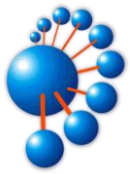
UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA
Y CIENCIAS DE LA COMPUTACIÓN



Plataforma de monitoreo de indicadores de compromiso con comunicación encriptada a sistemas externos

POR
Felipe Nicolás Abdón Henríquez Ricart

Memoria de Título presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al título profesional de Ingeniero Civil Informático

Profesor Patrocinante
Pedro Pablo Pinacho Davidson

Profesor Co-patrocinante
Sergio Kendrick Sobarzo Guzmán

Profesores Comisión
Gonzalo Eduardo Rojas Durán
José Sebastián Fuentes Sepúlveda

Concepción, 30 de junio de 2023

Agradecimientos

Aprovecho este espacio para agradecer a todas las personas que han formado parte de mi vida durante los últimos 6 años en la universidad, y de igual manera, a aquellas que siempre han estado a mi lado. Han sido años llenos de desafíos y sin duda alguna, no ha sido un camino fácil.

Comienzo agradeciendo a mi compañero y amigo Sergio Cifuentes, con quién desarrollé la mayoría de este proyecto, haciéndolo más llevadero y siendo un pilar fundamental para lograr lo conseguido.

Agradezco al profesor co-patrocinante Sergio Sobarzo por los conocimientos y ayuda que aportó durante este trabajo. Asimismo, extiendo mi gratitud al profesor Pedro Pinacho, con quien he trabajado y aprendido mucho los últimos años, y que también fue el patrocinante de esta memoria, en la cual contribuyó enormemente con sus ideas, consejo y apoyo constante.

Debo agradecer a todos los profesores y profesoras que han sido parte de mi proceso universitario. En especial, a los docentes del Departamento de Ingeniería Civil Informática y Ciencias de la Computación por su dedicación y enseñanzas. Han sido esenciales tanto en la realización de esta memoria como en mi formación como profesional.

Quiero expresar mi más profundo agradecimiento a mis padres, Denisse Ricart Mendoza y Abdón Henríquez Campillay. Su infaltable compañía, motivación y apoyo incondicional ha sido lo más valioso que he tenido durante cada etapa de mi vida. Gracias por estar siempre a mi lado y por creer en mí, los amo con todo mi corazón. De la misma forma, agradezco a mi familia por todo lo que han hecho por mí, principalmente a mi hermano Alonso y a mi padrino Gonzalo, gracias por estar presentes ayudándome cada vez que lo necesitaba.

Gracias a mis compañeros de carrera por su apoyo y trabajo en equipo durante mi formación en la universidad. Agradezco a Ivonne, Matías, Patricio y Leonardo por ser parte importante de mi vida y por la bonita amistad que hemos construido. También gracias a Joaquín y Pablo por formar parte de este viaje académico junto a mí.

Por último, doy mi sincero agradecimiento a mis amigos Aylen, Daniel, Eduardo, Jorge, Nazlo, Oscar, Ximena y a todos los demás que han estado a mi lado brindándome compañía, consejo y motivación en los momentos más difíciles. Esto ha significado mucho para mí y ha sido fundamental para superar los desafíos que he enfrentado. Quiero que sepan que su amistad es algo muy valioso que aprecio profundamente.

Resumen

En los últimos años se ha observado un aumento significativo de los ciberataques, afectando principalmente a las organizaciones y empresas. Teniendo esto en cuenta es importante que estas tomen medidas para protegerse de estos ataques, y un elemento importante para lograrlo son los indicadores de compromiso, capaces de evidenciar que ocurrió o está ocurriendo un ataque dentro de un sistema, explicando lo que sucedió y permitiendo tomar medidas y/o prevenir mayores daños o perdidas.

En esta memoria de título se implementó y testeó exitosamente un primer prototipo de una plataforma de monitoreo de indicadores de compromiso, donde un servidor monitorea constantemente, dentro de una red local, a *hosts* con sistema operativo Linux, comunicándose con ellos mediante un canal de comunicación seguro con gRPC, un sistema de llamada a procedimientos remotos.

