlo.

Un proceso de *desarrollo iterativo incremental*, como el seguido para la realización de este trabajo, tiene como objetivo entregar a los clientes un sistema funcional. Esto implica que se comenzará con los requisitos que mejor se comprendan y tengan mayor prioridad. Los clientes reconocerán, a grandes rasgos, los servicios que proporcionará el sistema, cuales son más importantes y cuáles menos. En este caso, yo he sido el cliente que ha identificado y determinado los requisitos y necesidades que el sistema debe cumplir. Posteriormente, se definen varios incrementos en donde cada uno proporciona una funcionalidad al sistema. Los servicios que constituyen los incrementos están determinados por la prioridad del servicio, así se garantiza que se realizan primero los que tienen mayor prioridad. Un ejemplo de este proceso se puede observar en la Figura §1.3.

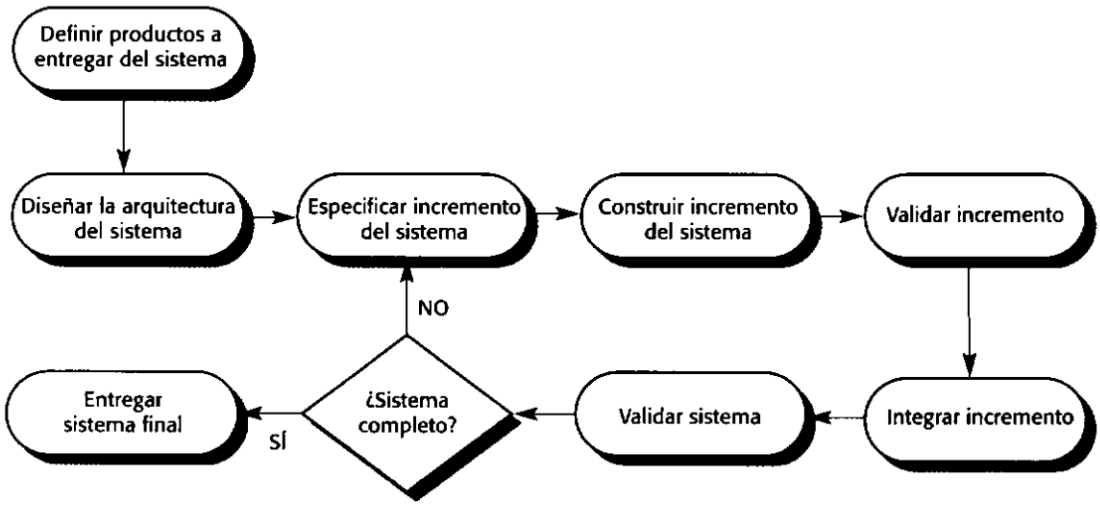

Figura 1.3: Un proceso interativo incremental [34]

La metodología iterativa incremental, tiene varios beneficios. Los clientes no tienen que esperar a la entrega del sistema completo para poder trabajar con él. Al satisfacerse los requerimientos más necesarios al inicio, los clientes lo podrán utilizar como prototipo y proporcionar retroalimentación sobre los requisitos para posteriores incrementos del sistema. Además, al realizarse los servicios más importantes al comienzo, a estos se les realizan multitud de pruebas, lo que provoca que sea menos probable que se encuentren fallos en el producto final.

Tecnologías y Herramientas

2.1 Hardware

A la hora de hacer la elección del hardware utilizado para el proyecto se tuvieron en cuenta diversos factores, como el coste, capacidad de ampliación con periféricos, disponibilidad, especificaciones del hardware, sistemas operativos soportados y facilidad de uso. Tras sopesar diferentes opciones, opté por utilizar una Raspberry Pi 1B+ [26] [25]. La decisión de utilizar la Raspberry pi 1B+ (Figura §2.1) en vez de alguna versión posterior se debe a su disponibilidad y el conocimiento de su funcionamiento, al haber trabajado con ella anteriormente en el grado. Sin embargo, si la configuración de los pines no ha cambiado entre las versiones no debería haber problema en utilizar una versión posterior. Además, la Raspberry Pi tiene una serie de ventajas que la hace la mejor opción para el proyecto.

- Bajo coste
- Capacidad de correr un sistema operativo GNU/Linux
- Disponibilidad de periféricos
- Tamaño reducido
- Puerto Ethernet para la comunicación por ssh y conexión a internet

Como hemos mencionado antes una de las ventajas es la disponibilidad de periféricos. Esto es importante porque uno de los objetivos es notificar mediante leds al usuario información relativa al estado del dispositivo USB. Para realizar esta tarea he utilizado una placa Berry-clip §2.2, que incorpora tres tipos de dispositivos: seis LED (dos rojos, dos amarillos y dos verdes), dos pulsadores y un zumbador. Esta placa se conecta directamente sobre el conector GPIO. Una vez conectada la placa, será necesario programar la misma para que realice las operaciones de entrada/salida deseadas.

Raspberry pi 1B+	
Especificación	
Chip	Broadcom BCM2835 SoC
Arquitectura del nucleo	ARM11
CPU	ARM1176JZFS Low Power
Velocidad de reloj	700 MHz
GPU	Dual Core VideoCore IV Multimedia Co-Processor
Memoria	512MB SDRAM
Dimensiones	85 x 56 x 17mm
Enegia	Micro USB socket 5V 2A
Conexiones	
Ethernet	10/100 BaseT Ethernet socket
Salida de Video	HDMI (rev 1.3 & 1.4)
Salida de Audio	3.5mm jack, HDMI
USB	4 x USB 2.0 Connector
Conector GPIO	40-pin 2.54 mm (100 ml) expansion header 2x20 strip
Conector de Camera	15-pin MPI Camera Serial Interface (CSI-2)
JTAG	Not populated
Conector de Pantalla	Display Serial Interface (DSI)
Ranura tarjeta de memoria	SDIO

Cuadro 2.1: Especificación Raspberry pi 1B+ [29] [27]

2.2 Software

Una vez ya elegido el hardware, se presentaban diferentes opciones para el sistema operativo utilizado en la Raspberry Pi. Entre las diferentes opciones que yo he tenido en consideración están: RISC OS, Raspbian y Arch Linux. Para la elección he tenido en cuenta aspectos como prestaciones, facilidad en su uso, conocimiento del sistema, seguridad y protección del sistema, similitud con los sistemas mas utilizados, licencia gratuita y que sea ligero y eficiente.

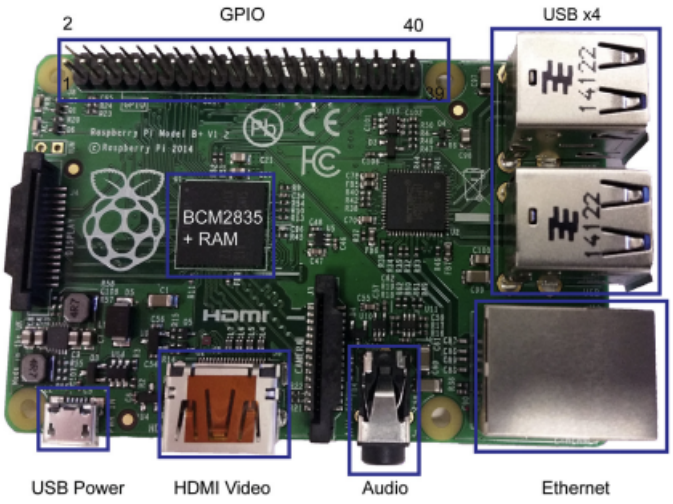

Figura 2.1: Raspberry pi 1B+ [33]

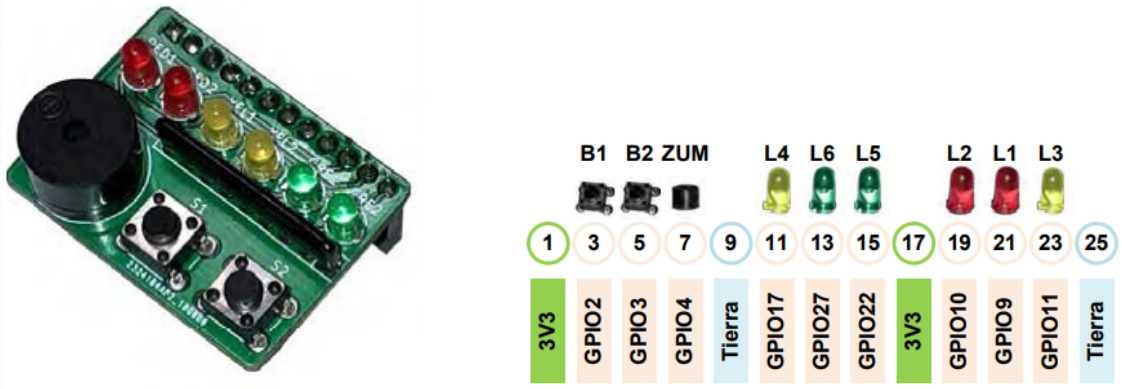

Figura 2.2: Berry-clip y su conexion al conector GPIO

Finalmente, me he decidido por utilizar Raspbian, que es una adaptación no oficial de Debian (una distribución de Linux §2.2.1) con una configuración optimizada para ejecutarla en la Raspberry Pi. Lo que mejora el rendimiento mediante el uso de instrucciones avanzadas de CPU [1]. A pesar de que RISC OS y Arch Linux, cumplen con muchas de las características propuestas. Por un lado Arch Linux no esta optimizado para su uso en la Raspberry Pi, y RISC OS es un sistema que se puede considerar experimental, y que no cuenta con una comunidad de desarrolladores tan amplia como Linux, en consecuencia su soporte para dispositivos modernos es limitada.

