Universitat Oberta de Catalunya
uoc.edu

IOT rehosting

William Camilo Cipagauta Hortua
Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones
INTERNET OF THINGS

Carlos Hernández Gañán
Víctor García Font
06/2020

FICHA DEL TRABAJO FINAL

Título del trabajo:	*IOT HoneyPot rehosting*
Nombre del autor:	*William Camilo Cipagauta Hortua*
Nombre del consultor/a:	*Carlos Hernández Gañán*
Nombre del PRA:	*Víctor García Font*
Fecha de entrega (mm/aaaa):	06/2020
Titulación:	*Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones*
Área del Trabajo Final:	*Seguridad internet de las cosas*
Idioma del trabajo:	*Español*
Palabras clave	*IOT, HoneyPot, Emulación, Re-Hosting*

Resumen del Trabajo:

El siguiente Trabajo de Fin de Máster propone el desarrollo de una serie de simulaciones y emulaciones sobre firmwares o secciones de firmwares comúnmente usados en dispositivos IoT, se busca poder lograr la ejecución de algunas de estas funcionalidades sin necesidad de contar con las arquitecturas de hardware para el que fueron diseñados, esto se lograra a través de un proceso llamado re-hosting haciendo uso de múltiples herramientas de código abierto usando una serie de ambientes de virtualización los cuales son adaptados a las necesidades de simulación requeridas, los firmware objetivos serán tomados de fuentes abiertas y usualmente usadas en hardware IoT.

Abstract :

The following Final Master's Project proposes the development of a series of simulations and emulations on firmwares or sections of firmwares commonly used in IoT devices, seeking to achieve the execution of some of these functionalities without the need for hardware architectures to which were designed, this will be achieved through a process called re-hosting using multiple open source tools using a series of virtualization environments which are adapted to the required simulation needs, the target firmware will be taken from sources open and usually used in IoT hardware.

Índice

Lista de figuras

1. Introducción

1.1. Contexto y justificación del Trabajo

Debido a la expansión de la tecnología en la vida cotidiana de las personas y el hacer uso de esta para simplificar muchas acciones que se realizan hoy, como encender una luz de manera programada, o tan solo contar con un sistema de video vigilancia en hogares y empresas, ha hecho que todos estos dispositivos estén conectados a una red de datos ya sea una red privada o una red pública como lo es Internet, al estar conectados a una red pública surgen riesgos de seguridad que no son usualmente evaluados como deberían, lo que ha desencadenado que muchos atacantes externos se aprovechen de falencias para tomar control de estos dispositivos con un fin en específico.

Estos dispositivos de la casa o de las oficinas que suelen conectarse a una red para intercambiar información se les denomina Internet de las cosas, debido a que son inteligentes y pueden tomar decisiones en base a diferentes situaciones las cuales son enviadas a través de internet, pero esto desencadena un problema y es, Quien protege a estos dispositivos de ser vulnerables, quien actualiza el firmware de estos dispositivos luego de que se descubra una vulnerabilidad de día Zero (Kaspersky, 2018)?

En consecuencia, estos problemas nacen la necesidad de implementar un sistema que permita evaluar los ataques a los que están usualmente expuestos estos dispositivos y evaluarlos para poder remediar las vulnerabilidades que están siendo explotadas, de esta manera simulando los firmware de los dispositivos más comunes de internet de las cosas se podrá tener unas acciones a realizar para mitigar vulnerabilidades y con esto riesgos de seguridad para las empresas y hogares conectados a internet.

El sistema que saldrá como resultado de esta investigación será un dispositivo que permitirá evaluar los ataques recibidos desde internet como si fuera un dispositivo real de internet de las cosas (IEEE, 2015) recolectando información con el fin de mitigar ataques más comunes y como consecuencia asegurar los firmware de dispositivos de internet de las cosas de manera proactiva, esto se pretende

realizar haciendo uso de sistemas operativos con licencia GNU como Linux, ya que es comúnmente usando en los dispositivos de internet de las cosas y es económico, así mismo se evaluara la necesidad de contar con un dispositivo de seguridad que nos permita recolectar información respecto a patrones de ataques como es un IDS en modo monitor, esta solución también puede ser desplegada haciendo uso de software de licencia libre como Snort.

1.1. Objetivos del Trabajo

Los principales objetivos con esta investigación son:

- Investigar cómo, a través de sistemas de código abierto se pueda adaptar múltiples firmwares de internet de las cosas.
- Investigar el funcionamiento de la mayoría de los dispositivos de internet de las cosas.
- Simular uno o varios sistemas IOT más comunes a través de múltiples herramientas de simulación.
- Comparar el funcionamiento un sistema de IOT en ambientes simulados con firmware de estos dispositivos.
- Entender el comportamiento de las redes y atacantes de internet hacia los dispositivos IOT más comunes del mercado.

1.2. Enfoque y método seguido

El enfoque de este proyecto es debido a un contexto tan grande como el internet de las cosas consistirá en investigar estrategias y métodos para asegurar los dispositivos de internet de las cosas que se encuentren conectados a redes de internet, a través de elementos de red que simulen ser objetos de internet de las cosas, de esta manera recolectar información relevante y en tiempo real.

Debido a que es un proyecto dinámico por la aparición de nuevos elementos en el internet de las cosas y en los ataques desde las redes de internet que todos los días son distintos se requiere de tres etapas.

- La primera etapa (Entrega2) consistirá en ejecutar una investigación que permita comprender como funcionan los dispositivos IoT, teniendo en cuenta sus debilidades, las

vulnerabilidades más comunes y medidas tecnológicas para evaluar estos ataques.

- La segunda etapa (Entrega3) permitirá aprender, seleccionar y desplegar las herramientas una vez seleccionadas, pasando por implementación, configuración, despliegue, pruebas y afinamiento, con el fin de recolectar información relevante de ataques a dispositivos tipo internet de las cosas, estas herramientas serán elementos de código abierto que permitan desplegar herramientas de simulación de manera sencilla y económica.

- Una tercera etapa (Entrega4) se basa en determinar y concluir la información recolectada y como esta información es valiosa para fortalecer los elementos de internet de las cosas usualmente expuestos a internet y sus firmwares.

Estas etapas y objetivos tienen como medios elementos disponibles en la red de Internet, con el fin de ajustarlos a las necesidades de la investigación, para de esta manera potenciarlos y ahorrar costos.

1.3. Planificación del Trabajo

A continuación, se ilustra el calendario propuesto de trabajo y el diagrama de Gantt con las tareas y su respectivo prerrequisito, estableciendo el tiempo de trabajo y las etapas descritas en el apartado anterior.

	Name	Work	Duration	Start	Finish
1	**Plan de trabajo**	88 hours	9 days?	**2/19/20 8:00 AM**	**3/2/20 5:00 PM**
2	Definicion problema a resolver	24 hours	3 days?	2/19/20 8:00 AM	2/21/20 5:00 PM
3	Definicion de objetivos	8 hours	1 day?	2/22/20 8:00 AM	2/24/20 5:00 PM
4	Definicion de metodologia	16 hours	2 days?	2/24/20 8:00 AM	2/25/20 5:00 PM
5	Listado de tareas	16 hours	2 days?	2/25/20 8:00 AM	2/26/20 5:00 PM
6	Planificación de tareas	16 hours	2 days?	2/27/20 8:00 AM	2/28/20 5:00 PM
7	Inicio estado del arte	8 hours	1 day?	2/29/20 8:00 AM	3/2/20 5:00 PM
8	**Investigacion Funcionamiento IOT**	188 hours	22 days?	**3/2/20 8:00 AM**	**3/31/20 5:00 PM**
9	Seleccionar elementos IOT a investigar	40 hours	5 days?	3/2/20 8:00 AM	3/6/20 5:00 PM
10	Investigar trabajos previos sobre IOT y botnets	104 hours	13 days?	3/5/20 1:00 PM	3/24/20 1:00 PM
11	Test sobre herrmientas IOT investigadas	44 hours	5.5 days?	3/24/20 1:00 PM	3/31/20 5:00 PM
12	**Despliegue de herramientas de analisis de ataques sobre IOT**	160 hours	20 days?	**4/1/20 8:00 AM**	**4/28/20 5:00 PM**
13	Implementacion, configuracion y afinamiento de herramientas de analisis IOT	104 hours	13 days?	4/1/20 8:00 AM	4/17/20 5:00 PM
14	Afinamiento de HoneyPot e IDS para recoleccion de informacion	56 hours	7 days?	4/20/20 8:00 AM	4/28/20 5:00 PM
15	**Concluir informacion recolectada para asegurar IOT**	400 hours	25 days?	**4/29/20 8:00 AM**	**6/2/20 5:00 PM**
16	Redaccion de conclusiones en memoria final	200 hours	25 days?	4/29/20 8:00 AM	6/2/20 5:00 PM
17	Elaboracion de conclusiones tras recoleccion de informacion	104 hours	13 days?	4/29/20 8:00 AM	5/15/20 5:00 PM
18	Elaboracion y preparacion de diapositivas y presentacion oral	96 hours	12 days?	5/18/20 8:00 AM	6/2/20 5:00 PM

Imagen 1 - Listado de tareas TFM

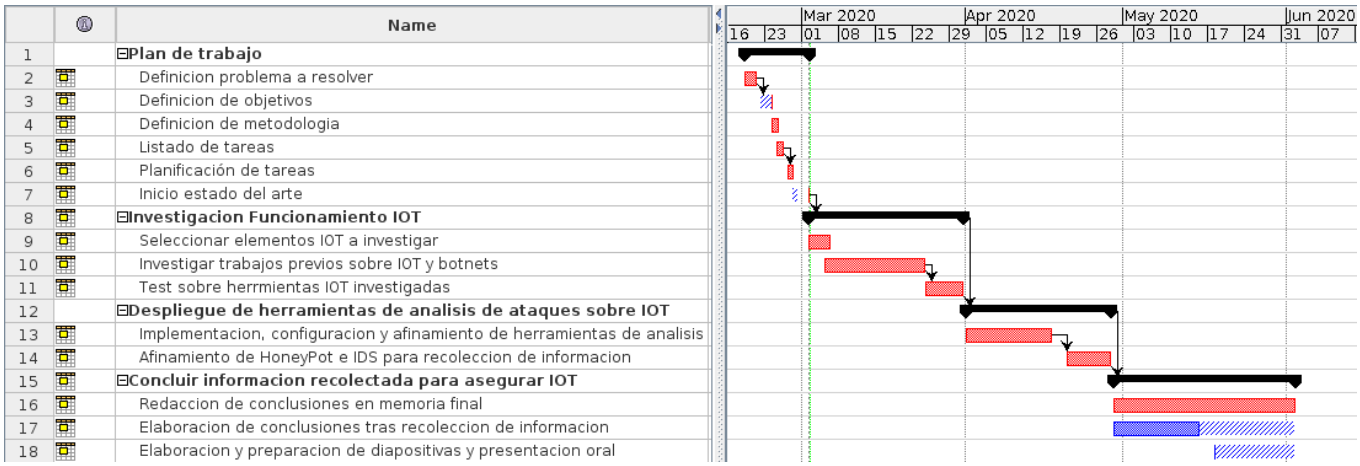


Imagen 2 - Diagrama de Gantt TFM

1.4. Riesgos de la investigación

En referencia a la planificación realizada en el punto 1.3, se debe tener en cuenta riesgos sobre la planeación del proyecto y sobre la probabilidad del fracaso sobre los resultados esperados, a continuación, se resumen riesgos identificados:

1.4.1. R1. Recolección insuficiente de información de ataques

Definición del riesgo:

No contar con la cantidad suficiente de información de ataques a dispositivos IOT con respecto a lo proyectado al inicio del proyecto.

Mitigación del riesgo:

Definir claramente la cantidad de tiempo de recolección, los artefactos de recolección y de información que se requiere para satisfacer las necesidades del proyecto.

1.4.2. R2. Objetivos iniciales de pruebas IOT.

Definición del riesgo:

No contar con las herramientas de software y hardware necesarias para la ejecución del proyecto en el tiempo requerido, por ejemplo, firmware IOT, hardware de instalación.

Mitigación del riesgo:

Definir claramente los artefactos necesarios para la ejecución de la investigación, de manera que se establezcan los costos asociados y el tiempo que se requiere obtenerlos.

1.5. Estado del arte

En el presente escrito se pretende dar una introducción al estado del arte relacionado con los dispositivos que hacen parte de internet de las cosas, así como establecer a través de que tecnologías e investigaciones previas se puede basar para ejecutar la presente investigación.

1.5.1. IOT en el mundo

Las diferentes definiciones y modelos arquitectónicos para IoT reflejan diferentes perspectivas y respaldan diferentes intereses comerciales, esto debido que IoT ha permitido que muchas acciones alrededor del mundo se vean simplificadas en un dispositivo inteligente, por lo que puede tener definiciones ambiguas de acuerdo con sus funciones, en la imagen que se ilustra a continuación se desarrolla el mercado y las partes interesadas en lo que es IoT.

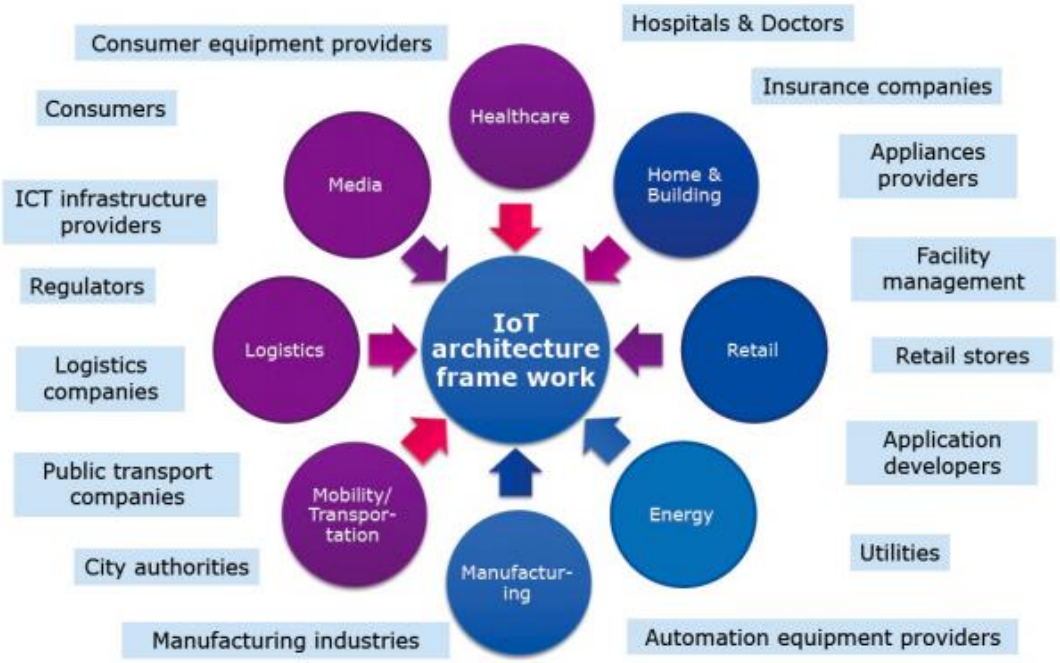


Imagen 3 - Internet de las cosas, mercado e interesados - Imagen tomada de: (IEEE, 2015)

1.5.2. Que es IoT

Así mismo muchas organizaciones mundiales aún no definen el termino IoT como oficial, en el caso de ETSI, lo definen como una comunicación maquina a máquina (M2M) sin intervención humana, “Las comunicaciones MachineOtoOMachine (M2M) son la comunicación entre dos o más entidades que no necesariamente necesitan ninguna intervención humana directa. Los servicios M2M pretenden automatizar los procesos de decisión y comunicación” (ETSI, 2013), así mismo IETF lo define como “La idea básica es que IoT conectará objetos a nuestro alrededor (electrónicos, eléctricos, no eléctricos) (ietf, 2010) para proporcionar una comunicación fluida y servicios contextuales proporcionados por ellos. El desarrollo de etiquetas RFID, sensores, actuadores, teléfonos móviles hace posible materializar IoT que interactúan y cooperan entre sí para hacer que el servicio sea mejor y accesible en cualquier momento, desde cualquier lugar "; de acuerdo a las anteriores definiciones se puede determinar que IoT es un conjunto de elementos electrónicos que interactúan entre sí para lograr una función sin necesidad de intervención humana, estos dispositivos se definirán a continuación con el fin de tener una visión inicial de los requerimientos técnicos para llevar acabo la presente investigación.