Este prototipo es capaz de detectar indicadores de compromiso, solicitar reportes con información actual del sistema a los hosts en la red local, como también entregar lo recolectado a una aplicación de terceros.

Índice

Lista de Tablas

Lista de Figuras

Glosario

Agentes Evolutivos: Población de agentes que utilizan principios basados en la evolución biológica para adaptarse al medio en que se desarrollan. Partiendo con una población inicial que, a medida que pasan las generaciones, se adaptan a los cambios en su medio.

Artefactos de red: Elementos que se generan durante la comunicación a través de una red y que pueden ser utilizados para identificar o analizar esa comunicación. Estos artefactos incluyen información sobre los protocolos utilizados, los puertos utilizados, las direcciones IP de origen y destino, los patrones de tráfico de red, entre otros.

Ataque del día Cero: Un ataque de día cero es un ataque contra una aplicación o sistema que tiene como objetivo la ejecución de código malicioso gracias al conocimiento de vulnerabilidades que son desconocidas para los usuarios y para el fabricante del producto que no hayan sido arregladas.

C2: Un ataque C2 o *Command and Control*, es un tipo de ataque en el que un ciberdelincuente se infiltra en un sistema informático y toma el control a través de un malware, esto se puede lograr mediante vulnerabilidades de software o ingeniería social, entre otras. Esto permite al ciberdelincuente realizar acciones maliciosas como la filtración de datos, eliminación de archivos, espionaje de la actividad del usuario, lanzamiento de ataques DDoS, e instalación de más malware en el sistema.

Cifrado Asimétrico: El cifrado asimétrico es un método de cifrado que utiliza un par de claves: una clave pública y una clave privada. A diferencia del cifrado simétrico, donde se utiliza la misma clave para cifrar y descifrar la información. En cuanto al procedimiento, un usuario genera ambas claves y comunica la clave pública al otro usuario. Luego los mensajes podrán ser cifrados con la clave pública y solo ser descifrados con la clave privada de cada uno.

Cifrado Simétrico: El cifrado simétrico es un método de cifrado en el que se utiliza la misma clave para cifrar y descifrar los datos. En este tipo de cifrado, tanto el remitente como el destinatario deben conocer y utilizar la misma clave secreta para garantizar la confidencialidad de la información. Es rápido y eficiente, pero requiere una distribución segura de la clave compartida entre el remitente y el destinatario.

DDoS: Los ataques de red distribuidos a menudo se conocen como ataques de denegación distribuida de servicio (DDoS). Este tipo de ataque aprovecha los límites de capacidad específicos que se aplican a cualquier recurso de red, tal como la infraestructura que habilita el sitio web de la empresa. El ataque DDoS, desde múltiples orígenes y al mismo tiempo, envía solicitudes al recurso web atacado, con la intención de desbordar la capacidad del sitio web para administrar varias solicitudes y de evitar que este funcione correctamente.

gRPC: gRPC es un sistema de llamada a procedimiento remoto de código abierto desarrollado inicialmente por Google. Utiliza como transporte HTTP/2 y Protocol Buffers como lenguaje de descripción de interfaz.

Hash o Hashing: Algoritmo matemático que transforma cualquier bloque arbitrario de datos en una nueva serie de caracteres con una longitud fija.

HTTP/2: Nueva versión del protocolo que utiliza internet para la transferencia de datos, HTTP (Hypertext Transfer Protocol) es el lenguaje o código a través del cual se comunican los navegadores, servidores y otros dispositivos de red. Con respecto a las mejoras que ofrece tenemos que es binario, permite multiplexación (permite enviar y recibir varios mensajes a la vez), entre otras.