Concretamente he utilizado la imagen de Raspbian Lite de 32 bits, esto se debe a que utilizo la Raspberry Pi 1B+ y no soporta sistemas operativos de 64 bits . Esta imagen es una versión de Debian 11 (bullseye) con una versión de kernel 5.15 y un tamaño de 297 MB. [30] [28].

2.2.1 Linux

Dado que se ha escogido un sistema operativo basado en Linux, merece la pena hacer una pequeña presentación de éste y los subsistemas más importantes para la realización de este trabajo.

Linux es un kernel diseñado por Linus Torvalds en 1991, y dió lugar a una revolución debido a su carácter de código abierto. Está diseñado para que sea accesible al público, de manera que cualquiera pueda modificarlo y distribuirlo. Linux se basa en los principios y el diseño de Unix, un sistema operativo desarrollado en 1969 en los laboratorios Bell. Debido a las características de este sistema operativo, se ha convertido en la principal herramienta utilizada en los servidores de todo el mundo [15].

Debido a su flexibilidad, Linux sirve para casi cualquier tipo de iniciativa de *Tecnologia de la Información* (*TI*), ya sea contenedores, aplicaciones cloud, seguridad. Es por esta última por la que he decidido utilizar un sistema operativo basado en Linux, que me proporciona casos prácticos, funciones, dispositivos, distribuciones útiles para el desarrollo de mi proyecto.

El kernel Linux tiene un diseño modular. Además, muchas de las distribuciones de Linux mantienen esta característica, ya que permite cargar un kernel mínimo en memoria a la ahora de arrancar y posteriormente ir cargando de forma dinámica en memoria otros módulos cuando sean requeridos (ya sea por el usuario o por el sistema operativo), que proporcionan diferentes funcionalidades. [20]

Uno de los principales objetivos que tiene el kernel es ocultar los detalles del hardware bajo una interfaz de programación abstracta de alto nivel. Esta interfaz proporciona las siguientes características hardware de bajo nivel. Gestiona y abstrae los dispositivos hardware, realiza la gestión de procesos e hilos, permite la comunicación entre procesos y la compartición de tiempo de CPU. También, se encarga de gestionar la memoria, tanto protegerla como hacerla parecer ilimitada a los procesos. Facilita la entrada/salida y proporciona funciones de encendido/apagado, temporizadores, operaciones multitarea, etc [22].

System management daemon

Linux es el corazón del sistema operativo Raspbian. Pero al rededor de éste se encuentra una cantidad innumerable de utilidades y subsistemas que complementan su funcionalidad. Uno de éstos es Systemd, que es un conjunto de programas para gestionar el arranque de servicios y multitud de otros aspectos básicos para un sistema Linux. Facilita un sistema y un gestor de servicios que se ejecuta con PID 1, del cual cuelgan el resto de los servicios del sistema. [35] Además, establece muchas características importantes de seguridad, utilizadas para aislar servicios y aplicaciones [3].

Las principales características de Systemd relacionadas con este trabajo se enumeran a continuación [22]:

- Aporta una paralelización muy agresiva para el arranque de servicios.
- Aprovecha la activación de sockets y D-Bus para la inicialización de los servicios, permitiendo su inicio bajo demanda.
- Mantiene un seguimiento de los procesos utilizando grupos de control de Linux.

Systemd consta de diferentes unidades, que son información o configuración codificada en un fichero para cualquier recurso o servicio. Existen diferentes tipos de ficheros de Systemd, como *target* que sirve para crear un árbol de nodos para el arranque, *automount* que al tenerlo ya no sería necesario el fichero *fstab* donde se especifica el lugar de montaje de los discos, etc. Pero los que más he utilizado para la realización de este proyecto son el *timer* que establece un temporizador que determina el momento en el que el servicio ejecuta alguna tarea, y el *service* que se usa para ejecutar alguna tarea especifica. Un ejemplo podría ser el servicio *dhcpcd@.service* que mantiene la configuración de red del sistema de manera dinámica a través del protocolo DHCP.

Udev

El subsistema udev gestiona los dispositivos que usa el kernel Linux en su versión 2.6 como sucesor de devfs. Está compuesto por servicios de kernel y el servicio udevd. En un sistema UNIX, existe un directorio llamado *dev* en la que se recogen todos los dispositivos del sistema. La función de udev es mantener esta lista de dispositivos en *dev* y realizar todas las acciones necesarias para que los dispositivos estén configurados correctamente, incluyendo la carga de módulos del kernel o firmware. Es capaz de hacer esto dinámicamente cuando el usuario agrega o quita dispositivos [39][37]. Este subsistema trabaja en espacio de usuario, no forma parte del kernel, y permitie al

administrador añadir reglas para ajustar la configuración de dispositivos. Las reglas se ejecutan en momentos clave del ciclo de vida de los dispositivos, como la conexión y desconexión. Además, estos eventos son manejados de forma concurrente lo que mejora potencialmente el rendimiento del sistema. El Objetivo principal de udev es actuar sobre la detección de periféricos y el hot-plugging sustituyendo a hotplug y hwdetect. [37]

Al ser un servicio del sistema, es arrancado por Systemd acorde al contenido del fichero *systemd-udevd.service*, pero su configuración se encuentra en *etc/udev/udev.conf*. Las reglas de udev pueden localizarse en */run/udev/rules.d*, */etc/udev/rules.d* o */lib/udev/-rules.d* la localización de éstas depende quién las crea. Si es un paquete de instalación aparecerán en /lib/udev/rules.d, mientras que las otras localizaciones facilitan que el administrador anule el comportamiento de las primeras. Si un archivo de reglas se repite en más de uno de estos directorios, se utiliza el primero y se ignoran los sucesivos. Cuando se añade o modifica una regla es necesario utilizar *udevadm* para recargar las reglas de systemd-udevd [38].

2.3 Expecificacion USB

Al ser el objeto de este trabajo la detección de dispositivos USB maliciosos, es conveniente conocer algunos detalles del estándar que les gobierna.

El USB es un estándar [7] que define un bus serie único para conectar multitud de periféricos al ordenador. La motivación original para la creación de este estándar viene de tres consideraciones principales. La necesidad de conectar el ordenador a la red telefónica. Su facilidad de uso y la expansión de los puertos del ordenador, ya que en la época en la que surgió la necesidad para crear este estándar, la adición de periféricos externos estaba limitada por la disponibilidad de puertos. Además, la falta de un bus de periféricos bidireccionales, de bajo coste y de media velocidad dificultó el desarrollo de periféricos como teléfonos, fax, escáneres, etc. El problema surge que cuando se añadía una nueva funcionalidad al PC, se definía una nueva interfaz para satisfacer las necesidades del dispositivo.

El estándar USB utiliza una topología de estrellas apiladas, que puede llegar a permitir el funcionamiento de 127 dispositivos al mismo tiempo. Para identificar los dispositivos se asignan direcciones únicas a cada uno. En el vértice de las capas, se encuentra un único anfitrión que controla todo el tráfico del bus. Además, esta topología permite que muchos dispositivos se conecten a un único bus lógico sin que otros que necesi-

tan pasar a través de más capas sufran retardo. Para crear esta topología se utilizan controladores, hubs o concentradores y periféricos.

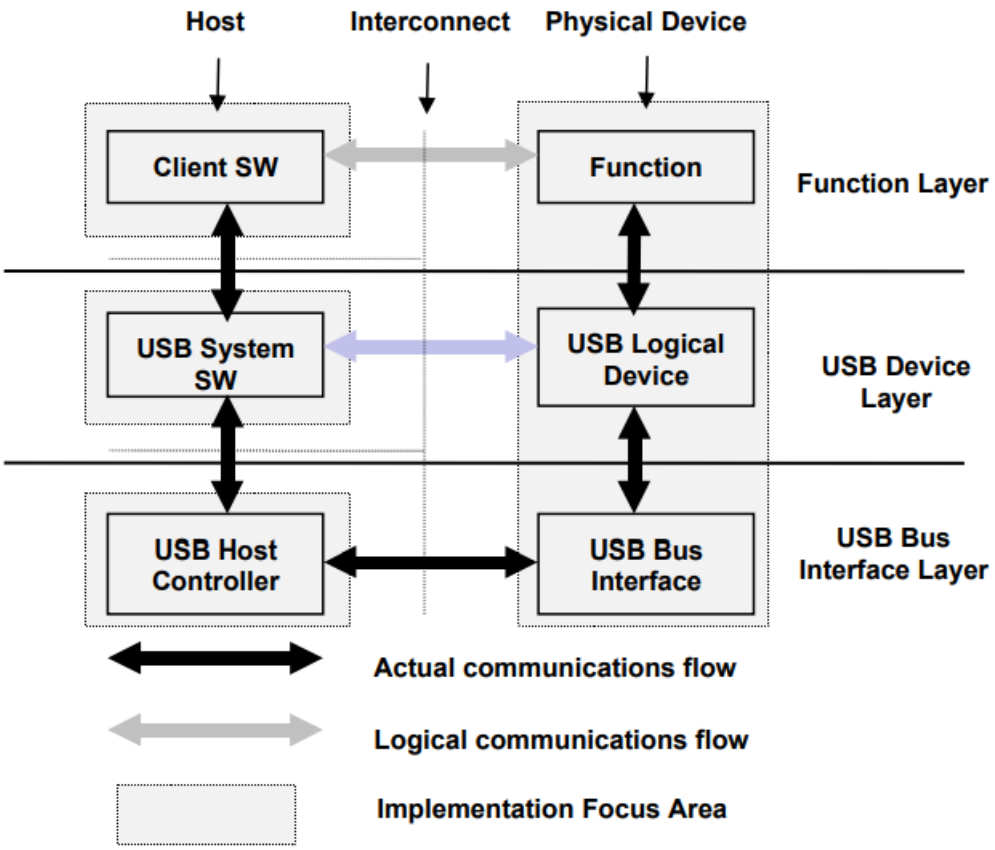

Figura 2.3: Modelo de intercomunicacion [7]