1.5.3. Características de comunicación IoT

Así mismo se debe tener en cuenta que los dispositivos IoT tienen una características de comunicación particulares debido a la arquitectura de hardware y software de las que están compuestos, ya que en la gran mayoría de ocasiones no cuentan con una capacidad de procesamiento grande, muchos protocolos a nivel de aplicación se han desarrollado para mejorar la comunicación entre estos dispositivos, por ejemplo protocolos como XMPP(Extensible Messaging and Presence Protocol) el cual es definido por la IEEE en el RFC2778 (IEEE, 2000) y se describe por sus siglas así:

- Extensible: se puede personalizar según las necesidades individuales del usuario.

- Mensajería: utiliza mensajes cortos como método de comunicación entre el cliente y el servidor.

- Presencia: es reactivo a la presencia del usuario y el estado del usuario.

- Protocolo: no es un lenguaje. Es una plataforma abierta que está en constante evolución. Está basado en XML y es asíncrono.

Así como el protocolo XMPP existen otros protocolos diseñados para la comunicación entre dispositivos IOT, como el protocolo MQTT, el cual es particularmente importante ya que junto con XMPP son ampliamente usados en los dispositivos IoT, MQTT (Message Queuing Telemetry Transport) el cual va en la versión 5, es un protocolo de conectividad máquina a máquina (M2M), este fue diseñado para el transporte de mensajes de publicación y suscripción, que se caracteriza por ser extremadamente ligero. Es útil para conexiones con ubicaciones remotas donde se requiere un despliegue y transporte de información ligero en donde la capacidad de ancho de banda no es un limitante, este está definido por la norma ISO/IEC 20922 del 2016 (ISO/IEC, 2016).

En la versión 5, se han realizado algunas mejoras considerables resumidas a continuación.

- Mejor informe de errores
- Más propiedades del mensaje
- Caducidad de la sesión de mensajes
- Reducción de tamaño de los mensajes
- Estado actual de los mensajes

Pero, así como se mencionó previamente, existen protocolos esenciales para que la comunicación entre los dispositivos IOT función de manera óptima, como lo son:

- AMQP: Advanced Message Queuing Protocol

  Es un protocolo de internet abierto para mensajes comerciales. Utiliza un modelo de comunicación de publicación / suscripción similar a MQTT. (OASIS, 2012)

- DDS
  Es un protocolo de capa media que permite la intercomunicación de sistemas y aplicaciones. (OMG, 2015)

- JMS: Java Message Service
  Es una API, que permite la intercomunicación entre aplicaciones.

- CoAP: Constrained Application Protocol

  Es un protocolo de transferencia web especializado para usar con nodos y redes limitadas por red. (IEEE, 06)

- UPnP: Universal Plug and Play

  Es un conjunto de protocolos los cuales permiten descubrimiento de dispositivos en una red. (IEEE, 2013)

Adicionalmente es necesario contar con una comparación entre los modos de comunicación, la seguridad, la arquitectura, la priorización de mensajes a través de QoS y el protocolo en capa 4 de OSI, que estos implementan, los cuales se comparan a continuación en la tabla 1.

Tabla 1 - Comparación de protocolos de aplicaciones (DIZDAREVIĆ, 2019)

Protocolo	Arquitectura	OSI/4	QoS	Seguridad en transporte
REST HTTP	Req-Rep	TCP	-	TLS/SSL
MQTT	Pub-Sub	TCP	3 Niveles	TLS/SSL
CoAP	Ambas	UDP	Limitado	DTLS
AMPQ	Ambas	TCP	3 Niveles	TLS/SSL
DDS	Pub-Sub	TCP/UDP	Extensivo	TLS/SSL/DDS
XMPP	Pub-Sub	TCP	-	TLS/SSL
HTTP 2.0	Pub-Sub	TCP	-	TLS/SSL

En cuanto a los tipos de arquitecturas de interacción que implementan en dispositivos IoT encontramos:

- Requerimiento – Repuesta

  Este modelo de comunicación es una de las arquitecturas de comunicación más básicos el cual permite un intercambio de mensajes especialmente común en arquitecturas cliente-servidor.

- Publicación - Subscripción

Este modelo nace como necesidad de proporcionar a esas comunicaciones de Requerimiento y respuesta, una comunicación distribuida, asíncrona y poco acoplada entre los generadores de datos y los destinos.

1.5.4. Seguridad en IoT

IoT es un conjunto de tecnologías de hardware y software compuesta por protocolos de comunicación entre ellos los resumidos anteriormente, estos protocolos y el hardware que componen los dispositivos IoT, son objeto de ataques de red como malware, intuitivamente, el malware es un software que ataca de manera dañina a otro software donde se puede observar que atacar de manera nociva significa que el comportamiento real difiere del comportamiento previsto (Kramer, 2009), de acuerdo a esta definición se busca que los dispositivos IoT, funcionen de una maneta no usual con el fin de ejecutar acciones para las cuales no están diseñados, por ejemplo hacer parte de una botnet, una botnet es una red de dispositivos tecnológicos incluido IoT, los cuales han sido secuestrados por medio de ataques informáticos, ya sea despliegue de malware o ataques de fuerza bruta, para ejecuten ataques o

acciones en conjunto generando ataques distribuidos por ejemplo denegación o bloqueo de un servicio, este ataque es comúnmente llamado DDOS, así como tipo de ataques como ejecución remota de código o comandos y robo de información confidencial (Bertino, 2010).

Para lograr con éxito los ataques a los dispositivos IoT, se requiere de aprovecharse de una vulnerabilidad en un sistema IoT, las vulnerabilidades son debilidades en un sistema informático el cual le permite a un atacante tomar control del dispositivo o realizar acciones no permitidas (Pipkin, 2000), y los sistemas IoT se basan en dos componentes principales; hardware y software del sistema.

Estos dos componentes tienen fallas de diseño las cuales son evidenciadas con frecuencia, las vulnerabilidades de hardware son muy difíciles de identificar y también difíciles de corregir, incluso si la vulnerabilidad se identificó debido a la compatibilidad e interoperabilidad del hardware y también al esfuerzo que se requiere para solucionarlo a diferencia del hardware se pueden encontrar vulnerabilidades de software en sistemas operativos, software de aplicaciones y software de control, como protocolos de comunicación los cuales componen los descritos anteriormente que hacen parte de IoT.

Existen varios factores que conducen a fallas en el diseño del software, incluidos los factores humanos y la complejidad del software, estas usualmente son originadas por fallas humanas debido a que generalmente no se comprenden los requisitos al comenzar el proyecto, no se cuenta con un plan, se tiene una comunicación deficiente entre los desarrolladores y los usuarios, la falta de recursos, habilidades y conocimiento, y la falta de administración y control del sistema son factores que desencadenan la aparición de vulnerabilidades (Kizza, 2013).

1.5.4.1. Ataques a IoT

Debido al conocimiento público o al descubrimiento de las vulnerabilidades descritas anteriormente, se ejecutan ataques con el fin de aprovecharse de estas vulnerabilidades, los ataques informáticos, son acciones tomadas para dañar un sistema o interrumpir las operaciones normales explotando vulnerabilidades utilizando diversas técnicas y herramientas. Los atacantes lanzan

ataques para lograr objetivos, ya sea para satisfacción personal o recompensa (Schneier, 2011).

Estos ataques a dispositivos IoT, pueden venir en múltiples formas, los cuales como se describió antes, causar denegación de servicio, revelación de información confidencial, interceptación de información entre otros, a continuación, se describen los ataques más comunes a dispositivos IoT.

Ataques físicos: este tipo de ataque es un poco más complejo de ejecutar debido a la naturaleza desatendida y distribuida de IoT, aunque en el caso de sensores públicos la mayoría de los dispositivos suelen funcionar en entornos al aire libre, que son muy susceptibles a los ataques físicos por su exposición.

Ataques de reconocimiento: descubrimiento e identificación de sistemas, servicios o vulnerabilidades en sistemas IoT, este tipo de ataques pueden ser descubrimiento de puertos TCP/UDP, análisis de vulnerabilidad, rastreadores de paquetes, análisis de tráfico y Footprinting pasivo y activo sobre el dispositivo IoT que se quiere vulnerar (M. De Vivo, 1999).

Minería de datos: permite a los atacantes descubrir información que no se anticipa en ciertas bases de datos.
Espionaje cibernético: uso de técnicas de craqueo y software malicioso para espiar u obtener información secreta de individuos, organizaciones o el gobierno.

Denegación de servicio (DoS) o Denegación de servicio distribuido (DDoS): este tipo de ataque pretende ejecutar que un recurso tecnológico o recurso de red no esté disponible para los usuarios que lo requieren, en el universo IoT este es uno de los ataques más comunes ejecutado por ejemplo por la Botnet Mirai (Kolias, 2017). Debido a las bajas capacidades de memoria y los recursos de computación limitados, la mayoría de los dispositivos en IoT son vulnerables a los ataques de denegación de servicio, o son usados en contra de otros dispositivos de red para causar DoS o DDoS.

Ataques a la privacidad: la protección de la privacidad en IoT se ha convertido en un desafío cada vez mayor debido a los grandes volúmenes de información fácilmente disponibles a través de mecanismos de acceso remoto, por ejemplo el espionaje cibernético hace uso de técnicas de craqueo y software malicioso para espiar u

obtener información secreta haciendo uso de malware y aprovechándose de vulnerabilidades de los dispositivos IoT (abomhara, 2015).

Adicionalmente a los ataques vistos anteriormente, existen los ataques basados en contraseña, estos ataques son muy comunes en el mundo de IoT debido a múltiples factores que los atacantes suelen ejecutar, por ejemplo, intentan duplicar una contraseña de usuario válida, este intento se puede hacer de dos maneras diferentes:

Ataque de diccionario: el cual consiste en probar posibles combinaciones de letras y números para adivinar las contraseñas de los usuarios, estos diccionarios suelen tener en su base las credenciales que los dispositivos IoT tienen por defecto, por lo que muchas veces son satisfactorio debido a la no interacción del usuario con el dispositivo IoT y la poca conciencia de seguridad TI, adicionalmente existen los ataques de fuerza bruta, estos se ejecutan utilizando herramientas de craqueo con el fin de probar todas las combinaciones posibles de contraseñas para descubrir contraseñas válidas.
Delitos cibernéticos: Internet y los objetos inteligentes como IoT, son usados para explotar a los usuarios de la red de Internet y los datos para obtener ganancias materiales, como el robo de propiedad intelectual, el robo de identidad, el robo de marca y el fraude electrónico, incluyendo fraude financiero o robo de credenciales (Wilson, 2008).

Usar troyanos, virus, gusanos con el fin de tomar el control del sistema estos también incluidos en los malware, comúnmente usados por Botnets IoT como Aidra, Bashlite y Mirai. (García, 2019)

1.5.5. HoneyPot

Honeypot es un sistema de defensa para la seguridad de una red. Su principal función es atrapar los ataques, adicionalmente registra la información de intrusión sobre las herramientas y actividades del proceso de ataque, y evita que los ataques salgan del sistema comprometido.

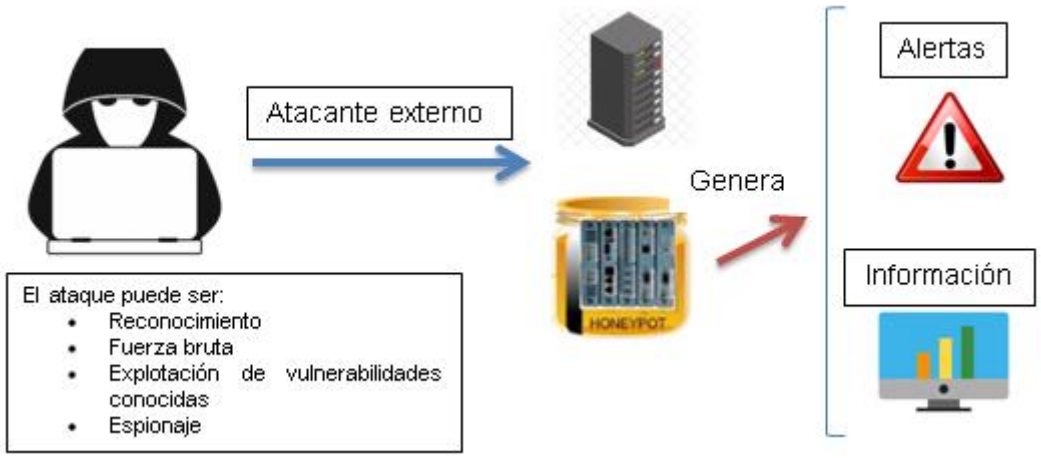


Imagen 4 - Estructura de una red con HoneyPot

Se recomienda hacer uso de un Honeypot en conjunto con otras herramientas de red, como firewalls, antivirus, IDS entre otros elementos de defensa de la red.

Según Lance Spitzner, "Honeypot es un recurso de seguridad cuyo valor reside en ser sondeado, atacado o comprometido" (Spitzner, 2002), según esta definición se podría definir que si este no es atacado o escaneado se puede inferir que no es valioso, pero igualmente un honeypot es una valiosa herramienta ya que actúa de manera diferente a otras herramientas de seguridad, como los firewall e IDS, estos son completamente pasivos, ya que su tarea es prevenir o detectar ataques.

Un Honeypot da paso al atacante para obtener información sobre nuevas intrusiones, esta naturaleza hace que honeypot sea indispensable para ayudar a otras herramientas de seguridad adicionalmente los Honeypot se caracterizan por integrarse de manera sencilla con otras herramientas, en la presente investigación se integrara con un IDS para crear un sistema de defensa activa que recolecte información relevante sobre ataques a dispositivos IoT.