ICMP: El protocolo de mensajes de control de Internet es parte del conjunto de protocolos IP. Es utilizado para enviar mensajes de error e información operativa indicando, por ejemplo, que un host no puede ser localizado o que un servicio que se ha solicitado no está disponible.

IDS: Un sistema de detección de intrusiones (IDS) es un programa de detección de accesos no autorizados a un computador o a una red. El IDS suele tener sensores virtuales con los que el núcleo del IDS puede obtener datos externos.

Ingeniería Social: La ingeniería social es una técnica utilizada por los ciberdelincuentes para engañar a las personas y obtener información confidencial o acceso no autorizado a sistemas o redes informáticas mediante la manipulación psicológica y emocional.

IoC: Un indicador de compromiso o IDC, es toda aquella información relevante que describe cualquier incidente de ciberseguridad, actividad y/o artefacto malicioso, mediante el análisis de sus patrones de comportamiento.

Man in the Middle (MITM): En un ataque *Man in the Middle*, el atacante intercepta la comunicación entre dos dispositivos para obtener información confidencial o para modificar la comunicación entre ellos.

MD5: En criptografía, MD5 es un algoritmo de reducción criptográfico de 128 bits ampliamente usado. Uno de sus usos es el de comprobar que algún archivo no haya sido modificado.

Open Source: Código de uso libre, es un modelo de desarrollo de software basado en la colaboración abierta.

Pass-The-Hash: Es una técnica donde un atacante captura el hash de una contraseña, y logra autenticarse teniendo acceso a un sistema.

Protobuf o Protocol Buffers: Protocol Buffers (abreviado como Protobuf) es un formato de intercambio de datos desarrollado originalmente para uso interno en Google y lanzado al público por la empresa en 2008 como proyecto de código abierto (en parte, con licencia Apache 2.0). Este formato binario permite a las aplicaciones almacenar e intercambiar datos estructurados fácilmente, incluso si los programas están compilados en diferentes lenguajes. Entre los lenguajes de programación compatibles se incluyen:

- C#
- C++
- Go
- Objective-C
- Java
- Python
- Ruby

Puerta trasera (Backdoor): Tipo de troyano que permite el acceso al sistema infectado y su control remoto. El atacante puede entonces eliminar o modificar archivos, ejecutar programas, enviar correos masivamente o instalar herramientas maliciosas.

Raspberry Pi: La Raspberry Pi es una serie de ordenadores monoplaca u ordenadores de placa simple de bajo costo desarrollado en el Reino Unido por la Raspberry Pi Foundation, con el objetivo de poner en manos de las personas de todo el mundo el poder de la informática y la creación digital.

Rootkit: Paquete de software con la finalidad de conseguir un acceso privilegiado y nivel administrador de un equipo, permitiendo realizar las mismas acciones que un administrador, y así poder ocultar malware, obtener acceso remoto, desactivar programas de seguridad, crear puertas traseras, entre otros.

SHA1: En criptografía, SHA-1 es una función hash que toma una entrada y produce un valor hash de 160 bits conocido como resumen de mensaje, normalmente representado como 40 dígitos hexadecimales.

SMNP: El protocolo simple de administración de red o SNMP es un protocolo de la capa de aplicación que facilita el intercambio de información de administración entre dispositivos de red.

SSH: SSH (*Secure Shell*) es el nombre de un protocolo y del programa que lo implementa cuya principal función es el acceso remoto a un servidor por medio de un canal seguro en el que toda la información está cifrada.

SSL/TLS: Seguridad de la capa de transporte y su antecesor Secure Sockets Layer son protocolos criptográficos, que proporcionan comunicaciones seguras por una red, comúnmente Internet.

TCP: El protocolo de control de transmisión (TCP) es, al igual que el protocolo UDP como el SCTP, un protocolo de Internet que está ubicado en la capa de transporte del modelo OSI. Es importante mencionar de este protocolo que, cuando un remitente envía datos, existe una confirmación de parte de receptor, si este no confirma la recepción, se vuelve a transmitir, entregando fiabilidad, ya que garantiza que los datos llegan incluso cuando hay perdida de paquetes de datos en la red.