Los protocolos de comunicaciones del USB se organizan en tres capas mostradas en la Figura §2.5: la interfaz del bus, el sistema USB y la capa de función. La capa inferior es la encargada de transmitir y recibir paquetes. La intermedia enruta los datos entre la capa inferior y los distintos drivers de los dispositivos, o clientes, residentes en la última capa. Los clientes son las entidades de software responsables de interactuar y proporcionar la funcionalidad del dispositivo USB conectado. En el caso de un ratón, en esta capa se generan los eventos que describen el movimiento del dispositivo por una superficie.

Los dispositivos USB tienen varios estados. Algunos son visibles tanto para el USB como para el anfitrión, y otros son internos del dispositivo. Merece la pena dar una breve descripción de los más relevantes, que pueden verse en la Figura §2.4.

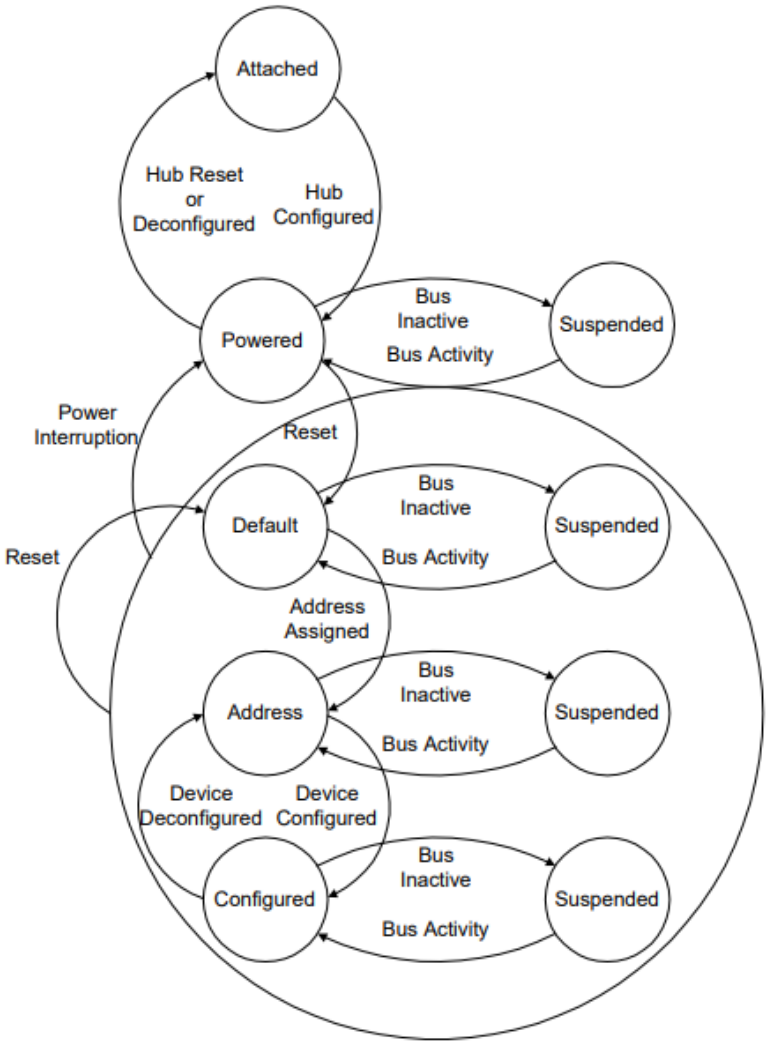

Figura 2.4: Diagrama de estados visibles del dispositivo [7]

- Attached (Conectado)
  Representa cuando un dispositivo está conectado al bus USB.

- Powered (Alimentado)
  Los USB pueden obtener la energía de una fuente externa (pila o batería) y/o la propia del bus USB. Los dispositivos no se consideran que estén en este estado hasta que se conectan al puerto a pesar de que ya dispongan de alimentación eléctrica. Si la configuración solo permite un modo de alimentación y ésta cambia, el dispositivo perderá su configuración y las direcciones asignadas volviendo al estado de alimentación.

  Un dispositivo USB debe ser capaz de admitir un direccionamiento en un periodo de tiempo determinado, desde que se le proporciona energía.

- Default (Por defecto)
  Una vez ya alimentado el dispositivo, este no debe responder a ninguna comunicación en el bus hasta que el anfitrión le haya comunicado un reinicio. Después de esto, el dispositivo es direccionable en la dirección por defecto. Una vez ya completado el proceso de reinicio, el dispositivo ya funcionará a la velocidad determinada por el dispositivo

- Address (Direccionamiento)
  Cada dispositivo recibe una dirección única después de la conexión o reinicio, además éstos mantienen su dirección mientras permanecen suspendidos. Ya sea utilizando la dirección por defecto o la dirección única asignada, los USB responden a las peticiones hechas por el anfitrión.

- Configured (Configuración)
  Antes de utilizar el bus USB el dispositivo ha de ser configurado. Esta configuración consiste en procesar correctamente una petición del anfitrión.

- Suspended (Suspensión)
  Para hacer un uso razonable de la energía, después de un tiempo determinado sin estar en uso, los dispositivos USB pasan al estado de suspensión, conservando su dirección y configuración.

  Los dispositivos deben estar en todo momento preparados para suspender su actividad, ya sea por la falta de actividad en el bus o por que el anfitión desactiva el puerto al que está conectado, lo que se denomina suspensión selectiva.

  Un dispositivo USB sale del estado de suspensión cuando detecta actividad en el bus o cuando el host acepta la solicitud para salir del modo de suspensión.

Después de conocer un poco la arquitectura de un USB vamos a continuar con un ejemplo de la implementación hardware y software, y poder ver cómo espera el host que funcione el dispositivo USB. El flujo de datos e interrelaciones de las comunicaciones del USB se muestran en la figura §2.3

La interfaz de bus USB es la que se encarga de manejar las capas eléctricas y de protocolo. Tanto el host como el dispositivo disponen del *Serial Interface Engine* (*SIE*), que es el componente encargado de manejar la serialización y deserialización de las transmisiones USB. Además en esta capa el host tiene responsabilidades adicionales por ello cuenta con el controlador del anfitrión (Host Controller). Este controlador es utilizado para administrar las transferencias de datos entre el anfitrión y los dispositivos, también su definición del hardware afectará a la interfaz entre el sistema USB

y el Host controller. Uno de los objetivos principales es realizar la traducción entre el resultado que recibe la capa de cliente, la transferencia de datos y las transacciones USB, además de ser responsable de gestionar los recursos del USB tal como el ancho de banda y la potencia del bus.

En la capa intermedia (USB system) consta de tres componentes principales (Figura §2.5)

- *Host Controller Driver*
  Permite asignar diferentes implementaciones de controladores de anfitrión en el sistema USB de forma más simple. Así el cliente puede interactuar de forma transparente con su dispositivo, independientemente de a qué controlador esté conectado.

  También existe una interfaz entre el *Universal Serial Bus Driver* (*USBD*) y *Host Controller Driver* (*HCD*) llamada *Host Controller Driver Interface* (*HCDI*) que no está definida en la especificación de USB. Por esto, no está disponible para los clientes y cada sistema operativo puede soportar varias implementaciones de ésta.

- *Universal Serial Bus Driver*
  El *USBD* proporciona a los clientes la interfaz básica de anfitrión. Uno de los principales objetivos del *USBD* es facilitar el intercambio de datos en forma de paquetes de solicitud (IRP), que se basa en mandar solicitudes de transferencia de datos a través de una tubería especifica. Además, es el encargado de presentar a los clientes una abstracción del dispositivo USB. El *USBD* tiene una tubería por defecto que representa una comunicación lógica entre el *USBD* y la abstracción del *USBD* del dispositivo.

- *Host Software*
  Contiene el software que gestiona el funcionamiento de los otros componentes de la capa. La especificación USB no impone una estructura concreta a este componente y como consecuencia, cada desarrollador la implementa de manera diferente.

Por último, la capa de cliente es la que describe las entidades software que interactúan con los dispositivos. Esto quiere decir que estos clientes podrían interactuar directamente con el hardware del periférico. Sin embargo, las características del USB sitúan la capa intermedia (USB system) entre el cliente y el dispositivo impidiendo que éste acceda directamente al hardware.