Por lo tanto, definimos honeypot en tres pliegues. Como un recurso de seguridad cuyo valor radica en ser escaneado, atacado o comprometido, como una herramienta de seguridad cuyo valor radica en atraer activamente los ataques para obtener información de intrusión y mejorar el rendimiento de otras herramientas de seguridad como IDS, como una tecnología cuyo valor radica en ser un método alternativo para la seguridad de la red.

Para los casos en los que se requiera una adaptación del Honeypot a tecnologías emergentes como IoT, este podría ejecutarse y desplegarse de múltiples formas, podría ser una aplicación emulada, un sistema operativo funcional completo con configuración predeterminada o una red real que incluye diferentes sistemas operativos y aplicaciones, incluso una red emulada en una sola máquina, es por esto que su principal función es atraer atacantes y flujo de tráfico por lo que toda conexión entrante y saliente debe ser registrada y se puede considerar como sospechosa. (Zhang, 2003)

Estas múltiples formas de las que se describe anteriormente pueden ser clasificadas entre altas medias y bajas interacciones, dependiendo su exposición y su capacidad de alojar información, estas se describen en la imagen número 5.

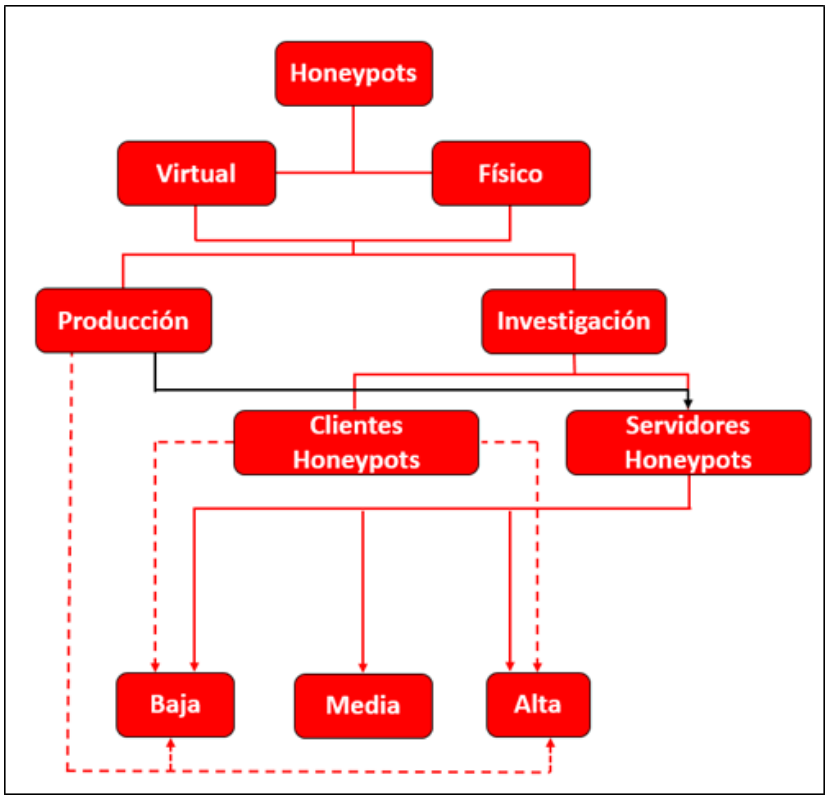


Imagen 5 - Tipos de HoneyPot - Tomado de: https://www.incibe-cert.es/blog/honeypots-industriales

Mayoritariamente se puede determinar que los HonetPot de baja y alta interacción cuentan con las siguientes características:

Baja Interacción:

- Muy bajo nivel de interacción ante un ataque

- Se requiere respuestas manuales para cada interacción
- No es completamente autónomo

Alta Interacción:

- Suele basarse en sistemas operativos existentes
- Se obtienen respuestas reales a través de un sistema simulado o físico
- Altamente personalizable

1.5.6. HoneyPot en IoT

Ante las crecientes amenazas a dispositivos en la red y particularmente dispositivos IoT, surge la necesidad de realizar investigaciones con el fin de evaluar el estado de seguridad en las que se encuentran este tipo de dispositivos.

Y es que en trabajos de investigación previos, se ha identificado que alrededor de un 13% de los dispositivos IoT de una colección de cerca de 157,000 dispositivos IoT, tienen vulnerabilidades de grado, bajo, medio, alto o critico; lo que lo convierte en un campo para proteger y analizar a través de dispositivos de detección y prevención como un IDS y un HoneyPot.

Los resultados de este análisis (William, 2017), demostraron que de los 20,300 dispositivos con vulnerabilidades cerca del 40 % eran dispositivos de video como Cámaras web, pero también se encontraban Televisores inteligentes e impresoras, como se describe en la siguiente tabla.

Tabla 2 - Vulnerabilidades en dispositivos IoT

Tipo de Dispositivo	# de dispositivos con vulnerabilidades descritas (Critica, Alta y media)	Vulnerabilidades más comunes
Webcams (8,127)	4961	MiniUPnP, NAT-PMP, Servidores Telnet desprotegidos
Smart TV's (1,437)	1174	Vulnerabilidades en el protocolo SNMP y algoritmos débiles en SSH
Printers (10,675)	17324	Vulnerabilidades en el protocolo SNMP y Servidores Telnet desprotegidos

Este tipo de investigaciones ayuda determinar que el universo IoT está creciendo rápidamente como una colección de dispositivos conectados con el propósito de crear hogares, ciudades e infraestructuras más inteligentes.

Si bien hay muchos beneficios y comodidades adicionales que brindan estos dispositivos más inteligentes, también invaden nuestra privacidad en una escala sin precedentes.

Ante las crecientes amenazas a dispositivos IoT, debido a la cantidad de vulnerabilidades que estos dispositivos cuentan, y los ataques antes mencionados en el apartado “1.5.4.1 Ataques a IoT”, nace la necesidad de crear un dispositivo que permita identificar y evaluar los ataques en tiempo real, creando una simulación de un dispositivo IoT expuesto en la red de internet, de ahí nace la necesidad de la creación del esquema de un HoneyPot de tipo IoT.

Para esto se han realizado estudios previos los cuales han permitido determinar los tipos de ataques a los que se exponen estos dispositivos y de esta forma identificar vulnerabilidades de día cero con el fin de remediar de una manera rápida y evitar el despliegue de ataques a nivel mundial, Honware (Clayton, 2019), es un HoneyPot virtual el cual fue desarrollado de manera satisfactoria y permitió determinar lo expuestos que estaban los dispositivos IoT en la red, a continuación se describen algunas de las vulnerabilidades y ataques descubiertos en esta investigación.

Broadcom UPnPHunter: UPnPHunter es el nombre de una vulnerabilidad de cadena de formato en el software Broadcom para Universal Plug and Play (UPnP) y afecta a varias marcas, como vimos esta vulnerabilidad se relaciona con la investigación descrita en la tabla numero 2 donde se evidencian debilidades en el protocolo UPnP.

DNS hijack: Este ataque, se trata de modificar de manera arbitraria los servidores o registros DNS de los equipos vulnerados.

UPnPProxy: EternalSilence es una familia recién descubierta de malware de reenvío UPnPProxy (Seaman, 2018), de nuevo se evidencia que UPnP es un protocolo vulnerable y común en dispositivos IoT como lo descrito en el apartado “1.5.3 Características de comunicación IoT”.

Variantes de Mirai: Se han encontrado en esta investigación que existen variantes activas del Malware con el que se creó la Botnet Mirai, la cual permite ejecución de código remoto en el dispositivo IoT.

Debido a la creciente cantidad de vulnerabilidades de seguridad en los Dispositivos IoT, se han desarrollado técnicas con el fin de emular las acciones realizadas por el Firmware de estos dispositivos, es por esto que ha nacido el termino, *rehosting*, todo esto se trata de poder extraer el firmware de los dispositivos e implantarlos en ambientes simulados que no sean el hardware para el que han sido diseñados, esto supone un gran reto ya que muchas de las interacciones hacia el software son ejecutadas por acciones de hardware, hay métodos que guardan las acciones del hardware para después replicas en ambientes simulados, hay otros que como los HALS que son librerías de software que proveen operaciones de hardware al desarrollador.

Para lograr esta emulación de hardware y poder ejecutar software de manera r- hosting, es necesario apoyarnos en herramientas de código abierto como QEMU (TheQEMUProjectDevelopers, 2020), este es un es un emulador y virtualizador de máquinas genérico y de código abierto.

Cuando se usa como un emulador de máquina, QEMU puede ejecutar sistemas operativos y programas creados para una máquina (por ejemplo, una placa ARM) en una máquina diferente. Mediante el uso de la traducción dinámica, logra un rendimiento muy bueno.

Cuando se usa como virtualizador, QEMU logra un rendimiento casi nativo al ejecutar el código de invitado directamente en la CPU del host. QEMU admite la virtualización cuando se ejecuta bajo el hipervisor Xen o cuando se usa el módulo del kernel KVM en Linux. Al usar KVM, QEMU puede virtualizar arquitecturas como x86, y PowerPC integrado, POWER de 64 bits, S390, ARM de 32 y 64 bits, y MIPS.

Arquitecturas de procesadores en IoT

Estas arquitecturas usualmente están basadas en microcontroladores, debido a la reducida necesidad de procesamiento y espacio, un microcontrolador se entiende como una computadora pequeña en un solo chip de circuito integrado de semiconductor de óxido de metal (MOS). Un microcontrolador contiene una o más CPU junto con memoria y periféricos de entrada / salidas programables, estos periféricos en el entorno IoT son los comúnmente llamados sensores.

Debido a que la presenta investigación tiene como objetivo emular firmware de dispositivos como si estuvieran en un entorno real, se identificó la arquitectura más común en dispositivos IoT.

Arquitectura ARM

“El procesador de arquitectura ARM es una máquina avanzada de computación de conjunto de instrucciones reducidas [RISC] y es un microcontrolador de computadora de conjunto de instrucciones reducidas (RISC) de 32 bits.” (Science Direct, 2020)

Y porque se determina que la arquitectura ARM es la más implementada en el mundo IoT, es porque se ha determinado que fabricar CPUs ARM resulta más barato, dado que pueden usar procesos que requieren menor miniaturización y hay más fabricantes en condiciones de hacerlo, debido que ARM como tal se ha encargado de diseñar la arquitectura más no a fabricarla adicionalmente, la disipación de calor es menory es por esto que la operación normal de un procesador ARM resulta mucho más aceptable para un equipo de perfil delgado, y más aún cuando está diseñado para llevarse en contacto con el cuerpo de su usuario, por ejemplo en relojes inteligentes o dispositivos IoT que tienen poco espacio de ventilación, adicionalmente, el consumo eléctrico es menor, permitiendo mucho mayor tiempo de operación a baterías o de consumo eléctrico en dispositivos que su función principal no es procesar información, el tamaño de los chips resultantes es menor en comparación con un CPU Intel para equipos de escritorio mide unos 4cm por lado, los procesadores ARM son 50% más pequeños. (Arm Limited, 2020)

2. Requerimientos técnicos IoT

2.1. Requerimientos de hardware

Con el fin de ejecutar la implementación de un conjunto de HoneyPot IoT, se requiere que los HoneyPot se implementen en un hardware real o virtualizado, para esto es posible hacer uso de unos equipos con las siguientes características funcionales:

Servidor de Virtualización

- Intel Core i7-9750H de 9na generación
- Tarjeta de video NVIDIA GTX 1650 4GB GDDR5
- 16 GB DDR4 2666 MHz
- 1TB de Disco Duro
- 128 GB Disco Estado solido
- USB-C
- Mini DisplayPort 1.4
- 3 puertos USB 3.1 de 1ª Generación
- HDMI™ 2.0
- Puerto Ethernet
- IEEE 802.11b, IEEE 802.11g, IEEE 802.11n
-

Equipo de cómputo

- CPU Intel Core i5 (3.a generación) 3210M / 2.5 GHz
- Velocidad de memoria RAM: 1600 MHz
- Tecnología SDRAM DDR3
- Tamaño instalado 8 GB
- Capacidad del disco duro 750 GB
- LAN Ethernet, Fast Ethernet, Gigabit Ethernet, IEEE 802.11b, IEEE 802.11g, IEEE 802.11n
- VGA
- HDMI
- 2 x USB 3.0
- USB 2.0

Software de virtualización de código abierto

Entorno virtual Proxmox, Software de virtualización de servidor de código abierto.

Proxmox VE es una plataforma completa de código abierto para la virtualización empresarial. Con la interfaz web incorporada, puede administrar fácilmente máquinas virtuales y contenedores, almacenamiento y redes definidos por software, clustering de alta disponibilidad y múltiples herramientas listas para usar en una sola solución.

Software de virtualización VmWare WorkStation Pro

VMware Workstation Pro es el estándar del sector para la ejecución de varios sistemas operativos en un único equipo Linux o Windows.

Workstation 15.5 Pro mejora el escritorio líder al ofrecer nuevos controles en la interfaz de usuario, así como compatibilidad con trama gigante y con los sistemas operativos Windows y Linux más recientes, entre otras actualizaciones. (VMware, Inc, 2020)

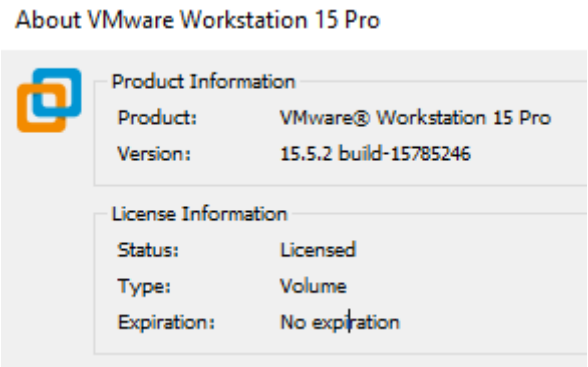


Imagen 6 - Licencia Instalada VmWare WorkStation

Se hizo uso de una licencia PRO para el uso de WorkStation en el servidor de Virtualización antes descrito, con esto se podrá virtualizar múltiples ambientes de manera paralela teniendo la posibilidad de crear puntos de chequeo o imágenes en cierto punto de avance de la solución y de los ambientes creados.

2.2. Soluciones de Software

Como se desarrolló la sección de marco teórico y estado del arte, actualmente existen múltiples soluciones de Honeypot de baja y alta interacción, a continuación, se describen algunos de ellos, los cuales serán valiosos para el desarrollo de la presente investigación.

En cuanto a soluciones de software, es posible basar la investigación en algunas soluciones de código abierto e investigaciones previas disponibles en la red, a continuación, listo algunos proyectos de interés.

- Dionea
- Kippo
- Cowrie
- Telnet IoT HoneyPot
- IoT Pot
- HoneyD
- Honeware Framework

Estos proyectos han sido desarrollados satisfactoriamente con el objetivo de obetener informacion relevante sobre los ataques y ofensas de seguridad que viven los dispositivos IoT en el mundo actual.