1. Introducción

En los últimos años, el aumento de los ataques cibernéticos se ha convertido en una preocupación cada vez mayor para individuos, empresas y gobiernos (*figura 1*). Con la creciente dependencia de la tecnología en la vida diaria y el aumento de la cantidad de datos personales y financieros almacenados en línea, los ciberdelincuentes tienen más incentivos y oportunidades para llevar a cabo ataques cibernéticos.

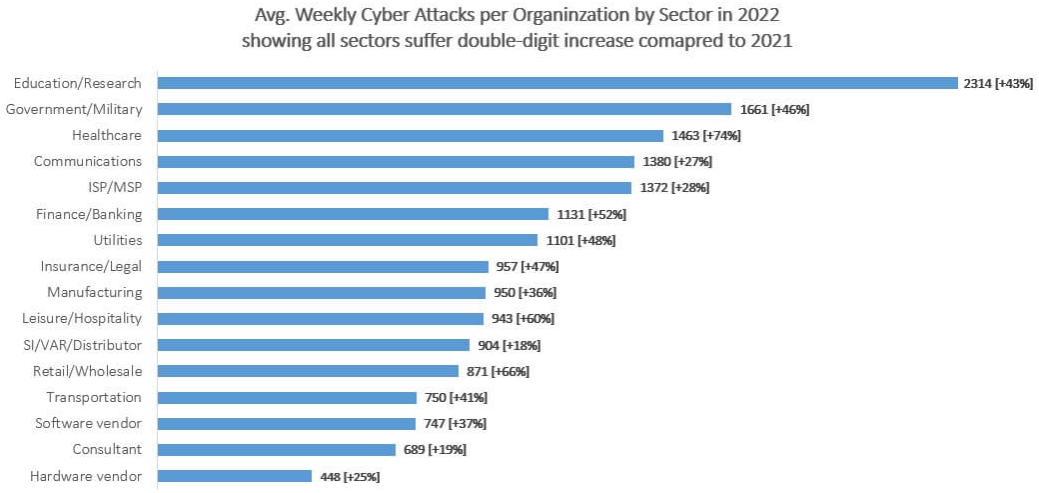


Figura 1: Aumento de ataques durante el 2022 por tipo de organización [1]

Los ataques cibernéticos pueden tomar muchas formas, incluyendo el *phishing*, el *malware*, el *ransomware*, los ataques de denegación de servicio (DDoS), entre otros. Estos ataques pueden tener consecuencias graves, desde la pérdida de datos y la interrupción del servicio, hasta el robo de información personal y financiera, la extorsión y la violación de la privacidad. Sin embargo, mientras se llevan a cabo estos ataques cibernéticos, dejan información o evidencias, que al recopilarlos pueden ayudar a detectarlos en el futuro, pudiendo responder de manera más rápida a las amenazas y minimizar el daño potencial que causan. A esta evidencia o información que dejan los ataques cibernéticos se les llama indicadores de compromiso (IoC) [2], y éstos son herramientas fundamentales en el ámbito de la ciberseguridad. Los indicadores de compromiso pueden incluir direcciones IP, nombres de dominio, *hash* de archivos, cadenas de texto únicas, patrones de comportamiento y otros detalles específicos que son utilizados por los investigadores de seguridad y los sistemas de defensa para detectar y prevenir ataques informáticos.

NIST (National Institute of Standards and Technology), recomienda, para cualquier tipo de organización, utilizar los indicadores de compromiso como una medida para detectar y responder rápidamente a los ciberataques [3]. Por otro lado, según INCIBE (Instituto Nacional de Ciberseguridad Español), las empresas saben reconocer un indicador de compromiso, pero no como utilizarlos y el gran potencial que tienen estos para la prevención, detección y resolución de incidentes [4]. En

otras palabras, se da a entender que los indicadores de compromiso son herramientas muy valiosas para todo tipo de organización, pero de poco conocimiento y uso de parte de éstas.