En resumen, las capas del host permiten, detectar la conexión y desconexión de dispositivos USB, gestionar el control de flujo y el flujo de datos entre el anfitrión y los dispositivos, recolectar estadísticas de actividad, y controlar la interfaz eléctrica entre Host controller y los dispositivos incluyendo el aprovisionamiento de energía.

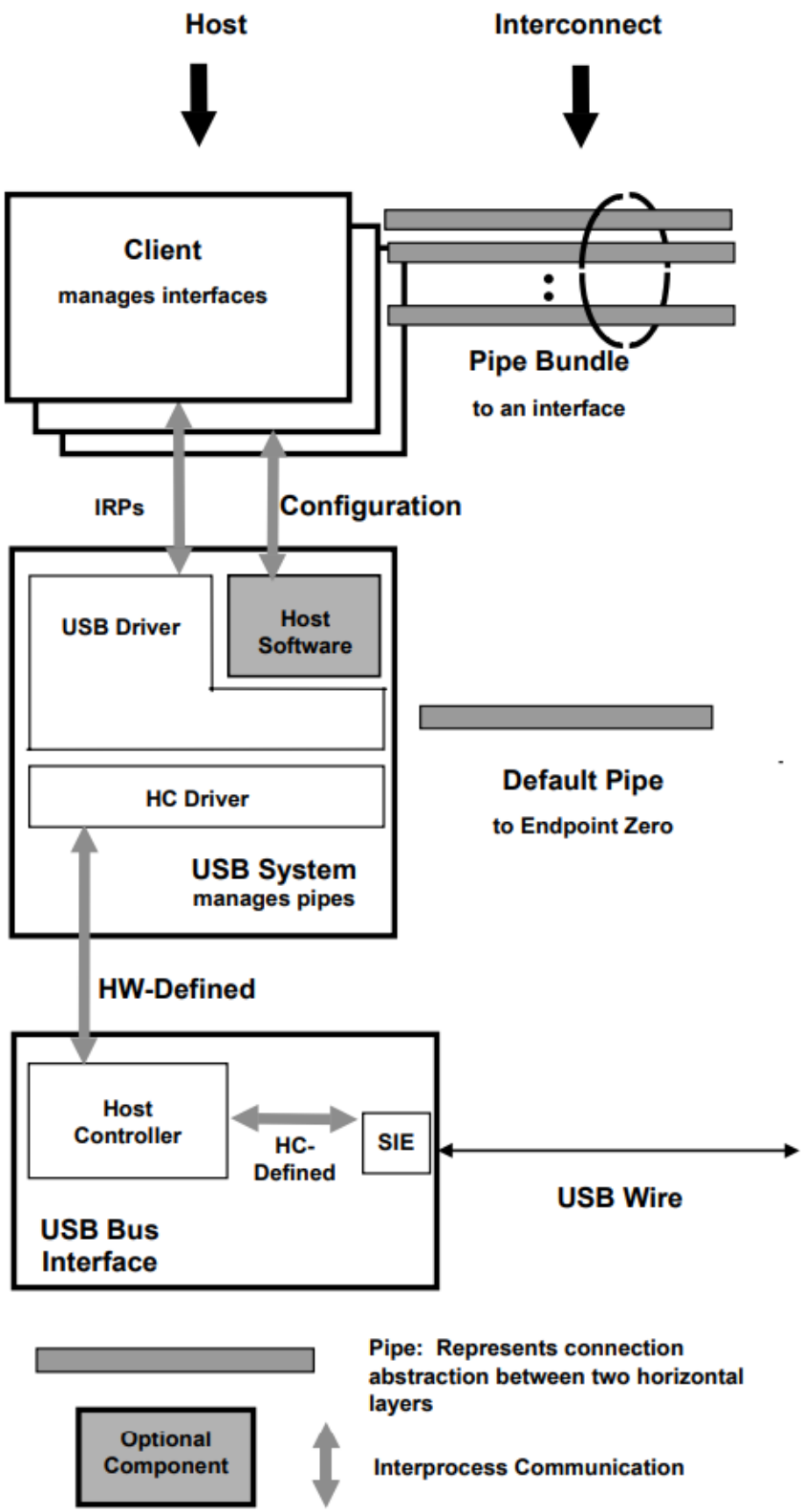

Figura 2.5: Modelo de intercomunicacion [7]

Diseño

3.1 Introduccion

Al comienzo de este trabajo fue necesario realizar una investigación relativa a los diferentes ataques que podía sufrir un ordenador a través de un dispositivo USB. Estos ataques se pueden agrupar en cuatro categorías [5]:

Ataques basados en Microcontroladores Reprogramable

Estos tipos de ataques se basan en microcontroladores, que parecen un dispositivo USB, pero que ha sido reprogramado para realizar operaciones típicas de otros dispositivos, como un teclado con el cual poder inyectar caracteres..

- Rubber Ducky
  Se basa en que una vez conectado el dispositivo al anfitrión, el Rubber Ducky es percibido como un teclado e inyecta una secuencia de caracteres precargados. Estos caracteres pueden llegar a ser muy dañinos, permitiendo al atacante descargarse archivos, instalar programas, etc.

- PHUKD/URFUKED
  Muy parecido al Rubber Ducky, pero introduciendo el factor del tiempo. Permite que el atacante decida cuando inyectar la secuencia maliciosa de caracteres. .

- Evilduino
  Es similar a PHUKD/URFUKED, pero utiliza microcontroladores Arduino. Además de emular un teclado, también simula un ratón mandando caracteres y movimientos del cursor, de acuerdo con el código programado.

- USBdriveby
  Este ataque funciona como el Evilduino, simulando un teclado y ratón permite una rápida instalación de una puerta trasera (backdoor), permite el acceso remoto de un atacante al equipo, y reemplazo de configuración de DNS (DNS spoofer), lo que permite al atacante controlar redirigir el tráfico de la víctima.

Para la puerta trasera no se requiere la apertura de ningún puerto, ya que crea una shell inversa conectándose a un servidor controlado por el atacante, cada cinco minutos usando una tarea programada. En el caso de perder la conexión se reconectará después de cinco minutos.

- Unintended USB channel
  Se basa en el *Proof of Concept* (*POC*) USB hardware trojan [6]. Que exfiltra datos aprovechando canales USB no intencionados, como el uso de altavoces USB, para crear comunicaciones bidireccionales violando así la integridad y confidencialidad de los datos.

- TURNIPSCHOOL (COTTONMOUTH-1)
  Es un implante hardware oculto en un cable USB. Este proporciona la capacidad de comunicación por radiofrecuencia de corto alcance a la víctima, usa una antena bluetooth que permite recibir y posteriormente inyectar comandos a la víctima.

- RIT attack / TOCTTOU attack mediante una memoria USB
  RIT attack es un ataque que se basa en el cambio de contenido de los archivos del pendrive mientras el dispositivo está conectado.

  Toda memoria USB tiene unos ficheros que permiten realizar actualizaciones de firmware e instalación de aplicaciones en el anfitrión. Éstos ficheros están protegidos contra alteraciones por unas firmas criptográficas. El anfitrión revisa la validez de estas firmas y posteriormente realiza las operaciones de instalación, esperando que estos ficheros no cambien. La vulnerabilidad se sitúa entre la comprobación de las firmas y la instalación, donde la memoria USB aprovecha y modifica los archivos, permitiéndola pasar la inspección y provocando que el anfitrión instale el código malicioso. [21]

- KeySweeper
  Consiste en un cargador USB que registra y descifra de forma encubierta las pulsaciones de muchos teclados inalámbricos RF de Microsoft.

- Default Gateway Override
  Utiliza un microcontrolador para falsificar un adaptador Ethernet USB para interceptar la configuración DHCP y secuestrar el tráfico del anfitrión.

Ataques basados en reprogramación maliciosa del firmware del periférico

Estos ataques a diferencia de los anteriormente mencionados se basan en la modificación del firmware de un dispositivo usb para realizar alguna operación maliciosa.

- Smartphone-based HID attacks
  Consiste en un controlador malicioso que interactúa con la API de los gadgets USB de Android para simular dispositivos de teclado y ratón USB. Fue descrito por primera vez en un artículo de investigación [41]

- DNS Override by Modified USB Firmware
  Este ataque consiste en la modificación del firmware del USB para emular un adaptador Ethernet lo que permite al atacante secuestrar el tráfico local de red.

- Keyboard Emulation by Modified USB Firmware
  Es una modificación del firmware que permite al atacante emular un teclado e inyectar comandos.

- Hidden Partition Patch
  Reprograma el USB para que parezca legítimo, pero crea una partición oculta que no puede ser formateada, permitiendo la exfiltración de datos.

- Virtual Machine Break-Out
  Utiliza el firmware del USB para escapar de entornos virtualizados.

- Boot Sector Virus
  El atacante utiliza los drivers del USB para infectar el ordenador antes de arrancar.

- iSeeYou
  Es un proyecto *Proof of Concept* (*POC*) que reprograma el firmware de una cámara web Isight, internamente montada en productos Apple, que el atacante pueda capturar vídeo de forma encubierta sin el aviso del indicador led. [4]

Ataques basados en dispositivos USB no programados

Por último, estos ataques no reprograman el dispositivo ni modifican el firmware, sino que se aprovechan de los fallos en la forma en que los sistemas interactúan con los protocolos y estándares USB.