2.3. Selección de herramientas para la investigación

Con el fin de cumplir con los objetivos propuestos en la investigación, se ha determinado hacer uso de los HoneyPot establecidos como Código abierto y que permiten dar conclusiones luego de su despliegue, debido a que estas permiten evaluar la cantidad de ataques hacia protocolos en entornos IoT más vulnerables del mundo, esto en base a la tabla número 2, por lo que desplegarán soluciones que expondrán protocolos vulnerables como Telnet.

2.3.1. Selección de herramientas

Tras la investigación realizada evaluando los HoneyPots de Código abierto disponibles en internet mencionados antes y las necesidades de la investigación, Telnet IoT HoneyPot y IoT Pot, serán las herramientas que permitirán en primera instancia recuperar información relevante al ser expuestos a una red externa.

Telnet IoT HoneyPot

Implementa un servidor telnet desarrollado en Python que intenta actuar como un honeypot para IoT Malware que se propaga a través de contraseñas predeterminadas horriblemente inseguras en los servidores telnet en Internet (Phype, 2016).

IoT Pot

IoTPOT consiste en un honeypot IoT y un sandbox para ataques Telnet. Emula los servicios Telnet de varios dispositivos. Consiste en dos partes: una interfaz que responde con poca interacción y un backend que es un entorno virtual de alta interacción llamado IoTBOX. IoTBOX admite 8 arquitecturas de CPU diferentes, incluidas MIPS y ARM (Suzuki, 2016).

Debido a que el presente trabajo tiene como fin el rehosting de firmware encontrados en los dispositivos IoT más encontrados en el mercado y en las arquitecturas de procesadores más comunes, se han seleccionado las siguientes herramientas en las cuales se basarán las pruebas de emulación y análisis dinámico de los firmware IoT, esto debido a que las técnicas dinámicas de análisis binario juegan un rol importante para estudiar la seguridad de los sistemas de software y detectar vulnerabilidades en una amplia gama de dispositivos y aplicaciones incluyendo el Mundo IoT.

Simics

Esta herramienta simula sistemas, desde el más pequeño hasta el más complejo, para que pueda adoptar nuevas técnicas de desarrollo que simplemente no son posibles con el hardware físico. Al pasar del hardware físico a un laboratorio virtual, los equipos de software integrados pueden revolucionar aspectos cruciales de sus procesos, permitiéndoles entregar un mejor software más rápido.

FiE

FiE es un motor de ejecución simbólico basado en KLEE, específicamente dirigido a encontrar vulnerabilidades en el firmware de los microcontroladores MSP430.

Avatar y Avatar2

Avatar es un marco de orquestación de destino con enfoque en el análisis dinámico del firmware de los dispositivos integrados, diseñado para permitir la interoperabilidad entre diferentes marcos de análisis binarios dinámicos, depuradores, emuladores y dispositivos físicos reales.

PROSPECT

Permite el análisis dinámico de código de código binario incorporado dentro de entornos de análisis arbitrarios. Al reenviar de forma transparente los accesos de hardware periférico desde el sistema host original a una máquina virtual, PROSPECT permite a los analistas de seguridad ejecutar la implementación de software incorporado sin la necesidad de saber a qué y cómo se accede a los componentes de hardware periférico integrados.

P retender

Utiliza observaciones de las interacciones entre el hardware original y el firmware para crear automáticamente modelos de periféricos, y permite la ejecución del firmware en un entorno totalmente emulado.

HALucinator

Proporciona un entorno de alto nivel de simulación usando el emulador QEMU. HALucinator admite firmware de múltiples proveedores de chips para la arquitectura ARM Cortex-M.

2.4. Firmware objetivo

A través de las herramientas anteriormente mostradas, se pretende emular distinta cantidad de firmware, el firmware para estos dispositivos generalmente se obtiene en forma de un formato binario, un objeto de código opaco que no contiene metadatos sobre su contenido.

Para la obtención de estos recursos IoT, se usará la plataforma de ARM, Mbed, esta plataforma permite la descarga y compilación de firmware para más de 150 boards de desarrollo, en las que comúnmente se implementan dispositivos IoT, Mbed ofrece un sistema operativo IoT de código abierto gratuito con funciones de red y seguridad incorporadas con los cuales se podrán hacer pruebas.

Permite seleccionar para que funciones utilizamos la plataforma con el fin de obtener acceso a más recursos.

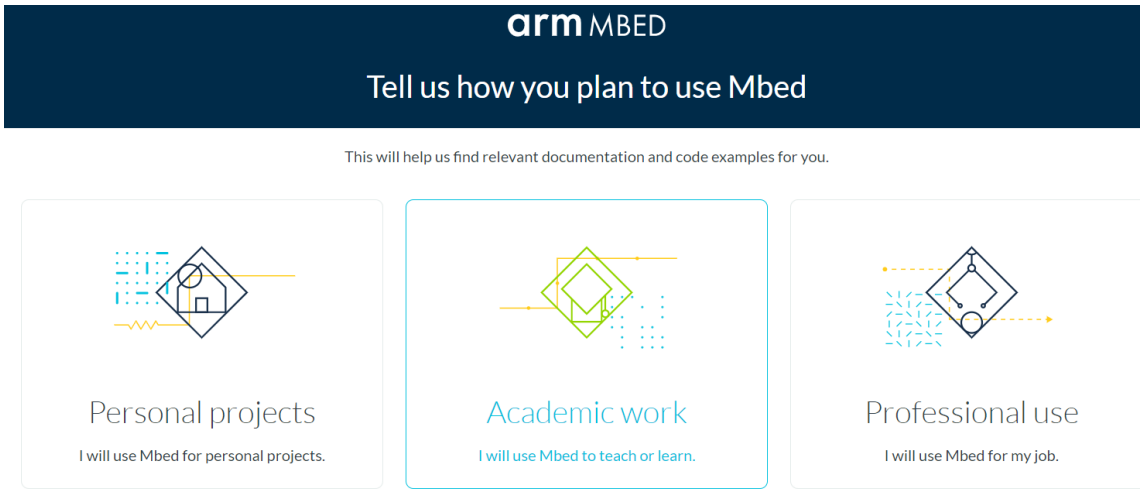


Imagen 7 - Planes de uso de ARM Mbed

Para la presente investigación se han venido haciendo pruebas con tres tarjetas, las núcleo F072RB, núcleo F070RB y las MAX3260, de acuerdo a esto seleccionamos estas tarjetas y permite entrar en el módulo de compilación de firmware.

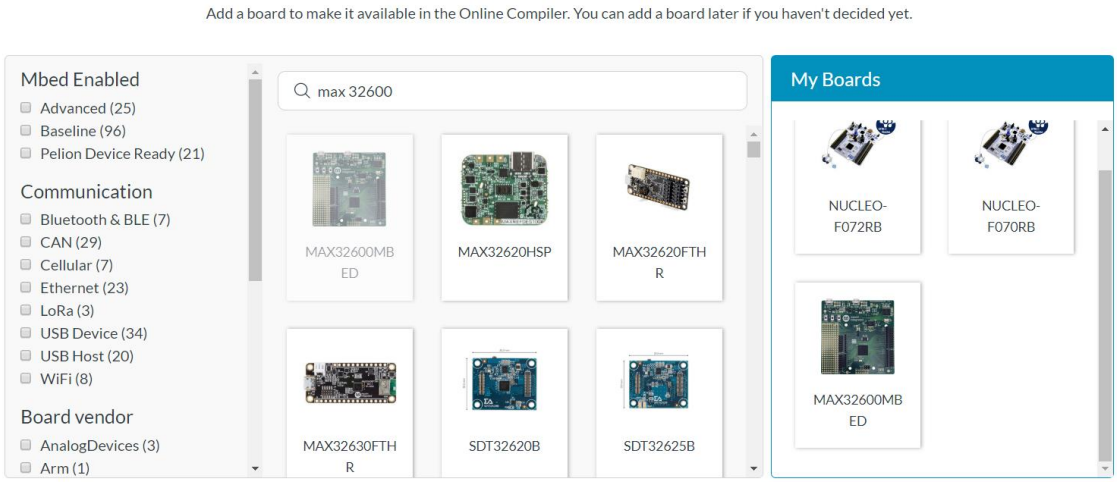


Imagen 8 - Boards seleccionadas para investigación.

NUCLEO F072RB y NUCLEO L152RE

La placa STM32 Nucleo proporciona una forma asequible y flexible para que los usuarios prueben nuevas ideas y construyan prototipos con cualquier línea de microcontroladores STM32, eligiendo entre las diversas combinaciones de rendimiento, consumo de energía y característica, esta implementa una microcontroladora STM32F072RBT6 de arquitectura ARM. (Arm Limited, 2020)

El soporte de conectividad Arduino ™ y los encabezados ST Morpho facilitan la expansión de la funcionalidad de la plataforma de desarrollo abierto STM32 Nucleo con una amplia variedad de escudos especializados.

Esta placa STM32 es compatible con las partes NUCLEO resumidas en la siguiente tabla.

Tabla 3 - Partes compatibles con la placa STM32 Nucleo

Referencia	Numero de parte
NUCLEO-XXXXRX	NUCLEO-F030R8, NUCLEO-F070RB, NUCLEO-F072RB, NUCLEO-F091RC, NUCLEO-F103RB, NUCLEO-F302R8, NUCLEO-F303RE, NUCLEO-F334R8, NUCLEO-F401RE, NUCLEO-F410RB, NUCLEO-F411RE, NUCLEO-F446RE, NUCLEO-L053R8, NUCLEO-L073RZ, NUCLEO-L152RE, NUCLEO-L452RE, NUCLEO-L476RG.

Caracteristicas de las placas STM32 compatibles con las NUCLEO resumidas anteriormente.

Microcontrolador STM32 en paquete QFP64 (farnell.com, 2016)
• Dos tipos de recursos de extensión:
- Conectividad Arduino ™ Uno V3
- Encabezados de pin de extensión ST morpho para full acceso a todas las E / S STM32
• ARM® mbed
• ST-LINK / V2-1 a bordo depurador / programador con conector

- Interruptor de modo de selección para usar el kit como ST LINK / V2-1 independiente
• Fuente de alimentación de placa flexible:
- USB VBUS o fuente externa
(3.3V, 5V, 7 - 12V)
- Punto de acceso de administración de energía
• Tres LED:
- Comunicación USB (LD1), LED de usuario
(LD2), LED de alimentación (LD3)
• Dos botones: USUARIO y RESET
• Capacidad de remuneración USB. Tres diferentes interfaces compatibles con USB:
- Puerto COM virtual
- Almacenamiento masivo
- Puerto de depuración

MAX32600MBED

El MAX32600MBED proporciona una plataforma conveniente para evaluar las capacidades del microcontrolador MAX32600. El MAX32600MBED también proporciona un sistema completo y funcional ideal para desarrollar y depurar aplicaciones comunes en IoT.

El MAX32600MBED incluye un microcontrolador Corte32®-M3 MAX32600, además de acceso de entradas y salidas a través de conectores compatibles con Arduino ™, acceso de entradas y salidas adicional a través de encabezados de 100mil x 100mil, interfaz USB, y otros dispositivos de entradas y salidas de uso general. (Maxim Integrated, 2015)

Las Caracteristicas de MAX32600MBED (Mbed, 2017) son:

- El AFE integrado permite mediciones de precisión
- ADC de 16 bits con entrada Mux y PGA Buffers programables de tasa de conversión de hasta 500ksps para ADC y DAC
- Dos DAC de 12 bits y dos DAC de 8 bits
- Cuatro amplificadores operacionales
- Cuatro comparadores de baja potencia
- Cuatro interruptores analógicos SPST
- Unidad de protección de confianza para seguridad de extremo a extremo
- Interfaz de dispositivo USB
- Controlador LCD
- El motor DMA de 6 canales permite una operación periférica inteligente mientras Micro está en modo de suspensión
- Cuatro temporizadores de 32 bits, configurables a 8 x 16 bits
- Reloj de tiempo real de 32 bits
- Tres Maestros SPI
- Dos UART
- Dos puertos maestros I2C
- Un puerto esclavo I2C
- Hasta 64 pines GPIO con interrupción externa y activación desde el modo de bajo consumo
- Motor de tren de pulsos con ocho canales de salida digital

En la presente investigación nos centraremos en la simulación en ambientes extraídos de los siguientes fimrware, que son descritos a continuación.

- blink_led

  Este firmware muestra lo más simple que puede hacer con una placa para ver la salida física: parpadea el LED incorporado.

  Este ejemplo utiliza el LED incorporado que tienen la mayoría de las placas Arduino y Genuino. Este LED está conectado a un pin digital y su número puede variar de un tipo de placa a otra. Para facilitarle la vida, tenemos una constante que se especifica en cada archivo descriptor de la placa. Esta constante es LED_BUILTIN y le permite controlar fácilmente el LED incorporado.

- read_hyperterminal

  Este firmware recibe entrada externa de un usuario u otro dispositivo a través de un puerto serie y enciende o apaga un LED ("1" o "0") según la entrada. Este ejemplo muestra una ejecución de firmware divergente basada en diferentes entradas, ya que un usuario puede enviar varias entradas posibles, en cualquier orden. (IEEE, 2015)

- button_interrupt

  Este firmware hace uso de interrupciones que se activan por un evento externo como un botón o una interacción externa, ejemplo, cuando se presiona el botón físico, provoca una interrupción para ejecutar una acción sobre un LED.

- thermostat

  Es un firmware que maneja un termostato típico, el firmware lee la temperatura y la humedad del sensor AM2315 y ahora también acepta comandos que sondean la temperatura y la humedad.

- rf_door_lock

  Este firmware utiliza un módulo de radio Grove Serial RF Pro conectado a un periférico receptor / transmisor asincrónico universal (UART), que acepta múltiples comandos. Entre otros, esos comandos incluyen "ping" y "desbloqueo", que aceptan una contraseña. Si la contraseña es correcta, el firmware activa un GPIO, que desbloquea un bloqueo mecánico hipotético.

3. Proceso de Re-Hosting

Para el proceso de re-hosting se ha desplegado diferentes herramientas que serán ilustradas a continuación y su utilidad frente a la investigación.

3.1. Elementos de virtualización/emulación

Se implementó una plataforma de virtualización usando el Software de licencia VMWARE PRO en la versión 15.5, este permite crear múltiples máquinas virtuales donde se desarrollará la investigación, en la siguiente imagen se evidencia la creación de dos máquinas virtuales para el desarrollo de la investigación.