Teniendo todo lo anterior en cuenta, este proyecto se genera a partir desde la necesidad de la Armada de Chile de una plataforma de monitoreo del estado de la seguridad de su plataforma TI. Para ello, se creó una plataforma de indicadores de compromiso, y aunque inicialmente se pensó para una organización militar, teniendo en cuenta los altos estándares de confidencialidad, integridad y disponibilidad que se necesitan, está dirigida a cualquier tipo de entidad, ya que como se mencionó anteriormente, los indicadores de compromiso resultan beneficiosos para todas.

Por otro lado, R-ABS o sistema de bioindicadores artificiales reactivos, es un IDS que se encuentra en desarrollo y está a cargo del profesor Pedro Pinacho Davidson en la Universidad de Concepción (Proyecto Fondecyt nro. 11230359). Este sistema monitorea una red, y gracias a sus agentes evolutivos, es capaz de detectar el tráfico anómalo caracterizándolo como un ataque, para que así, cuando vuelva a suceder, sea fácilmente detectable y alertado, pudiendo tomar medidas en su contra. Si bien, este modelo es capaz de detectar ataques incluso del día cero, no puede saber si existe compromiso dentro de los equipos en la red que está monitoreando, perdiendo fiabilidad en estos casos.

Asimismo, la plataforma creada es capaz de entregar información sobre el compromiso que ha sido detectado dentro de la red, esto para sistemas de terceros, dando la posibilidad a R-ABS de continuar su desarrollo.

Por último, y explicando la inexistencia de referencias a trabajos similares, es debido a que la Armada de Chile solicitaba una plataforma de desarrollo propio, y, además, que la mayoría de las soluciones relacionadas con esto son de pago, y no entregan información sobre cómo se implementaron.

1.1 Objetivos

1.1.1 Objetivo General

Diseño e implementación sobre sistema Linux de una plataforma segura para el monitoreo de indicadores de compromiso (IoC). Se implementará de igual forma un sistema que enviará la información recolectada a sistemas externos.

1.1.2 Objetivos Específicos

Los objetivos específicos son los siguientes:

- Levantar requerimientos necesarios para la plataforma de monitoreo de IoC.
- Diseñar la arquitectura del sistema, teniendo en cuenta los altos estándares de seguridad que esta necesita.
- Implementar un primer prototipo de la plataforma de monitoreo de IoC, que al igual que su arquitectura, debe cumplir altos estándares de seguridad.
- Crear una interfaz para la comunicación encriptada de IoC a sistemas externos.

1.1.3 Limitaciones

Este proyecto contempla el diseño de la arquitectura de la plataforma y un primer prototipo de lo expuesto anteriormente.

1.1.4 Metodología de trabajo

Este proyecto se realizó con la colaboración de los alumnos memoristas Felipe Henriquez Ricart y Sergio Cifuentes Torres, y bajo la supervisión de los profesores Pedro Pinacho Davidson y Sergio Sobarzo Guzmán.

Se dividió en 3 distintas etapas, de acuerdo con los objetivos específicos, que se detallan a continuación:

1. **Investigación**
   Para poder levantar los requerimientos necesarios para la plataforma de monitoreo de IoC era preciso investigar los mismos indicadores de compromiso a profundidad, soluciones similares existentes y las herramientas que se utilizarían sobre estos.

2. **Diseño**
   En esta etapa se diseñó la arquitectura general de la plataforma, bajo la supervisión de los profesores Pedro Pinacho y Sergio Sobarzo y de acuerdo con lo investigado anteriormente.

3. Implementación

Según lo establecido en la arquitectura general, se definió un prototipo como prueba de concepto. Cabe destacar que durante el desarrollo del prototipo se realizaron modificaciones del diseño.