- CVE-2010-2568.LNK exploit
  Este es la vulnerabilidad usada por Stuxnet y Fanny malware. Consisten en que la shell de Windows, permitía a los usuarios locales o a los atacantes remotos ejecutar código arbitrario a través de un archivo de acceso directo .LNK o .PIF manipulado, que no se maneja correctamente durante la visualización de iconos en el explorador. [8]

- USB Backdoor into Air-Gapped Hosts
  Fue desarrollado por un grupo de la *Agencia de Seguridad Nacional* (*NSA*) y es usado por Fanny malware. Utiliza un almacenamiento USB para guardar comandos que mapean los ordenadores en las redes, se recopila esta información en el USB.

- Data Hiding on USB Mass Storage Devices
  Consiste en un conjunto de trucos para camuflar el malware o los datos robados dentro del USB. Por ejemplo, utilizando particiones o carpetas ocultas.

- AutoRun Exploits
  Dependiendo de la configuración del anfitrión, el dispositivo puede auto ejecutar programas predeterminados localizados en el USB.

- Cold Boot Attacks or RAM dump attack
  El atacante puede guardar un volcado de memoria (que consiste en un registro no estructurado del contenido de la memoria) en una unidad flash USB y extraer los datos sobrantes de la RAM.

- Buffer Overflow based Attacks
  Consiste en sobrecargar el buffer del sistema operativo cuando un USB se conecta.

- Driver Update
  Este ataque es bastante complejo ya que requiere de un certificado VeriSign Class 3, y mandar drivers a Microsoft que son automáticamente instalados en el anfitrión cuando es insertado el USB.

- *Device Firmware Update* (*DFU*)
  Se Aprovecha del *DFU* que es un proceso legitimo soportado por el estándar para actualizar el firmware, con una versión maliciosa.

- USB Thief
  Es una memoria USB programada para el robo de datos. No deja ninguna señal de su paso por el sistema, y utiliza un mecanismo para protegerse de ser reproducido o copiado. [12]

- Attacks on Smartphones via the USB Port
  Los atacantes esconden y distribuyen malware malicioso a través de puertos de carga USB de móviles.

- USBee attack
  Hace que el bus de datos del conector USB emita señales electromagnéticas para provocar una fuga de datos.

Ataques eléctricos

- USB Killer
  Este tipo de ataque es bastante complicado evitarlo, ya que consiste en enviar un alto voltaje por la conexión USB, llegando a quemar los componentes del anfitrión.

3.2 Requisitos

Una vez ya investigados los diferentes ataques que existen, se plantearán unos requisitos iniciales. Al trabajar con una metodología iterativa incremental no es necesario que los requisitos propuestos sean muy específicos ya que tiene la posibilidad de ir ampliándose de forma incremental, pero hay que realizar los más prioritarios al comienzo para así obtener un sistema funcional lo antes posible.

La lista de requisitos funcionales en orden ascendente de prioridad es la siguiente.

1. Proporcionar una interfaz de usuario con luces.
2. Poder diferenciar entre los diferentes dispositivos conectados.
3. Proteger el sistema de la mayor cantidad de ataques.
4. Detectar la mayor cantidad de ataques.
5. Poder analizar las memorias USB en busca de ficheros infectados.

La lista de requisitos no funcionales es la siguiente.

1. Producir un dispositivo compacto y fácil de utilizar.
2. Minimizar el gasto energético.
3. Minimizar el tiempo de arranque a 30 segundos.
4. Automatizar el proceso de instalación.

El requisito no funcional 1 se satisface con la elección de las tecnologías y herramientas.

Implementación

4.1 Introduccion

La implementación de los requisitos requiere un conocimiento detallado del sistema. Como se menciona en la Sección §2, el sistema operativo utilizado en la Raspberry Pi, es un Raspbian, que está articulado al rededor de un kernel Linux. Éste ha sido usado en diferentes asignaturas del grado, lo que ha facilitado mucho el trabajo de implementar los diferentes requisitos, que se explica en las siguientes secciones.

4.2 Diferenciar entre los diferentes dispositivos conectados

La importancia de este requisito se debe a que los diferentes dispositivos son reconocidos por el anfitrión de forma diferente. Por ejemplo, algunos son reconocidos como interfaz humana (human interface) y otros como unidades de almacenamiento (mass storage), entre otras posibles opciones. Para poder ver en Linux los dispositivos conectados por USB existe el comando *lsusb* [32]. El resultado de ejecutarlo sin opciones y sin periféricos conectados aparece en el listado 4.1. Como se puede observar enumera tres dispositivos conectados al USB. Estos dispositivos son, el propio Hub USB de la Raspberry Pi y los chips controladores de los puertos USB y Ethernet. Estos dispositivos son considerados legítimos por defecto, y es necesario tenerlos en cuenta para poderlos diferenciar de los periféricos conectados.

Listado 4.1: Comando lsusb sin dispositivos conectados

```
$ lsusb
    Bus 001 Device 003: ID 0424:ec00 Microchip Technology, Inc. SMSC9512/9514 Fast ↩
        Ethernet Adapter
    Bus 001 Device 002: ID 0424:9514 Microchip Technology, Inc. SMC9514 Hub
    Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Con este mismo comando, si utilizamos la opción -v nos muestra información más detallada acerca de los dispositivos conectados. En el listado 4.2 se puede observar alguna de esta información más detallada de un teclado conectado al bus. En esta información se encuentran dos variables que son de mucha ayuda, estas son bInterfaceClass y bInterfaceProtocol. La primera indica de qué clase es el dispositivo, puede ser una unidad de almacenamiento, de audio o en este caso interfaz humana. La segunda es un descriptor de la interfaz del dispositivo, en este caso un teclado.

Listado 4.2: Comando lsusb -v con un teclado conectado

```
$ lsusb -v
    Bus 001 Device 004: ID 258a:0001 SINO WEALTH USB KEYBOARD
    Couldn't open device, some information will be missing
    Device Descriptor:
      bLength                 18
      bDescriptorType          1
      bcdUSB                1.10
      [...]
      idVendor           0x258a
      idProduct          0x0001
      bcdDevice            1.00
      iManufacturer           1 SINO WEALTH
      iProduct                2 USB KEYBOARD
      iSerial                 0
      bNumConfigurations      1
      Configuration Descriptor:
        bLength                 9
        bDescriptorType         2
      [...]
        MaxPower              100mA
        Interface Descriptor:
          bLength                 9
          bDescriptorType         4
          bInterfaceNumber        0
          bAlternateSetting       0
          bNumEndpoints           1
          bInterfaceClass         3 Human Interface Device
          bInterfaceSubClass      1 Boot Interface Subclass
          bInterfaceProtocol      1 Keyboard
      [...]
```

El comando *lsusb* con sus opciones, permite implementar el USBGuardian que detecta los dispositivos conectados. Cuando arranca, lee los dispositivos conectados, que son los considerados legítimos por defecto y guarda sus identificadores en un fichero llamado *USB_default*. A partir de ese momento, cuando se vuelva a ejecutar, comprobará qué dispositivos hay conectados y descartará los que tienen identificadores listados en ese fichero. Debido a esto, al arrancar la Raspberry no debe haber ningún periférico conectado al USB.

Posteriormente, al detectar que ya existe este fichero trata de localizar otros dispositivos. Esto lo hace utilizando el comando mencionado, pero descartando los dispositivos con los identificadores guardados en el fichero USB_default.

De los dispositivos no descartados, el USBGuardian recoge su identificador, así como las variables importantes mencionadas anteriormente y las almacena en un fichero llamado *USB_unknown*. Por último, revisa esta información en busca de periféricos comunes como ratones, teclados, almacenamiento, audio, hubs y muestra por pantalla la cantidad de estos encontrados. Si algún dispositivo no es identificado como alguno de estos periféricos se clasifica como *otro* y también se imprimirá por pantalla el número de estos.

Por último, es necesario determinar la forma de ejecución de este script. He realizado varios cambios en este aspecto. La primera versión utilizaba un bucle infinito para detectar nuevos dispositivos, lo que provocaba un uso de CPU considerablemente alto. Para mejorar esto, la segunda versión eliminaba el bucle infinito y se delegaba a un servicio de systemd la ejecución periódica con un intervalo de varios segundos. Aunque esto reducía el uso de CPU, también hacía la detección menos rápida, ya que había que esperar a la próxima invocación del script. La versión final aprovecha el servicio Udev (Sección §2.2.1). En éste se configuraron dos reglas (Listado 4.3) que ejecutan el script solamente cuando un periférico se conecta o desconecta. Así hacia el uso mínimo necesario de CPU para esta ejecución manteniendo la velocidad de respuesta.

Listado 4.3: Reglas Udev, que detectan la conexion y desconexion de perifericos