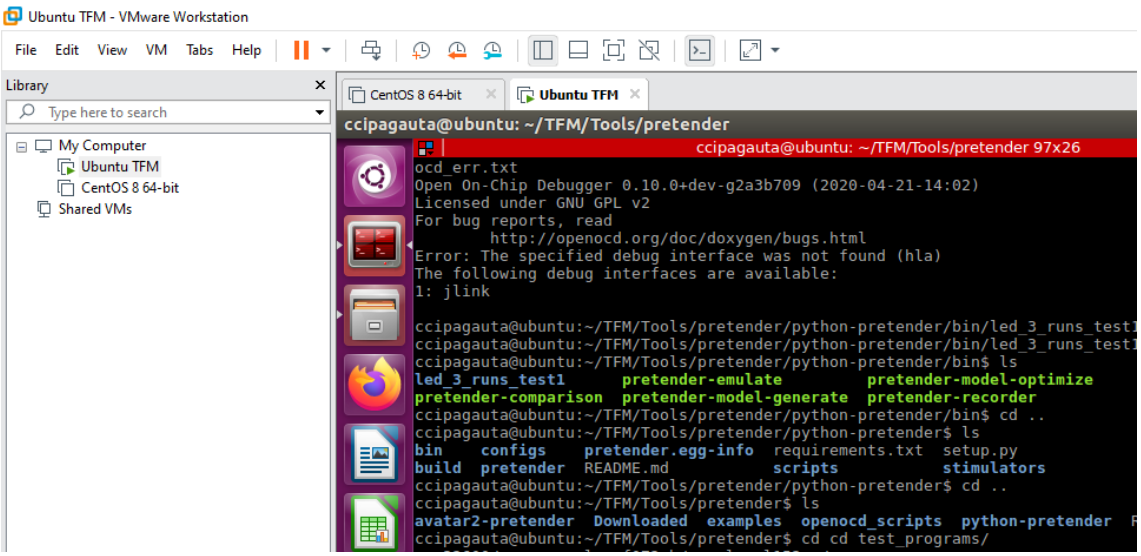


Imagen 9 - Plataforma de virtualización

Adicionalmente a la plataforma de virtualización se instalaron cuatro sistemas operativos de pruebas de acuerdo con las compatibilidades requeridas para las pruebas.

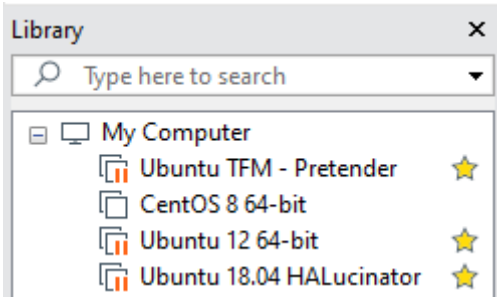


Imagen 10 - Maquinas virtuales instaladas

La primera es una máquina virtual que aloja el sistema Operativo Ubuntu en la versión 16.04, esta será usadas para la ejecución de la herramienta Pretender y tiene las siguientes características.

- 1 procesador y 2 cores asignados
- 4 GB de memoria RAM dedicada
- 70GB de disco duro
- Sistema operativo Ubuntu versión 16.04

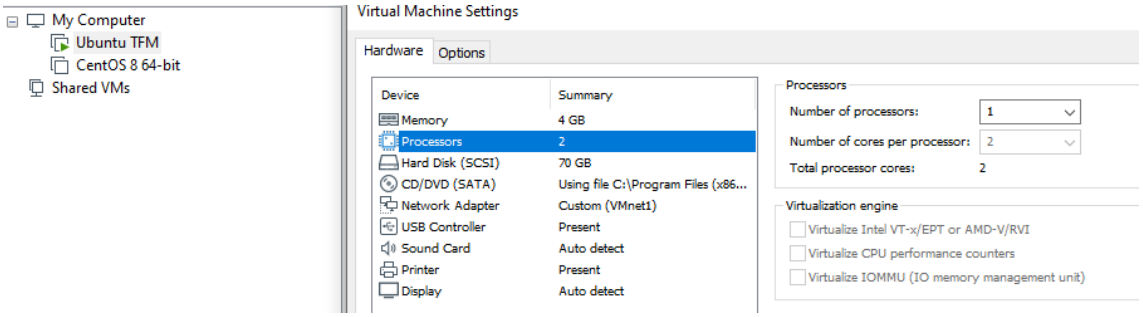


Imagen 11 - Caracteristicas de la máquina Ubuntu 16.04

La segunda es una máquina virtual que aloja el sistema Operativo en la versión CentOS en la versión 8, e esta tiene las siguientes características.

- 2 procesador y 2 cores asignados
- 4 GB de memoria RAM dedicada
- 100GB de disco duro
- Sistema operativo CentOS versión 8

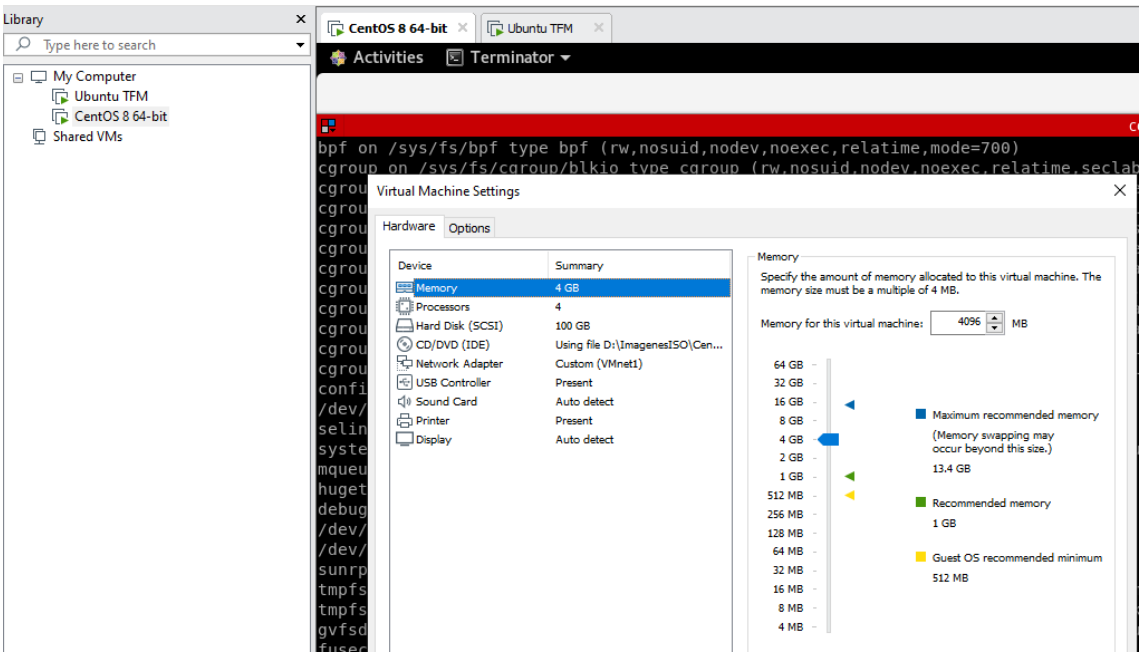


Imagen 12 - Caracteristicas maquina CentOS

La tercera es una máquina virtual que aloja el sistema Operativo en la versión Ubuntu 12.04 Con el fin de ejecutar pruebas para la herramienta FiE.

- 2 procesador y 2 cores asignados
- 4 GB de memoria RAM dedicada
- 60GB de disco duro
- Sistema operativo Ubuntu 12.04

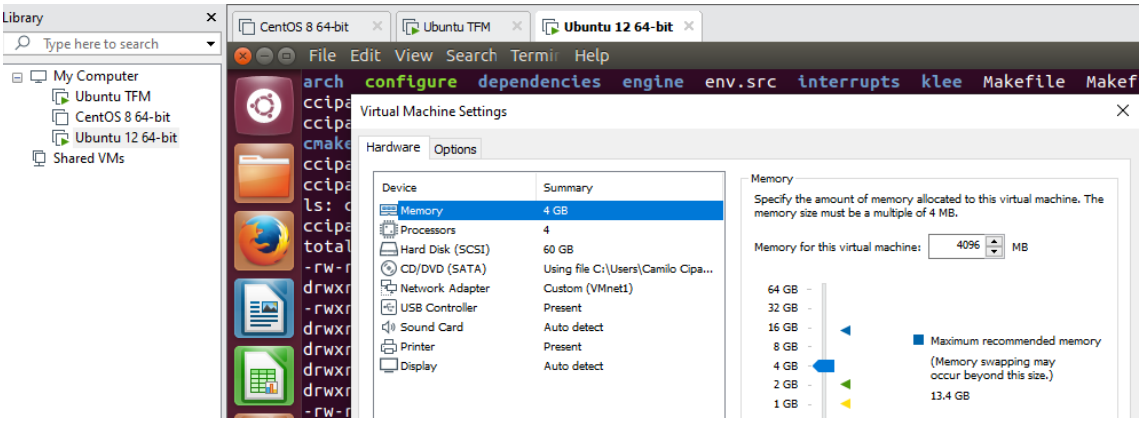


Imagen 13 - Ubuntu 12.04 instalado y ejecutando pruebas

Se ha elegido el Sistema operativo Ubuntu en su versión 16.04 debido a que el uso de herramientas seleccionadas en el capítulo 2.3 “selección de herramientas para la investigación” requieren el uso de versiones específicas de paquetes para obtener un resultado adecuado, por ejemplo, arm-none-eabi, requiere estar en la versión

7.10-1 con el fin de que herramientas como Pretender, Avatar y Avatar2 funcionen de manera correcta.

La cuarta es una máquina virtual que aloja el sistema Operativo en la versión Ubuntu 18.04 Con el fin de ejecutar pruebas para la herramienta HALucinator.

- 2 procesador y 2 cores asignados
- 4 GB de memoria RAM dedicada
- 500 GB de disco duro
- Sistema operativo Ubuntu 18.04

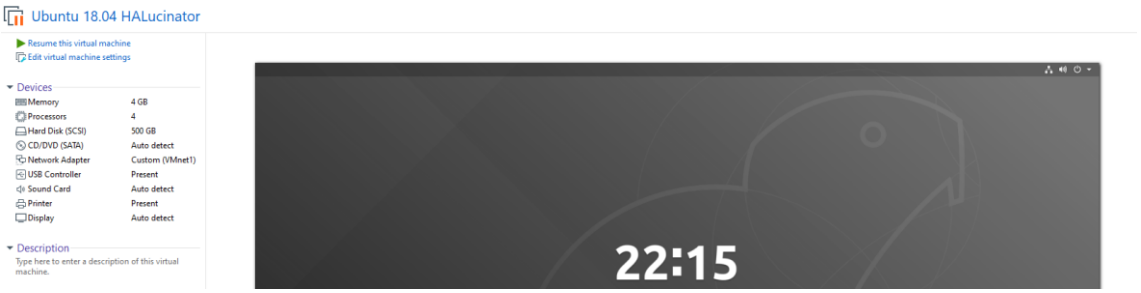


Imagen 14 - Maquina virtual Ubuntu 18.04 HALucinator

Otras herramientas usadas fueron:

arm-none-eabi, es la cadena de herramientas que utilizamos en esta investigación. Esta cadena de herramientas apunta a la arquitectura ARM, por lo que puede ser usada en múltiples pruebas de emulación y compilación. (Universidad de Michigan, 2012)

arm-none-eabi-gcc, nos permitirá compilar experimentos de firmwares que se vayan desplegando, este se requiere en la versión 4.9.3.

Para la presente investigación se desplegó la versión de Avatar 2 en la versión 1.3.5, aunque también se han realizado algunos experimentos con la versión 1.3.0.

Así mismo se desplego OpenOCD, en la versión de git edb6796, que de acuerdo con la documentación de Avatar2 y Pretender es estable para la ejecución de pruebas de re-hosting.

```
libjaylink configuration summary:
 - Package version ................. 0.2.0-git-8645845
 - Library version ................. 0:0:0
 - Installation prefix ............. /opt/TFM/openocd
 - Building on ..................... x86_64-pc-linux-gnu
 - Building for .................... x86_64-pc-linux-gnu

Enabled transports:
 - USB ............................. yes
 - TCP ............................. yes



OpenOCD configuration summary
--------------------------------------------------
MPSSE mode of FTDI based devices        yes (auto)
ST-Link JTAG Programmer                 yes (auto)
TI ICDI JTAG Programmer                 yes (auto)
Keil ULINK JTAG Programmer              yes (auto)
Altera USB-Blaster II Compatible        yes (auto)
Bitbang mode of FT232R based devices    yes (auto)
Versaloon-Link JTAG Programmer          yes (auto)
TI XDS110 Debug Probe                   yes (auto)
OSBDM (JTAG only) Programmer            yes (auto)
eStick/opendous JTAG Programmer         yes (auto)
Andes JTAG Programmer                   yes (auto)
USBProg JTAG Programmer                 yes (auto)
Raisonance RLink JTAG Programmer        yes (auto)
Olimex ARM-JTAG-EW Programmer           yes (auto)
CMSIS-DAP Compliant Debugger            no
Cypress KitProg Programmer              no
Altera USB-Blaster Compatible           yes (auto)
ASIX Presto Adapter                     yes (auto)
OpenJTAG Adapter                        yes (auto)
SEGGER J-Link Programmer                yes (auto)

ccipagauta@ubuntu:~/TFM/Tools/openocd$
```

Imagen 15 - Instalación y configuracipon de OpenOcd compatible con multiples programadores

Como se describió en la sección 2.4, se eligieron algunas placas base para la emulación de firmwares basadas en ellas, es por esto por lo que se ha hecho algunas simulaciones sobre firmware de pruebas como un simple encendido de leds, estos firmwares y sus modificaciones fueron descargados y compilados en el portal de ARM Mbed.

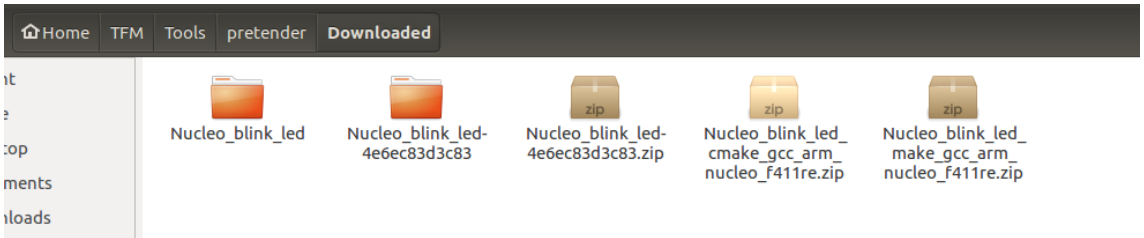


Imagen 16 - Firmware de pruebas compilados en AMR Mbed

Para la ejecución de estos firmwares de pruebas se estableció un ambiente de pruebas dentro del sistema operativo Ubuntu, para esto se hizo uso de una herramienta llamada virtualenv.

Virtualenv es una herramienta para crear entornos Python aislados, esta soportada desde la versión 2.7 hasta la versión 3.6 de python, actualmente en la presente investigación se hace uso de la versión 2.7 debido al sistema operativo instalado a los requerimientos de las herramientas. (DOUG HELLMANN, 2020)

Una vez establecido el ambiente de pruebas se ha iniciado con la ejecución de pruebas dentro de este, a través de la ejecución de pretender-recorder, hacemos uso de múltiples herramientas como Avatar y ejecutamos dentro del ambiente un Firmware de pruebas llamado Núcleo_blink_led, compilado para la tarjeta NUCLEO-F072RB.