2. Marco Teórico

2.1 La seguridad de la información dentro de organizaciones

La seguridad de la información es el conjunto de medidas preventivas y reactivas que permiten resguardar y proteger la información. Para lograr esto, es necesario conocer a la tríada CIA [5], un acrónimo que engloba los 3 términos fundamentales dentro de la seguridad de la información:

1. **Confidencialidad**
   Se basa en que controlar el acceso a la información confidencial, de tal manera de que solo los usuarios autorizados puedan acceder a ella. Esta se puede quebrar a través de ataques directos para obtener acceso no autorizado, y también por errores humanos, descuidos o controles de seguridad insuficientes.
   La confidencialidad se asegura mediante fuertes controles de acceso y autenticación, encriptación de datos, educación y capacitación adecuadas para todas las personas con acceso a los datos, entre otras.

2. **Integridad**
   La integridad de la información conlleva mantener la consistencia, precisión y confiabilidad de esta, durante todo su ciclo de vida. Debe asegurarse que las personas que no están autorizadas no puedan alterar los datos. Dentro de la integridad están también los permisos de archivos y controles de acceso de usuarios.

   La integridad de la información se puede lograr con diversas medidas como el *hashing* de los datos asegurando que no han sido modificados, control de versiones, firmado digital, copias de seguridad, actualización de software, auditorias periódicas para así identificar vulnerabilidades o políticas claras y concisas para el uso y manejo de la información.

3. **Disponibilidad**
   La disponibilidad se refiere a que los sistemas y servicios deben estar disponibles y funcionando correctamente cuando un usuario necesite utilizarlos, sin fallos o interrupciones que le imposibiliten el acceso.
   Para garantizar la disponibilidad se pueden seguir algunas medidas, tales como las siguientes, diseño de sistemas y servicios tolerantes a fallos, detección de problemas previniendo fallos críticos, definir planes de contingencia y recuperación en caso de interrupciones inevitables.

Dentro de toda organización, la tríada CIA y cada uno de los elementos que la componen es fundamental y crítica, aunque, en algunas organizaciones, la importancia de cada uno de los elementos puede variar. Por ejemplo, dentro de un banco, la integridad es el elemento que toma más importancia, ya que una manipulación no autorizada de los datos financieros puede ser catastrófico,

teniendo grandes consecuencias legales y económicas, esto no significa que los demás elementos no sean relevantes, ya que la confidencialidad y disponibilidad son criticas también, ya que proteger la información financiera de los clientes y disponer de los sistemas y servicios que procesan transacciones es vital.

Ahora si nos centramos en las organizaciones gubernamentales y militares, el elemento que toma más importancia es la confidencialidad, porque la filtración o divulgación de información clasificada puede comprometer la seguridad nacional. Esto sin dejar de lado el resto de los elementos, ya que sigue siendo vital asegurar la integridad y disponibilidad, confirmando que los sistemas y servicios estén protegidos contra fallos y cambios no autorizados.

Este proyecto, ya en su concepción estaba destinado a organizaciones gubernamentales y militares, la confidencialidad debería ser elemento más relevante, sin embargo, debido a que es una plataforma de monitoreo, es crítico contar con una alta disponibilidad, así dentro de la arquitectura general, es un requisito garantizar los 3 elementos de la tríada CIA.

2.2 Indicadores de Compromiso

Los indicadores de compromiso (IoC, por sus siglas en inglés) son características o patrones con los cuales se puede afirmar que existe presencia de actividad maliciosa en un sistema o red. Este término se ocupó por primera vez en el año 2005 por el Departamento de Defensa de Estados Unidos, con una guía sobre como detectar y responder a los ataques cibernéticos, y desde ese año se han utilizado ampliamente en el ámbito de la ciberseguridad.

Los indicadores de compromiso se pueden describir como direcciones IP, cadenas de texto, *hash* de archivos, y entre otras características que indican directamente que existe, en mayor o menor medida, compromiso.