```
#/etc/udev/rules.d/80-execute-script.rules
SUBSYSTEM=="usb", ACTION=="add", ENV{DEVTYPE}=="usb_device", RUN+="/usr/local/sbin/↩
    usbguardian/usb-detector.sh"
SUBSYSTEM=="usb", ACTION=="remove", ENV{DEVTYPE}=="usb_device", RUN+="/usr/local/sbin/↩
    usbguardian/usb-detector.sh"
```

4.3 Proteger el sistema y detectar la mayor cantidad de ataques

Proteger y detectar de los ataques a la Raspberry es el requisito principal del proyecto, por ello me ha requerido más tiempo de investigación y de implementación. Este requisito es muy amplio, se han enumerado 29 tipos de ataques diferentes. Por ello se ha optado por identificar un vector común al mayor número de ataques, y así evitar varios con una sola implementación. Se detectó que un conjunto significativo de ataques se basaban en una misma premisa, simular un teclado USB para emitir al anfitrión comandos que lo pongan en peligro. Los ataques identificados son los siguentes:

- Rubber Ducky
- PHUKD/URFUKED
- Evilduino
- USBdriveby
- Keyboard Emulation by Modified USB Firmware

Antes de proceder a la detección de un dispositivo malicioso de este tipo, es importante proteger nuestro sistema. De otro modo, un dispositivo podría anular el proceso de detección y pasar desapercibido. Para ello se ha tomado la decisión de ignorar las pulsaciones de los teclados. Además, la versión del sistema operativo es Raspbian Lite, que no dispone de interfaz gráfica, lo que nos protege de dispositivos que simulen un ratón.

Para bloquear las pulsaciones de los teclados, he vuelto a utilizar el servicio de Udev, por lo que he creado una regla que bloquea las pulsaciones de estos 4.4. Esto se probó conectando un teclado y viendo que el sistema era insensible a los eventos del mismo.

Listado 4.4: Reglas Udev, que detectan la conexion y desconexion de perifericos

```
#/etc/udev/rules.d/81-disable-usb-devices.rules
SUBSYSTEMS=="usb", DRIVERS=="usbhid", ACTION=="add", ATTR{authorized}="0"
```

La parte de detección recae sobre el USBGuardian. Pero este debe distinguir entre un teclado legítimo y uno simulado, y para probar si esto funcionaba correctamente era necesario conseguir algún dispositivo malicioso. En la sección §5 se detalla el proceso seguido para la obtención e instalación de un Rubber Ducky.

Al probar el Rubber Ducky se podía observar en la pantalla que este dispositivo conseguía escribir los caracteres que habían sido programados, a pesar de la protección contra los teclados. En principio, el dispositivo no podía vulnerar el sistema ya que se encontraba en la pantalla de inicio de sesión, y para ello el Rubber Ducky tendría que tener almacenados usuario y clave. Para proteger el sistema de esta posibilidad opté por cambiar la contraseña por defecto, por una más segura. Pero en cualquier caso es necesario extremar las precauciones ya que es posible que incluso si se usa una contraseña segura, el sistema puede seguir siendo vulnerable a un ataque por fuerza bruta o un ataque de diccionario.

La solución escogida fue deshabilitar el servicio encargado del inicio de sesión, el *getty@tty*. [11] Esto consigue impedir el acceso al dispositivo por las consolas de texto. Otras cosas que intenté para solucionarlo fue modificar la regla de Udev o añadir alguna otra para bloquear este dispositivo. La información necesaria para construir tal regla se obtiene con el comando *udevadm info*, que muestra información detallada de controladores de dispositivos. Pero esta información identifica al Rubber Ducky de manera demasiado detallada, a través de su vendorID. El vendorID es el identificador del proveedor que ayuda al ordenador a identificar el hardware del dispositivo que se conecta [23]. Pero como este tipo ataques se puede realizar desde otros dispositivos con diferentes identificadores, no había una única forma que permitiese identificar a todos los dispositivos maliciosos. Habría que hacer una lista negra de las empresas proveedoras de hardware utilizado para la realización de estos ataque, teniendo la posibilidad de bloquear hardware legitimo. Por otro lado, el dispositivo aparece en la carpeta /dev como ttyACM [36], indicando que esta siendo gestionado por el driver de módems de Linux, que es diferente al usado por los teclados. Por ello intenté bloquear éste mediante una regla de Udev, pero bloqueaba también los dispositivos legítimos y además impedía que el Rubber Ducky pudiera ser analizado.

Mirando en más detalle cómo se percibe el Rubber Ducky desde Linux, se puede ver que es un dispositivo de comunicación USB *CDC* del subtipo *Abstract Control Model* (*ACM*). El *ACM* es una estandarización que generaliza la comunicación con módems, unos aparatos que mediante operaciones de modulación trasforman los bits en señales analógicas, las cuales se pueden transportar por los cables telefónicos para ser posteriormente demoduladas, y reconvertirlas en señales digitales. El driver *ACM* de Linux, hoy en día se utiliza comúnmente para comunicarse con dispositivos programables, como por ejemplo los Arduino. Es necesario buscar una manera de distinguir a los módems de los dispositivos programables, que son susceptibles de ser usados como dispositivos de ataque.

Una vez que el sistema operativo se ha puesto en contacto con un módem y ha activado su driver *ACM*, se establece un canal bidireccional de comunicaciones entre ambos. Este canal está gobernado por el estándar V.250, que define una serie de comandos, conocidos como comandos AT [40]. Éstos son usados para la configuración de los módems y el establecimiento de comunicaciones. Sin embargo los dispositivos programables no utilizan este protocolo, por lo que no responderán a este tipo de comandos. Gracias a esto podremos realizar pruebas al dispositivo, ya que si al mandarle un comando AT, este no responde correctamente, podemos determinar que es un dispositivo programable potencialmente malicioso.

A veces este tipo de microcontroladores no vienen con una interfaz hardware USB, aunque es posible utilizar un stack software-only USB (V-USB). El cual permite crear hardware USB de forma lógica en los microcontroladores AVR, que son unos microcontroladores que se basan en una arquitectura RISC fabricados por ATMEL microchip [19], sin la necesidad de un chip adicional. Las limitaciones adicionales impuestas a la CPU y el tamaño de almacenamiento suelen llevar a los diseñadores de placas a incluir un puente *UART* dedicado al USB. Como Linux agrupa los dispositivos similares bajo un mismo nombre de dispositivo, los *Universal Aynchronous Receiver Transmiter* (*UART*)s, son unos dispositivos hardware de comunicación, en el cual el formato de los datos y la velocidad es configurable, y manda bits uno a uno. Si están presentes en un ordenador se llamarán ttyS* y del mismo modo los dispositivos con funcionalidades UARTS sobre USB se llamarán ttyUSB*.

En definitiva, para poner a prueba estos dispositivos necesitamos probar los comandos AT por los dispositivos ttyACM, ttyUSB y no por los dispositivos ttyS, ya que el hardware del dispositivo conectado puede no entender estos comandos, aun siendo este legitimo. El comando más simple es el *AT* el cual comprueba si el dispositivo está disponible, y devuelve 'OK' si es así.

Para la implementación de este comando he utilizado una herramienta llamada Atinout [2] que lee una lista de comandos AT, los manda al módem y devuelve la respuesta de este. Con esta herramienta he incorporado al USBGuardian la funcionalidad de detectar un dispositivo por alguna de las interfaces mencionadas, y le ponga a prueba mediante comandos AT, mientras le proporciona feedback con la interfaz de luces §4.5.

Al finalizar con la protección y detección de estos ataques que simulan un teclado, he avanzado ampliándolo a otros tipos de ataques. Estos eran los que simulaban un adaptador Ethernet: Default Gateway Override y DNS Override by modified USB firmware. He protegido el sistema bloqueando la interfaz Ethernet mediante una regla Udev, pero como no disponía de ningún dispositivo malicioso de este tipo, no he podido probar su funcionamiento. Además, durante el proceso de desarrollo es necesario acceder al sistema por una conexión ssh, por lo que esta regla permanece deshabilitada.

4.4 Minimizar el tiempo de arranque

Minimizar el tiempo de arranque es un requisito relativamente importante, de lo contrario hace que el dispositivo sea incómodo de usar. A medida que avanzaba el proyecto, el tiempo de arranque podía estar entre uno y tres minutos en arrancar.

Reducir el tiempo de arranque requiere saber identificar qué servicios y módulos serán utilizados y cuales no en la Raspberry Pi. En mi caso, al solo necesitar utilizar el bus USB, el GPIO y el servicio Udev, puedo deshabilitar la mayor cantidad de servicios. Sin embargo, existen muchas dependencias entre servicios, por lo que es necesario utilizar herramientas que permitan gestionar esta complejidad. El kernel de Linux proporciona una serie de comandos para inspeccionar, cargar y descargar módulos. *Lsmod* muestra la lista de modulos cargados e *insmod* carga un módulo. Del mismo modo, systemd tiene el comando *systemctl* [10] para la activación y arranque de servicios, entre otras cosas, y el comando *systemd-analyze* para recabar información sobre el proceso de arranque, como por ejemplo los tiempos de arranque y espera o el camino crítico de los servicios.

Se parte de la base de que la Raspberry Pi está recién instalada, con los parámetros por defecto. Y que se va a determinar una lista de servicios y módulos innecesarios que han de ser deshabilitados. Para que este proceso sea más automático, he creado un script que se encarga de realizar las operaciones necesarias, para disminuir en gran medida el tiempo de arranque.

Este script realiza lo siguiente: deshabilita servicios innecesarios, deshabilita módulos innecesarios y retoca variables en los ficheros de arranque para optimizarlos. Los servicios que han sido deshabilitados son los siguientes [16]:

```
Man-db                          Systemd-remount-fs
Sendmail                        Systemd-fsck-root
Systemd-timesyncd               Apt-daily
systemd-tmpfiles-setup          Wpa-supplicant
systemd-update-utmp             Avahi-daemon
Systemd-tmpfile-setup-dev       Dphys-swapfile
Systemd-sysusers                Dhcpcd
```

El kernel modular de Linux en el arranque carga diferentes módulos por defecto, que no son utilizados en el proyecto, por ello los he deshabilitado añadiéndolos al fichero /etc/modprobe.d/blacklist.conf. Estos módulos son los siguientes:

```
Ip_tables                Bcm2835_isp
Fixed                    Raspberrypi_hwmon
Uio_pdrv_genirq          Vc_sm_cma
I2c_bcm2835              Vc4
Bcm2835_codec            8021q
Snd_bcm2835              Drm
Bcm2835_v4l2
```

Por último, se añaden una serie de parámetros a los ficheros de configuración del arranque propios de una Raspberry Pi. Estos ficheros son: /boot/config.txt, donde se deshabilitan las conexiones bluetooth y el retraso en el arranque entre otras cosas, y /boot/cmdline.txt donde es necesario eliminar la palabra slash y añadir quiet, para que en el inicio del sistema no se muestre el proceso de arranque por pantalla. Todos estos cambios permiten minimizar el tiempo de arranque a menos de 30 segundos.

4.5 Interfaz con luces

Este requisito es imprescindible, ya que es la forma en la que comunicaré al usuario el estado del periférico. Como se explica en la sección §2 dispongo de 6 LED: 2 verdes, 2 amarillos y 2 rojos en el Berry-Clip y como se ha de conectar.

Para gestionar este dispositivo [17] se ha optado por encender y apagar los LED mediante Python [24]. Para ello he implementado un controlador que según los parámetros de línea de comandos permite encender o apagar los LED. El controlador es bastante sencillo, requiere 2 parámetros de entrada, con el primero se especifica el color de los LED a encender o apagar, que puede ser: green, yellow, red o all (para referirse a todos los LED). El segundo parámetro es 0 para apagar los LED y 1 para encenderlos.

Acerca del significado de las luces y el funcionamiento de la interfaz. He de mencionar que al arrancar la Raspberry (con el Berry-Clip) todas las luces se encenderán, y después se apagarán, esto significa que una vez apagadas las luces el usuario ya puede conectar el periférico que quiere comprobar. Posteriormente, pueden encenderse los LED verdes o los LED amarillos. Si son los LED verdes es que el dispositivo ha pasado los filtros con éxito, si son los LED amarillos significa que se está comprobando el dispositivo. Al finalizar la comprobación los LED amarillos se apagarán y se encenderán los verdes si ha superado los filtros con éxito o los rojos en caso contrario.

Para el control de los LED en el arranque he creado dos servicios, enciende-leds.service y apaga-leds.service. El primero se encarga de encender los LED lo antes posible en el proceso de arranque. Este momento es cuando ya podamos acceder al sistema de ficheros, ya que es necesario acceder al controlador y ejecutar python, lo que se sitúa entre unos 5 y 10 segundos antes de acabar el proceso de arranque. Exactamente después del servicio e2scrub_reap que comprueba los metadatos de los sistemas de ficheros ext* [42]. Por otro lado, el servicio encargado de apagar los LED se ejecuta después de finalizar el systemd-user-sessions.service, que es el encargado de permitir al usuario iniciar sesión después del arranque. [9]

4.6 Facil instalacion

Este requisito no es el mas importante, pero si necesario. Esto es porque si un usuario desea crear su propio USBGuardian, el proceso de instalacion deberia realizarse de forma sencilla.

Para conseguir esto, al estar en un entorno Debian, se puede crear un paquete Debian [13]. Este paquete es un archivo comprimido, con una estructura predefinida, que permite que sea manipulado por el gestor de paquetes. Con el objetivo de lograr su compilacion, intalacion, actualizacion o eliminacion, de forma segura, estable, comoda y centralizada.

La estructura interna de este archivo consta de una carpeta DEBIAN, la cual contiene un fichero de control, que dispone de la configuración del paquete (dependencias, versión, autor, etc). Luego puede llegar a tener varios ficheros que se ejecutarán en diferentes momentos de la instalación o eliminación. Yo he utilizado el postinst, para realizar la ejecución de algún script necesario, así como la ejecución de comandos importantes después del proceso de instalación. Y postrm para eliminar algún fichero de datos, creados por el USBGuardian.

Pruebas

5.1 Rubber Ducky

Como he comentado en la parte de implementación, para realizar las pruebas es necesario disponer de dispositivos maliciosos. Así se puede comprobar que los filtros de seguridad son correctos, es decir, que detectan correctamente estos dispositivos y no los legítimos. Como se puede imaginar, conseguir algún dispositivo de esta clase no es tarea fácil, ya que no es un producto con un gran mercado. Debido a esto, busqué la forma de implementarlo yo mismo, y encontré el siguiente tutorial [14], que me permitió crear mi propio Rubber Ducky de forma sencilla.

En este tutorial se utiliza una placa CJMCU BadUsb con un chip ATMEGA32u4 que simula un Arduino Leonardo §5.1, esto permite programarla mediante al IDE de Arduino. Ademas, se proporciona el codigo arduino y algunos scripts de prueba, pero es compatible con cualquier script de Rubber Ducky que se pueden encontrar por repositorios de github [18].

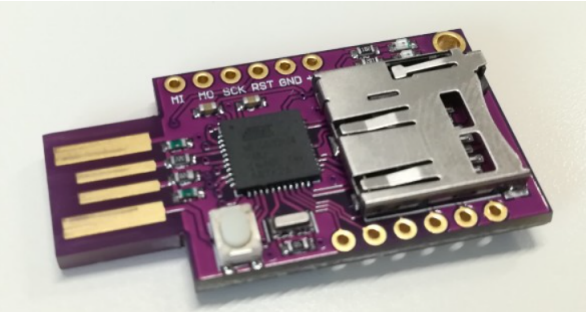

Figura 5.1: Placa CJMCU con chip ATMEGA32u4

A la hora de programar este Rubber Ducky, hay que proporcionarle una configuracion básica, que consta de lo siguiente.

- Modo
  El modo de operación puede tomar tres opciones diferentes. “m” modo de gestión, el cual al conectar el Rubber Ducky no ejecuta la carga de trabajo. “a” modo de auto desarme, que al conectarlo por primera vez ejecuta la carga de trabajo y se desarma, es decir, la siguiente vez que se conecte estará en modo de gestión. Y por ultimo el modo de ejecución continua “c”, que implica que cada vez que el Rubber Ducky sea conectado ejecutara la carga de trabajo.

- Lenguaje
  Este parametro de configuracion, especifica el lenguaje del teclado en mi caso es “ES” español.

- Carga de trabajo
  La carga de trabajo es el nombre del fichero que ha de incorporarse previamente a la tarjeta micro SD, y contiene los comandos que el Rubber Ducky ejecutará

5.2 Dispositivos probados

Dispositivos que pasan las pruebas de seguridad con éxito

Los siguientes son los dispositivos con los que se puso el sistema a prueba, y pasaron exitosamente los filtros de seguridad.

- Teclado CM storm Cooler Master devastator
  Es un teclado USB normal y corriente que se identificó como hidraw0, hidraw1 y USB.

- UtechSmart Venus MMO gaming mouse
  Este ratón gaming es reconocido por el USBGuardian como un ratón, pero el sistema lo reconoce como hidraw0, hidraw1, hidraw2 y USB. Los tres dispositivos hidraw se debe a que este ratón tiene un teclado numérico en un lateral.

- Pendrive Toshiba 2.0
  El USBGuardian reconoce como dispositivo de almacenamiento a este pendrive, y el sistema como sda1.

- Logitech g430
  Unos auriculares con micrófono, que son reconocidos como audio por el USBGuardian y como hidraw0 por el sistema.

- Wholesale 4 port USB 2.0 hub
  Un hub con cuatro puertos USB 2.0, que es reconocido como tal por el USBGuardian. Pero el sistema no lo identifica como un dispositivo en /dev al ser este una extensión de puerto USB.

- Webcam Logitech c270
  Esta webcam es reconocida por el sistema como 4 dispositivos, media0, v41, video0 y video1. El USBGuardian lo detecta dentro del conjunto de "otros"

- Samsung Portable DVD Writer Model SE-208
  Es un Lector de DVD que al conectarlo el USBGuardian lo identifico como "otros". El propio kernel de Linux lo identifico como un dispositivo BSG.

- Subblim combo Wireless ergo
  Este dispositivo consta de dos componentes un teclado y un ratón inalámbricos, que al conectarlos fueron detectados por el USBGuardian como teclado y por el sistema como hidraw0, hidraw1 y USB.

- High-Power Wireless USB Adapter
  Para probar con algún dispositivo diferente probé con este adaptador wifi, que al ser un dispositivo de red no se reconoce como un dispositivo en /dev, sino que salía añadido en /sys/bus/. Sin embargo, el USBGuardian lo detecto como "otros".

- Mini lector EVO lector de smartcard