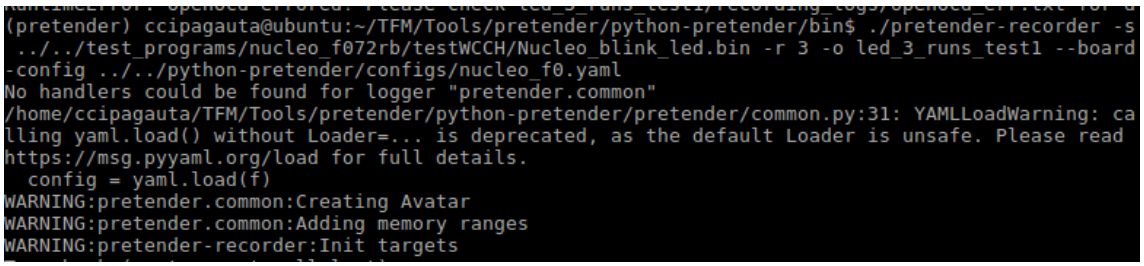


Imagen 17 - Emulación de firmware Blink_led en ambientes virtuales

La emulación del firmware blink_led se logra de manera satisfactoria, como se ilistró en la imagen 17 e imagen 18 con la cofngiuración adecuada de OCD.

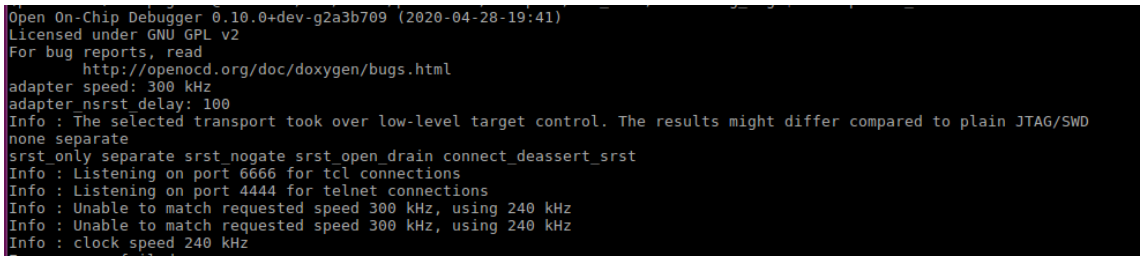


Imagen 18 - Open OCD emulación

Adicional a las pruebas que se están realizando con pretender, actualmente se están realizando pruebas con la herramienta Fie, como se describió previamente está permite encontrar vulnerabilidades en sistemas embebidos como sistemas IoT, FiE, usa como insumos algunas herramientas entre ellas Flex.

Flex

flex es una herramienta para generar escáneres: programas que reconocen patrones léxicos en el texto. flex lee los archivos de entrada dados, o su entrada estándar si no se dan nombres de archivos, para generar una descripción de un escáner, se realizó la instalación de Flex.

```
make[4]: Nothing to be done for `install-data-am'.
make[4]: Leaving directory `/home/ccipagauta/flex-2.5.37/tests/test-noansi-r'
make[3]: Leaving directory `/home/ccipagauta/flex-2.5.37/tests/test-noansi-r'
Making install in test-top
make[3]: Entering directory `/home/ccipagauta/flex-2.5.37/tests/test-top'
make[4]: Entering directory `/home/ccipagauta/flex-2.5.37/tests/test-top'
make[4]: Nothing to be done for `install-exec-am'.
make[4]: Nothing to be done for `install-data-am'.
make[4]: Leaving directory `/home/ccipagauta/flex-2.5.37/tests/test-top'
make[3]: Leaving directory `/home/ccipagauta/flex-2.5.37/tests/test-top'
Making install in test-table-opts
make[3]: Entering directory `/home/ccipagauta/flex-2.5.37/tests/test-table-opts'
make[4]: Entering directory `/home/ccipagauta/flex-2.5.37/tests/test-table-opts'
make[4]: Nothing to be done for `install-exec-am'.
make[4]: Nothing to be done for `install-data-am'.
make[4]: Leaving directory `/home/ccipagauta/flex-2.5.37/tests/test-table-opts'
make[3]: Leaving directory `/home/ccipagauta/flex-2.5.37/tests/test-table-opts'
make[3]: Entering directory `/home/ccipagauta/flex-2.5.37/tests'
make[4]: Entering directory `/home/ccipagauta/flex-2.5.37/tests'
make[4]: Nothing to be done for `install-exec-am'.
make[4]: Nothing to be done for `install-data-am'.
make[4]: Leaving directory `/home/ccipagauta/flex-2.5.37/tests'
make[3]: Leaving directory `/home/ccipagauta/flex-2.5.37/tests'
make[2]: Leaving directory `/home/ccipagauta/flex-2.5.37/tests'
make[1]: Leaving directory `/home/ccipagauta/flex-2.5.37'
ccipagauta@ccipagautaUBNT:~/flex-2.5.37$
```

Imagen 19 - Compilación de flex versión 2.5

Se procedió a instalar en la maquina Ubuntu 12.04 la solución FiE, compiló solucionando problemas durante el proceso.

```
make[3]: Entering directory `/home/ccipagauta/fie/memorymodels/symbolic/msp430g2231'
rm -f libregisters.so libinterrupts.so
g++ -c -fPIC -o registers.o registers.cpp -I/home/ccipagauta/fie/engine/include -I/home/ccip
MIT_MACROS -D__STDC_CONSTANT_MACROS
g++ -shared -Wl,-soname,libregisters.so.1 -o libregisters.so.1.0.1 registers.o -lc
ln -s libregisters.so.1.0.1 libregisters.so
clang -ccc-host-triple msp430-elf -I/home/ccipagauta/fie/plugin_gen/common -I/home/ccipagaut
make[3]: Leaving directory `/home/ccipagauta/fie/memorymodels/symbolic/msp430g2231'
make[3]: Entering directory `/home/ccipagauta/fie/memorymodels/symbolic/msp430f2274'
rm -f libregisters.so libinterrupts.so
g++ -c -fPIC -o registers.o registers.cpp -I/home/ccipagauta/fie/engine/include -I/home/ccip
MIT_MACROS -D__STDC_CONSTANT_MACROS
g++ -shared -Wl,-soname,libregisters.so.1 -o libregisters.so.1.0.1 registers.o -lc
ln -s libregisters.so.1.0.1 libregisters.so
clang -ccc-host-triple msp430-elf -I/home/ccipagauta/fie/plugin_gen/common -I/home/ccipagaut
make[3]: Leaving directory `/home/ccipagauta/fie/memorymodels/symbolic/msp430f2274'
make[2]: Leaving directory `/home/ccipagauta/fie/memorymodels/symbolic'
make[1]: Leaving directory `/home/ccipagauta/fie/memorymodels'
ccipagauta@ccipagautaUBNT:~/fie$ cat /etc/*release
```

Imagen 20 - Insttalación y compilación de FiE

HALucinator

Permite la simulación de firmware a través de capas de abstracción de hardware (HAL), las cuales son usualmente usados implementadas por desarrolladores, de esta manera se permite la simulación a través de reemplazos de alto nivel para las funciones HAL (un proceso denominado emulación de alto nivel - HLE), desacoplamos el hardware del firmware.

Imagen 21 - Configuración de ambiente HALucinator Ubuntu 18.04

Luego de la instalación de la herramienta HALucinator, se procede a la emulación de firmware simulando sobre el microcontrolador stm32f4 UART, en la comunicación UART, dos UART se comunican directamente entre sí. El UART transmisor convierte datos paralelos de un dispositivo de control como una CPU en forma serial, los transmite en serie al UART receptor, que luego convierte los datos seriales nuevamente en datos paralelos para el dispositivo receptor, lo que se simula es la recepción y envío de datos a través de UART.

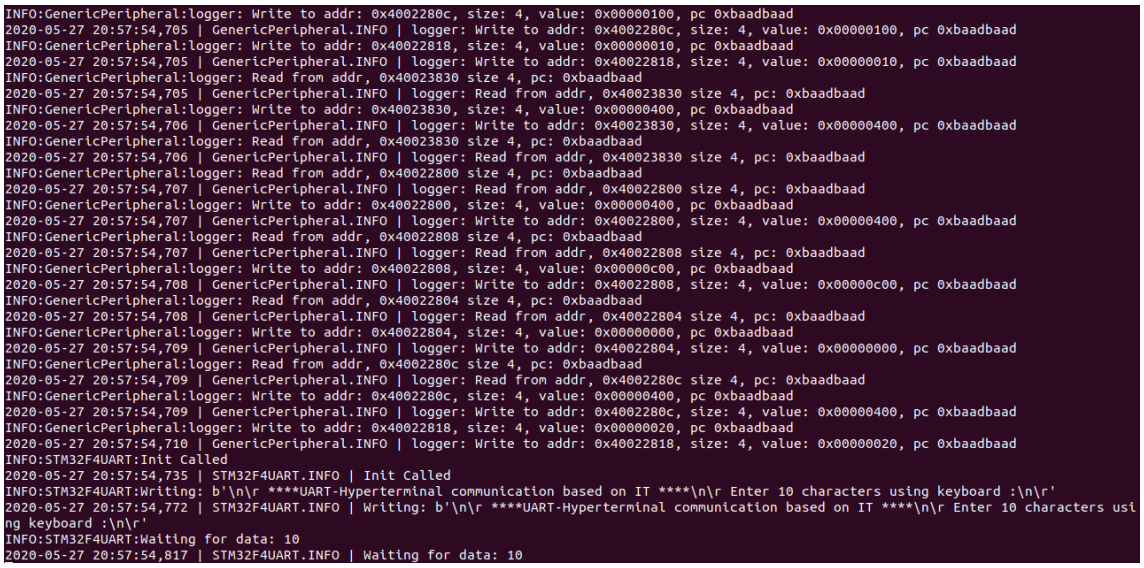


Imagen 22 - Simulación de firmware con protocolo UART en STM32F4

Durante la ejecución de las pruebas se evidencia el uso de QEMU para la emulación de Firmware, así como la apertura de puertos de servicio para la ejecución de procesos e intercomunicación.

En esta simulación presente se evidencia la apertura de los siguientes puertos:

Proceso	Puerto	Protocolo
Qemu-System	1234	TCP
Qemu-System	1235	TCP
Python	5556	TCP

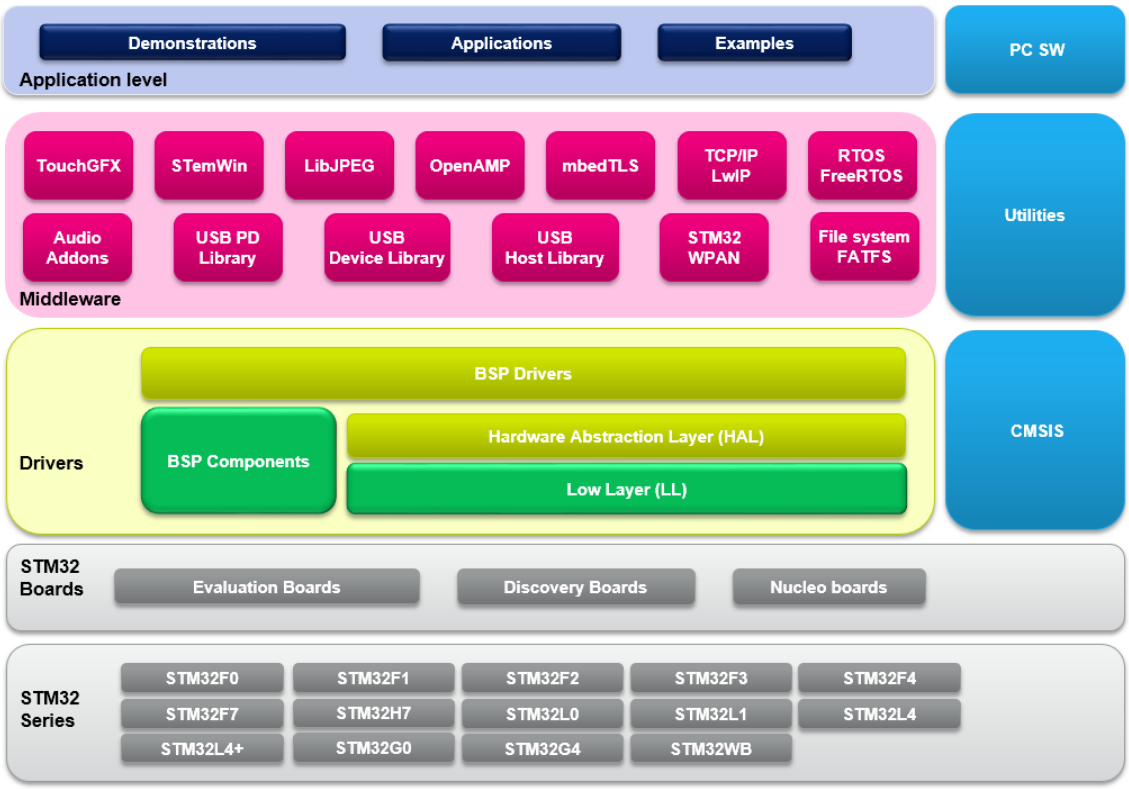
Imagen 23 - Puertos aperturados en la simulación

Para la ejecución de pruebas en STM32, se hizo uso de una serie de firmware llamado STM32Cube, los cuales vienen un paquete y están disponibles por STMicroelectronics, este paquete es llamado STM32CubeF4 MCU Firmware Package.

STM32Cube es una iniciativa original de STMicroelectronics para facilitar la vida de los desarrolladores al reducir los esfuerzos, el tiempo y los costos, este cubre la cartera general de productos STM32 e incluye una plataforma integral de software integrado el cual está siendo usado en este proyecto, que se entrega para cada serie STM32.

- Los módulos CMSIS (núcleo y dispositivo) correspondientes al núcleo ARM-tm implementado en este producto STM32
- Los controladores STM32 HAL-LL: una capa de controladores de abstracción, la API garantiza la máxima portabilidad en todo el portafolio STM32
- Los controladores BSP de cada placa de evaluación o demostración proporcionada por esta serie STM32
- Un conjunto consistente de componentes de middlewares como RTOS, USB, FatFS, LwIP, Gráficos ...
- Un conjunto completo de proyectos de software (ejemplos básicos, aplicaciones o demostraciones) para cada placa proporcionada por esta serie STM32
- Los proyectos del paquete STM32CubeF4 MCU se ejecutan directamente en las placas de la serie STM32F4. Puede encontrar en cada directorio de Proyectos / Nombre de la Junta un conjunto de proyectos de software (Aplicaciones / Demostración / Ejemplos)

En este paquete FW, los módulos Middlewares / ST / TouchGFX, Middlewares / ST / STemWin y Middlewares / ST / STM32_Audio no son accesibles directamente. Deben descargarse de un servidor ST, la URL respectiva está disponible en un archivo readme.txt dentro de cada módulo.



(1) The set of middleware components depends on the STM32 Series.

Imagen 24 - Arquitectura modular de componentes STM32

Los controladores HAL (Capa de abstracción de hardware) que se incluyen en este paquete son compatibles con:

STM32F405 / 415/407/417/427/437/429/439 / 401xC / 401xE / 411xC / 411xE / 412x / 413/423/446/469/479 / 410xx líneas

Usando los archivos fuente de firmware de ejemplos descritos anteriormente se procede a la realizar la compilación de estos para la ejecución a través de ambientes emulados.

```
root@halucinator:/home/osboxes/Downloads/STM32CubeF4-master/Projects/STM32469I_EVAL/Examples/UART/UART_HyperTerminal_IT/SW4STM32/STM32469I_EVAL# python3 /home/o
or/src/tools/stm_tools/build_scripts/CubeMX2Makefile.
CubeMX2Makefile.py   CubeMX2Makefile.tpl
root@halucinator:/home/osboxes/Downloads/STM32CubeF4-master/Projects/STM32469I_EVAL/Examples/UART/UART_HyperTerminal_IT/SW4STM32/STM32469I_EVAL# python3 /home/o
or/src/tools/stm_tools/build_scripts/CubeMX2Makefile.py .
Board: STM32469I_Eval
APP: Uart_IT
../../../../../../Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_rcc.c
../../../../../../Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_gpio.c
../../../../../../Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal.c
../../../../../../Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_i2c.c
../../Src/main.c
../startup_stm32f469xx.s
../../../../../../Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_i2c_ex.c
../../../../../../Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_uart.c
../../readme.txt
../../Src/stm32f4xx_hal_msp.c
../../../../../../Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_dma.c
../../../../../../Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_rcc_ex.c
../../../../../../Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_pwr_ex.c
../../../../../../Drivers/BSP/STM32469I_EVAL/stm32469i_eval.c
../../Src/stm32f4xx_it.c
../../Src/system_stm32f4xx.c
../../../../../../Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_cortex.c
Makefile created: /home/osboxes/Downloads/STM32CubeF4-master/Projects/STM32469I_EVAL/Examples/UART/UART_HyperTerminal_IT/SW4STM32/STM32469I_EVAL/Makefile
root@halucinator:/home/osboxes/Downloads/STM32CubeF4-master/Projects/STM32469I_EVAL/Examples/UART/UART_HyperTerminal_IT/SW4STM32/STM32469I_EVAL# CCIPAGATA
```

Imagen 25 - Generación de archivo MAKEFILE sobre el firmware Uart_IT

Luego de la compilación satisfactòria se obtienen archivos en formato .elf.

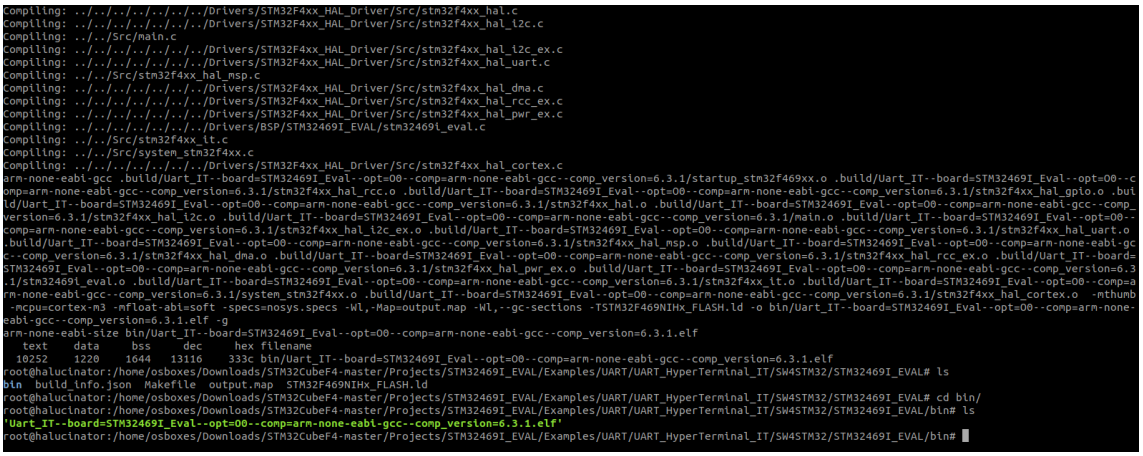


Imagen 26 - Compilación con exito del frmware Uart_IT

Estos archivos formato .elf obtenidos a través de la compilación exitosa son el insumo para poder proceder con la creación de los archivos binarios .bin, los cuales son archivos usualmente instalados e implementados en placas y arquitecturas IoT, para este caso nos será insumo para la creación de:

- Un archivo que contiene el mapa de memoria del Chip a simular en un archivo formato .yaml
- Un archivo que contiene los apuntadores y nombres de funciones usadas
- Un archivo de configuración. El cual define las funciones a interceptar o lo que se quiere manipular.

Estos archivos son los que permiten la emulación de un firmware o sección de firmware como lo ilustrado en la imagen 19.

Se crea el archivo en formato .bin como se ilustra en la imagen a continuación.

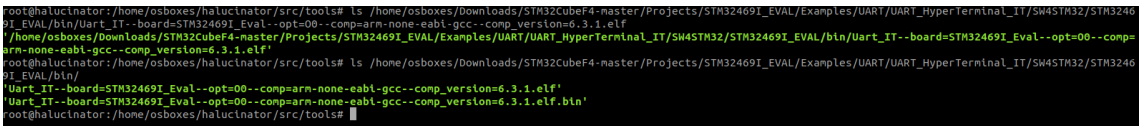


Imagen 27 - Creación de archivo .bin

Este archivo .elf es usado para la creación del archivo “addrss”, el cual fue mencionado previamente y permite a partir de un .elf generar el mapa de memoria del Chip a simular en un archivo formato .yaml.

Así mismo se procedió a realizar la creación de otros archivos .elf con el fin de realizar más simulaciones, se realizar compilaciones requeridas, se usó una herramienta para convertir el Software Workbench de STM con el fin de permitir la compilación de sus proyectos IDE usando make, como se realizó en múltiples ejemplos este script los compila como dispositivos cortex-m3 y no como cortex-m4 para permitir una emulación más fácil en QEMU, en la siguiente imagen se se realizó la constricción de archivos .elf para la simulación de un fiemware llamado UART_Printf, el cual muestra cómo redirigir la función printf de la biblioteca C al UART.
El UART emite un mensaje en HyperTerminal.

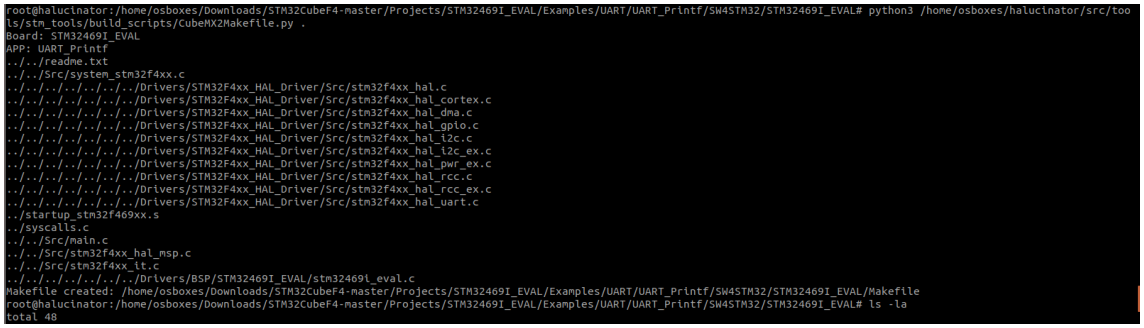


Imagen 28 - Compilación de UART_Printf parte 1

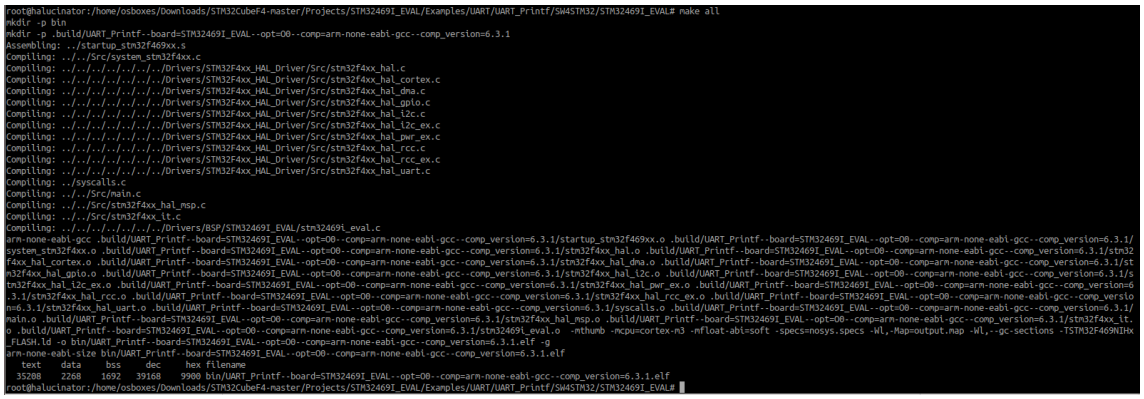


Imagen 29 - Compilación de UART_Printf parte 2

A partir del archivo .elf, se genera los archivos .bin.

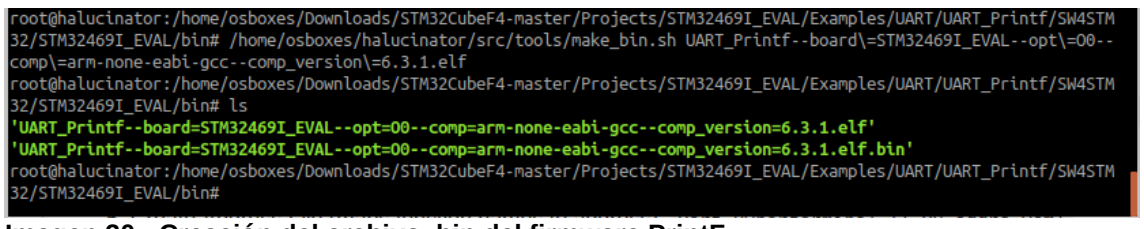


Imagen 30 - Creación del archivo .bin del firmware PrintF

A partir del archivo .elf se pude generar el archivo de “addrs”, el cual permite direccionar las funciones a direcciones de memoria durante la ejecución del firmware.

```
(halucinator) root@halucinator:/home/osboxes/Downloads/STM32CubeF4-master/Projects/STM32469I_EVAL/Examples/UART/UART_Printf/SW4STM32/STM32469I_EVAL/bin# python /home/osboxes/halucinator/src/halucinator/util/elf_sym_hal_getter.py -b UART_Printf--board\=STM32469I_EVAL--opt\=O0--comp\=arm-none-eabi-gcc--comp_version\=6.3.1.elf
('Loading', 'UART_Printf--board=STM32469I_EVAL--opt=O0--comp=arm-none-eabi-gcc--comp_version=6.3.1.elf')
(halucinator) root@halucinator:/home/osboxes/Downloads/STM32CubeF4-master/Projects/STM32469I_EVAL/Examples/UART/UART_Printf/SW4STM32/STM32469I_EVAL/bin#
(halucinator) root@halucinator:/home/osboxes/Downloads/STM32CubeF4-master/Projects/STM32469I_EVAL/Examples/UART/UART_Printf/SW4STM32/STM32469I_EVAL/bin# ls
'UART_Printf--board=STM32469I_EVAL--opt=O0--comp=arm-none-eabi-gcc--comp_version=6.3.1_addrs.yaml'
'UART_Printf--board=STM32469I_EVAL--opt=O0--comp=arm-none-eabi-gcc--comp_version=6.3.1.elf'
'UART_Printf--board=STM32469I_EVAL--opt=O0--comp=arm-none-eabi-gcc--comp_version=6.3.1.elf.bin'
(halucinator) root@halucinator:/home/osboxes/Downloads/STM32CubeF4-master/Projects/STM32469I_EVAL/Examples/UART/UART_Printf/SW4STM32/STM32469I_EVAL/bin#
(halucinator) root@halucinator:/home/osboxes/Downloads/STM32CubeF4-master/Projects/STM32469I_EVAL/Examples/UART/UART_Printf/SW4STM32/STM32469I_EVAL/bin#
```

Imagen 31 - Creación de addrs.yaml en firmware UART_PrintF

Para la creación de estos archivos .elf se requiere la instalación de Angr en en ambiente virtual, este ambiente ya se habia creado de la misma manera duranmte la ejecución de las simulaciones en la herramienta Pretender, ilustrada en la imagen número 15, a continuación se ilñustra la instalación y configuración de angr en ambiente virtual “halucinator”.

```
Requirement already satisfied: angr in /root/.virtualenvs/halucinator/lib/python3.6/site-packages
Requirement already satisfied: capstone>=3.0.5rc2 in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from angr)
Requirement already satisfied: pyvex==8.20.5.27 in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from angr)
Requirement already satisfied: sortedcontainers in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from angr)
Requirement already satisfied: networkx>=2.0 in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from angr)
Requirement already satisfied: psutil in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from angr)
Requirement already satisfied: cle==8.20.5.27 in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from angr)
Requirement already satisfied: archinfo==8.20.5.27 in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from angr)
Requirement already satisfied: pycparser>=2.18 in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from angr)
Requirement already satisfied: mulpyplexer in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from angr)
Requirement already satisfied: protobuf in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from angr)
Requirement already satisfied: progressbar2 in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from angr)
Requirement already satisfied: cachetools in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from angr)
Requirement already satisfied: unicorn>=1.0.2rc2 in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from angr)
Requirement already satisfied: itanium-demangler in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from angr)
Requirement already satisfied: cffi>=1.7.0 in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from angr)
Requirement already satisfied: ailment==8.20.5.27 in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from angr)
Requirement already satisfied: GitPython in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from angr)
Requirement already satisfied: dpkt in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from angr)
Requirement already satisfied: rpyc in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from angr)
Requirement already satisfied: claripy==8.20.5.27 in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from angr)
Requirement already satisfied: bitstring in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from pyvex==8.20.5.27->angr)
Requirement already satisfied: future in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from pyvex==8.20.5.27->angr)
Requirement already satisfied: decorator>=4.3.0 in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from networkx>=2.0->angr)
Requirement already satisfied: pefile in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from cle==8.20.5.27->angr)
Requirement already satisfied: pyelftools>=0.25 in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from cle==8.20.5.27->angr)
Requirement already satisfied: setuptools in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from protobuf->angr)
Requirement already satisfied: six>=1.9 in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from protobuf->angr)
Requirement already satisfied: python-utils>=2.3.0 in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from progressbar2->angr)
Requirement already satisfied: gitdb<5,>=4.0.1 in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from GitPython->angr)
Requirement already satisfied: plumbum in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from rpyc->angr)
Requirement already satisfied: pysmt in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from claripy==8.20.5.27->angr)
Requirement already satisfied: z3-solver>=4.8.5.0 in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from claripy==8.20.5.27->angr)
Requirement already satisfied: smmap<4,>=3.0.1 in /root/.virtualenvs/halucinator/lib/python3.6/site-packages (from gitdb<5,>=4.0.1->GitPython->angr)
```