  El Mini lector sirve para leer los DNI electrónicos los cuales son muy utilizados actualmente. El USBGuardian lo identifica como "otros", pero el sistema no lo incorpora como un dispositivo en /dev, seguramente al ser un dispositivo más específico se incorporará en alguna otra parte del sistema de ficheros, como /sys/bus.

Dispositivos que NO superan las prueba de seguridad

Aquí se recogen los dispositivos que no han superado los filtros de seguridad

- Rubber Ducky
  Como ya he explicado anteriormente, el funcionamiento de este dispositivo se basa en la simulación de un teclado que inyecta caracteres con el fin de vulnerar el sistema. El sistema lo reconoce como ttyACM y el USBGuardian como "otros" al realizar las pruebas, este no las supera.

Conclusiones

6.1 Objetivos conseguidos

Como resumen, de los objetivos propuestos al comienzo de esta memoria, varios de ellos han sido logrados. Se ha consiguiendo un dispositivo autónomo, que sirve como herramienta de seguridad para los periféricos USB.

Este dispositivo, basado en un computador Raspberry Pi, permite detectar periféricos USB maliciosos, y mostrar al usuario el estado de estos mediante una interfaz de luces. Además se han implementado medidas para mejorar su robustez frente a ataques por parte de los dispositivos que ha de detectar. Para facilitar tanto la implementación como el uso, el código desarrollado se proporciona en un paquete Debian para la instalación. [13]

En la fase de pruebas se ha probado con multitud de periféricos legítimos sin que el dispositivo diera falsos positivos. Además se ha implementado un periférico malicioso que el aparato detecta correctamente.

6.2 Trabajos futuros

Este proyecto se podría ampliar permitiendo detectar ataques de distinta naturaleza. Por ejemplo, se podría utilizar alguna herramienta como ClamAV, que es un antivirus capaz de analizar los ficheros dentro de una memoria USB conectada al sistema, para detectar ficheros infectados con algún tipo de malware.

6.3 Reflexión personal

Como reflexión final del trabajo de fin de grado, el proyecto que he realizado ha sido un logro personal. No solo por el hecho de poder haber aprovechado para aprender acerca de tecnologías nuevas. Sino que además, por haber podido poner en práctica todo lo aprendido durante estos cuatro años de carrera, y poder hacer realidad una idea con la que había imaginado desde hacía años.

Porque el hecho de haber podido llegar ha realizar este proyecto de una idea propia, me hace darme cuenta de que aunque se me ocurran ideas útiles pero algo disparatadas o impensables, solo requieren de estudio, determinación y pasión para que se hagan realidad.

Bibliografía

[1] About Raspbian. About raspbian. URL https://www.raspbian.org/RaspbianAbout.

[2] Atinout. Atinout, Marzo 2014. URL http://atinout.sourceforge.net.

[3] B. Breard. Mastering systemd: Securing and sandboxing applications and services. URL https://www.redhat.com/sysadmin/mastering-systemd.

[4] M. Brocker and S. Checkoway. iSeeYou: Disabling the MacBook webcam indicator LED. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 337–352, San Diego, CA, Aug. 2014. USENIX Association. ISBN 978-1-931971-15-7. URL https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/brocker.

[5] C. Cimpanu. Here's a list of 29 different types of usb attacks. *Bleeping Computer*, March 13, 2018. URL https://www.bleepingcomputer.com/news/security/heres-a-list-of-29-different-types-of-usb-attacks/.

[6] J. Clark, S. Leblanc, and S. Knight. Hardware trojan horse device based on unintended usb channels. In *2009 Third International Conference on Network and System Security*, pages 1–8, 2009. doi: 10.1109/NSS.2009.48.

[7] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, and Philips. *Universal Serial Bus Specification*, April 27,2000. URL https://www.usb.org/document-library/usb-20-specification.

[8] CVE-2010-2568. Cve-2010-2568, July 2010. URL https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2568.

[9] Debian Man pages. *Debian Man pages user-sessions*, Junio 2022. URL https://manpages.debian.org/experimental/systemd/systemd-user-sessions.service.8.en.html.

[10] Debian wiki. *Debian wiki systemd - system and service manager*, Marzo 2022. URL https://wiki.debian.org/systemd.

[11] W. Fink and K. Zak. *Debian Man pages AGETTY(8)*, February 2016. URL https://manpages.debian.org/bullseye/util-linux/getty.8.en.html.

[12] T. Gardoň. New self-protecting usb trojan able to avoid detection, Marzo 2016. URL https://www.welivesecurity.com/2016/03/23/new-self-protecting-usb-trojan-able-to-avoid-detection/.

[13] P. I. González. Usb guardian, Junio 2022. URL https://gitlab.com/Pablo-Izquierdo/usbguardian.

[14] M. Harjac. Bad ducky: un rubber ducky"por sólo 6,50€, marzo 24, 2018. URL https://www.hackplayers.com/2018/03/bad-ducky-un-rubber-ducky-por-solo-650.html.

[15] R. Hat. Red hat, el concepto de linux. URL https://www.redhat.com/es/topics/linux.

[16] R. Hertzog and R. Mas. *Manual del administrador de Debian*. URL https://debian-handbook.info/browse/es-ES/stable/unix-services.html. isbn 979-10-91414-20-3.

[17] Jecrespom. Aprendiendo arduino, manejar gpio raspberry pi, Marzo 2020. URL https://aprendiendoarduino.wordpress.com/2020/03/04/manejar-gpio-raspberry-pi/.

[18] D. Kitchen, midnitesnake, and Hak5Jason. Rubber ducky payloads, Mar 1, 2022. URL https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Payloads.

[19] Microchip corporation. Microchip corporation. URL https://www.microchip.com.

[20] S. A. Moore, T. Fox, E. C. Bailey, J. Fuller, and J. Ha. *Red Hat Enterprise Linux 3 Manual de administración del sistema*, 2003. URL https://web.mit.edu/rhel-doc/3/rhel-sag-es-3/index.html.

[21] C. Mulliner and B. Michéle. Read it twice! a mass-storage-based tocttou attack. In *WOOT*, 2012.

[22] E. Nemeth, G. Snyder, T. R. Hein, B. W. D. M. with James Garnett, F. Branca, and A. Mouat. *UNIX and Linux System Administration Hadbook*. Addison-Wesley, 2017.

[23] U. organization. Valid usb vendro id numbers. URL https://www.usb.org/sites/default/files/vendor_ids032322.pdf_1.pdf.

[24] Python. Python. URL https://www.python.org.

[25] Raspberry Pi. Raspberry pi. URL https://www.raspberrypi.org.

[26] Raspberry Pi 1B+. Raspberry pi 1b+. URL https://www.raspberrypi.com/products/raspberry-pi-1-model-b-plus/.

[27] Raspberry Pi 1B+ expecification. Raspberry pi 1b+ expecification. URL https://cdn-shop.adafruit.com/datasheets/pi-specs.pdf.

[28] Raspberry Pi Documentacion. Raspberry pi documentacion. URL https://www.raspberrypi.com/documentation/computers/processors.html.

[29] Raspberry Pi documentation. Raspberry pi documentation. URL https://www.raspberrypi.com/documentation/computers/processors.html.

[30] Raspberry Pi OS. Raspberry pi os. URL https://www.raspberrypi.com/software/operating-systems/.

[31] W. W. ROYCE. Managing the development of large software systems: concepts and techniques. *En Proceedings of the 9th international conference on Software Engineering.*, pages 328–338, 1987. ISSN 2123-1234.

[32] T. Sailer. *Linux man page, lsusb(8)*. URL https://linux.die.net/man/8/lsusb.

[33] D. M. H. W. Sarah L. Harris. *Digital Design and Computer Architecture ARM Edition*. Morgan Kaufmann, 225 Wyman Street, Waltham, MA 02451, USA, 2016. ISBN 978-0-12-800056-4.

[34] l. SOMMERVJLLE. *Ingeniería del software (7.ª ed.)*. Pearson, Madrid, Spain, 2005. ISBN 84-7829·074-5.

[35] System and Service Manager. System and service manager. URL https://systemd.io.

[36] S. Tardieu. What is the difference between /dev/ttyusb and /dev/ttyacm?, Marzo 2013. URL https://rfc1149.net/blog/2013/03/05/what-is-the-difference-between-devttyusbx-and-devttyacmx/.

[37] Udev. Udev, 4 June 2022. URL https://wiki.archlinux.org/title/udev.

[38] udev - Linux dynamic device management. udev - linux dynamic device management, 02 January 2021. URL https://wiki.debian.org/udev.

[39] Udev/es. Udev/es, 31 January 2022. URL https://wiki.gentoo.org/wiki/Udev/es.

[40] I. T. UNION. *SERIES V: DATA COMMUNICATION OVER THE TELEPHONE NETWORK, Serial asynchronous automatic dialling and control*, 2004. URL http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-V.250-200307-I!!PDF-E&type=items.

[41] Z. Wang and A. Stavrou. Exploiting smart-phone usb connectivity for fun and profit. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, page 357–366, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450301336. doi: 10.1145/1920261.1920314. URL https://doi.org/10.1145/1920261.1920314.

[42] D. J. Wong. *Debian Man pages*, December 2021. URL https://manpages.debian.org/testing/e2fsprogs/e2scrub.8.en.html.