Imagen 32 - Instalación de Angr en ambiente virtual Halucinator

Para la creación de archivo addrss se requiere del script, elf_sym_hal_getter.py, el cual se ejecuta con Python haciendo uso del archivo elf así.

python -m halucinator.util.elf_sym_hal_getter -b “archivo .elf Fuente”

Este archivo contiene apuntadores de direcciones a nombres de funciones de la siguiente forma ilustrados a continuación:

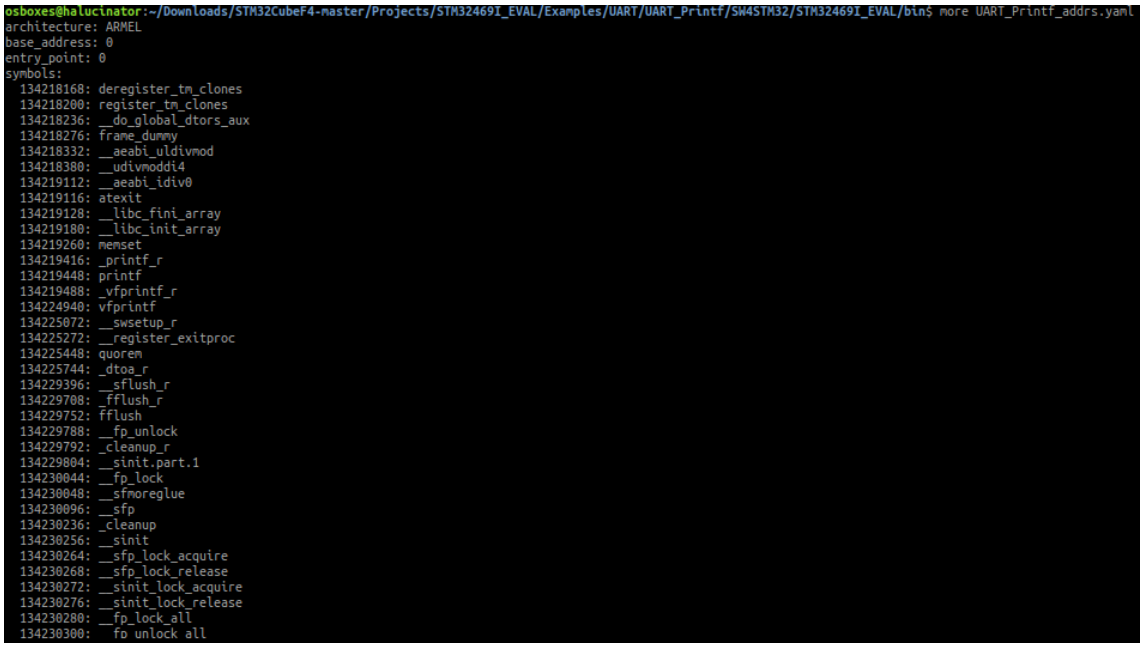


Imagen 33 - Espacios de memoria apuntados a nombres de funciones

Luego de contar con los insumos requeridos para realizar la simulación de este firmare de prueba se ejecuta, ilustrado en la siguiente imagen.

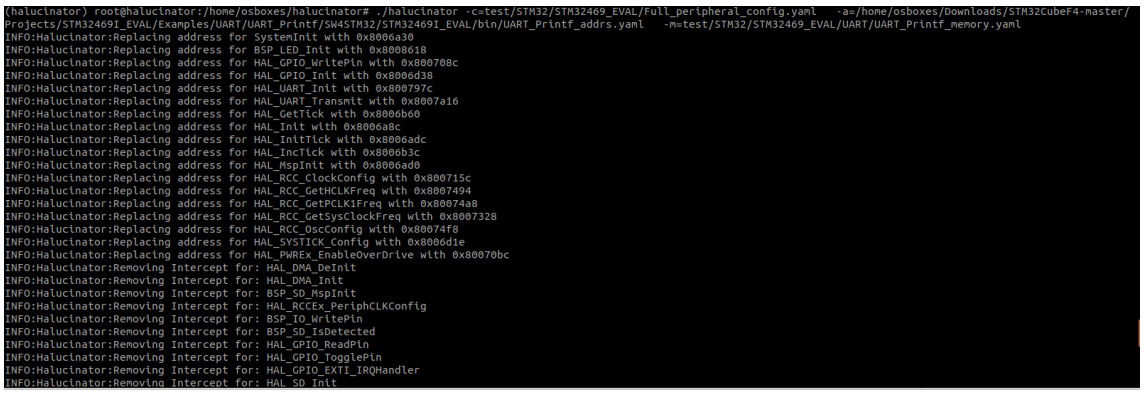


Imagen 34 - Ejecución de la simulación de Firmware PrimtF

A través de las múltiples simulaciones que se han realizado, se ha podido emular múltiples Componentes y módulos sencillos que hacen parte de un dispositivo IoT, esto permite así determinar que con una combinación de herramientas se puede llegar a efectuar un ambiente simulado de IoT sin necesidad de contar con un hardware.

Algunas de las herramientas usadas en esta investigación y sus funciones fueron:

Tabla 4 - Firmwares simulados

Herramienta	Función	Simulaciones
QEMU	Qemu es un emulador y virtualizador genérico, este es usado por otras herramientas.	Todas
HALucinator	Permite la simulación de firmware a través de capas de abstracción de hardware (HAL), haciendo uso de QEMU, Avatar, entre otros	• UART_PrintF • UART_Hyperterminal
P retender	Utiliza observaciones de las interacciones entre el hardware original y el firmware para crear automáticamente modelos de periféricos, y permite la ejecución del firmware en un entorno totalmente emulado.	• blink_led • read_hyperterminal • button_interrupt • thermostat • rf_door_lock
Fie	FiE es un motor de ejecución simbólico basado en KLEE, específicamente dirigido a encontrar vulnerabilidades en el firmware de los microcontroladores MSP430.	
Avatar y Avatar2	Avatar es un marco de orquestación de destino con enfoque en el análisis dinámico del firmware de los dispositivos integrados, diseñado para permitir la interoperabilidad entre diferentes marcos de análisis binarios dinámicos, depuradores, emuladores y dispositivos físicos reales.	Todas

4. Conclusiones y líneas de trabajo futuro

A través de las simulaciones y la implementación de escenarios satisfactorios de simulación, se logra llevar a cabo la investigación de como los sistemas IoT pueden ser ejecutados y emulados fuera de su contexto o arquitectura de hardware para los que fueron diseñados cumpliendo con uno de los objetivos, en la tabla numero 4, se listaron los 7 firmware que pudieron ser ejecutados con resultados satisfactorios así como el uso y configuración de herramientas de código abierto que permitieron desplegar un sistema de bajo costo y de desempeño esperado.

En cuanto a los objetivos que se habían planteado al inicio del trabajo de investigación, pienso que estos han sido cumplidos, esto gracias al desarrollo de una gran fase de investigación que nos permitió entender que es, como funciona, el contexto y sus principales problemas y necesidad de los sistemas IoT en el mundo, para de esta forma "e*ntender el comportamiento de las redes y atacantes de internet hacia los dispositivos IOT más comunes del mercado*."

A partir de estos insumos fue posible comprender cuales son sus principales debilidades, cuales son los protocolos comúnmente implementados y de estos cuales presentan mayores vulnerabilidades, para luego abordar las necesidades de el rehosting de firmware IoT , cumpliendo con el objetivo de "*simular uno o varios sistemas IOT más comunes a través de múltiples herramientas de simulación*", por medio de la investigación "*a través de sistemas de código abierto se pueda adaptar múltiples firmwares de internet de las cosas*".

Como trabajo futuro seria optimo abordar más simulaciones de firmwares o partes de firmwares más completos, esto requiere un tiempo de investigación más extenso por lo que se propondría un calendario de aproximadamente de 6 meses más donde para la emulación de más firmware se sugiere la interacción de las herramientas de simulación, así como el diseño de una nueva herramienta a partir de Python que se especialice en simulaciones ARM para sistemas IoT.

Así mismo como trabajo futuro sería optimo contar con un ambiente de análisis de código estático y dinámico en ambientes reales, donde sea posible analizar en análisis dinámico el funcionamiento en tiempo real de los dispositivos IoT, con el fin de identificar

patrones o acciones inseguras que puedan desencadenar un riesgo de seguridad; con el análisis estático se podría realizar una investigación sobre el código fuente de los firmware de dispositivos IoT que permitan identificar vulnerabilidades en componentes del código como librerías desactualizadas, librerías inseguras, código inseguro y vulnerabilidades conocidas.

5. Bibliografía

Arm Limited. (2020). *ST-Nucleo-F072RB*. (NUCLEO-F072RB) Recuperado el 24 de 04 de 2020, de https://os.mbed.com/platforms/ST-Nucleo-F072RB/

Arm Limited. (2020). *THE FOUNDATION FOR SECURE IOT*. Recuperado el 25 de 04 de 2020, de https://www.arm.com/solutions/iot

abomhara, M. (2015). Cyber Security and the Internet of Things: Vulnerabilities, Threats, Intruders and Attacks. *Journal of Cyber Security and Mobility*, 10.

B.V, E. (2020). *What is the ELK Stack?* Obtenido de https://www.elastic.co/what-is/elk-stack

Bertino, E. (2010). Web services threats, vulnerabilities, and countermeasures. *Springer*, 25-44.

Brian Caswell, J. B. (s.f.). Snort 2.1 Intrusion Detection, Second Edition. En B. Caswell, *Snort 2.1 Intrusion Detection, Second Edition* (pág. 419).

Cisco Systems. (2020). *What is Snort?* Obtenido de https://www.snort.org/faq/what-is-snort

Clayton, A. V. (2019). Honware: A Virtual Honeypot Framework for. *Computer Laboratory, University of Cambridge*.

DIZDAREVIĆ, J. (2019). A Survey of Communication Protocols for Internet of Things. *ACM Computing Surveys*, 6.

DOUG HELLMANN. (2020). *virtualenvwrapper Documentation*. Recuperado el 15 de 04 de 2020, de https://virtualenvwrapper.readthedocs.io/en/latest/install.html

ETSI. (2013). Machine-to-Machine communications (M2M). *ETSI TS 102 689*, https://www.etsi.org/deliver/etsi_ts/102600_102699/102689/01.02.01_60/ts_102689v010201p.pdf.

farnell.com. (11 de 2016). *Datasheet STM32 Nucleo-64 board NUCLEO-XXXXRX.* Recuperado el 15 de 05 de 2020, de http://www.farnell.com/datasheets/2308719.pdf?_ga=2.148837685.321637528.1590372428-969571180.1590372428

flylib.com. (2017). *Analizyng and graphing Logs*. Obtenido de https://flylib.com/books/en/2.12.1/analyzing_and_graphing_logs.html

García, P. B. (2019). IoT Botnets. *WITCOM 2019: Telematics and Computing*, 247-257.

IEEE. (06). *The Constrained Application Protocol (CoAP).* Recuperado el 15 de 03 de 2020, de 2014: https://tools.ietf.org/html/rfc7252

IEEE. (02 de 2000). *A Model for Presence and Instant Messaging.* Recuperado el 15 de 03 de 2020, de tools.ietf.org: https://tools.ietf.org/html/rfc2778

IEEE. (07 de 2013). *Universal Plug and Play (UPnP).* Recuperado el 15 de 03 de 2020, de https://tools.ietf.org/html/rfc6970

IEEE. (2015). Towards a definition of the Internet of things. *IEEE Internet Initiative*.

ietf. (2010). The Internet of Things - Concept and Problem Statement. *Internet Research Task Force*.

ISO/IEC. (06 de 2016). *Information technology — Message Queuing Telemetry Transport (MQTT).* Recuperado el 15 de 03 de 2020, de ISO/IEC

20922:2016 [ISO/IEC 20922:2016]: https://www.iso.org/standard/69466.html
Kaspersky. (2018). *What is Zero Day Exploit?* Recuperado el 03 de 2020, de https://www.kaspersky.com/resource-center/definitions/zero-day-exploit
Kizza, J. M. (2013). *Guide to Computer Network Security.* Springer.
Kolias, C. (2017). DDoS in the IoT: Mirai and Other Botnets. *IEEE Computer Society*, 4.
Kramer, S. (2009). A general definition of malware. *Journal in Computer Virology*, 2.
Krishnaprasad, P. (2017). Capturing attacks on IoT devices.
M. De Vivo, E. C. (1999). A review of port scanning techniques. *ACM SIGCOMM Computer Communication Review*, 41.
Maxim Integrated. (10 de 2015). *MAX32600MBED ARM mbed Enabled.* Recuperado el 16 de 05 de 2020, de https://datasheets.maximintegrated.com/en/ds/MAX32600MBED.pdf
Mbed. (2017). *MAX32600 Evaluation Board.* Recuperado el 16 de 05 de 2020, de https://os.mbed.com/platforms/MAX32600mbed/
OASIS. (10 de 2012). *OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0.* Recuperado el 15 de 03 de 2020, de OASIS Standard: http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-messaging-v1.0-os.html
OMG. (04 de 2015). *Data Distribution Service (DDS).* Recuperado el 15 de 03 de 2020, de https://www.omg.org/spec/DDS/1.4/PDF
Phype. (2016). *Telnet IoT honeypot*. Obtenido de https://github.com/Phype/telnet-iot-honeypot
Pipkin, D. L. (2000). *Information security.* Prentice Hall PTR.
Schneier, B. (2011). *Secrets and lies: digital security in a networked world.* John Wiley & Sons.
Science Direct. (2020). *reduced instruction set computer.* Recuperado el 2020, de https://www.sciencedirect.com/topics/computer-science/reduced-instruction-set-computer
Seaman, C. (2018). *UPNPROXY: ETERNALSILENCE.* Recuperado el 17 de 03 de 2020, de https://blogs.akamai.com/sitr/2018/11/upnproxy-
Spitzner, L. (05 de 2002). *Honeypot: Definitions and Values*. Recuperado el 17 de 03 de 2020, de http://www.spitzner.net
Suzuki, S. (2016). IoTPOT: A Novel Honeypot for Revealing CurrentIoT Threats. *Journal of Information Processin*, 522-233.
TheQEMUProjectDevelopers. (2020). *wiki qemu*. Obtenido de https://wiki.qemu.org/Main_Page
Universidad de Michigan. (2012). Lecture 3: Toolchains. Obtenido de https://web.eecs.umich.edu/~prabal/teaching/resources/eecs373/toolchain-notes.pdf
VMware, Inc. (2020). *workstation Pro*. Recuperado el 04 de 2020, de https://www.vmware.com/co/products/workstation-pro/workstation-pro-evaluation.html
William, R. (2017). Identifying Vulnerabilities of Consumer Internet of. *Management Information Systems* , 180.
Wilson, C. (2008). Botnets, cybercrime, and cyberterrorism: Vulnerabilities and policy issues for congress. *DTIC Document*.

Zhang, F. (2003). Honeypot: a Supplemented Active Defense System for Network Security. *College of Computer Science and Engineering*, 231-232.