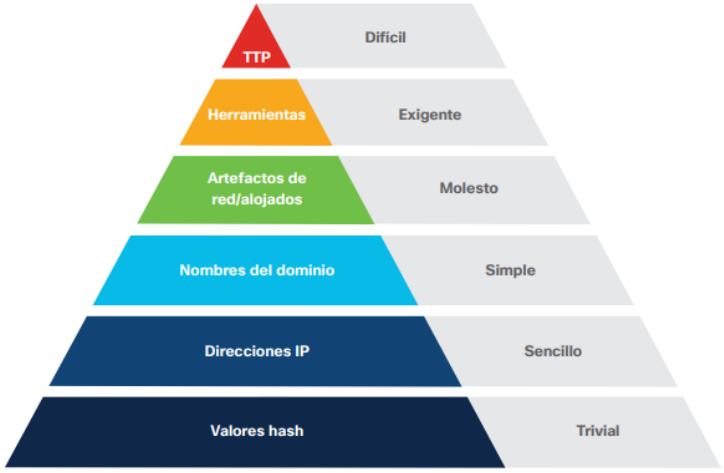


Figura 2: Pirámide del dolor [6]

Los indicadores de compromiso se pueden categorizar en niveles, esto según la pirámide del dolor (*figura 2*) y a medida que avanzan los niveles aumenta, para los atacantes, el dolor que causan los indicadores, o, en otras palabras, la dificultad para reemplazar o modificar el ataque para que no pueda ser detectado [7].

En el primer nivel tenemos los valores hash de archivos maliciosos, fáciles de recopilar y detectar, pero para el atacante, es trivial, ya que solo necesita modificar el contenido de manera mínima del archivo para que el valor cambie radicalmente, y así, no pueda ser identificado.

En el segundo y tercer nivel tenemos las direcciones IP y dominios respectivamente, las cuales se pueden bloquear o denegar a los atacantes, sin embargo, y aunque sean más difíciles de cambiar que un valor hash, se puede lograr usando las herramientas correctas.

Al subir un piso, nos encontramos con los artefactos de red y host, lo cual supone un desafío para los atacantes, ya que resulta difícil realizar acciones significativas sin dejar rastro en los logs. A nivel de host, se pueden identificar archivos, entradas de registro y cadenas en memoria, entre otros. Mientras tanto, a nivel de red, se pueden observar patrones específicos en las cadenas de agente de usuario, repeticiones de petición a recursos a una API, y tamaños de petición y respuesta, entre otros aspectos. Si bien algunos de estos elementos son modificables, son muchos los parámetros que deben ser considerados para evitar su detección en nuestros sistemas y redes.

Finalmente, en los dos últimos niveles están las herramientas y las TTP (Tácticas, Técnicas y Procedimientos), donde ambas describen la metodología utilizada por un individuo en sus ataques. El nivel de herramientas hace referencia específicamente al software (y en menor medida, al hardware) empleado para perpetrar el ataque, mientras que el nivel de TTP recoge todos los demás aspectos de la estrategia. Los indicadores de compromiso de estos niveles son más complejos, y pueden incluir detalles acerca de cómo, por ejemplo, el atacante implementa un código malicioso para llevar a cabo un reconocimiento de la red de la víctima, cómo avanza hacia un objetivo valioso y, finalmente, cómo despliega una carga útil de ransomware. Debido a su complejidad, los TTP y las herramientas requieren un gran esfuerzo por parte del defensor para ser identificados y diagnosticados.

El objetivo de esta pirámide es que, a medida que se avanza en ella, obliga a los adversarios a invertir más recursos para poder atacar la red protegida, volviéndola muy difícil de hackear en los últimos niveles.

Otra forma de clasificar los indicadores de compromiso es a través de las siguientes categorías:

- Network IoC: Son señales que indican actividades sospechosas en una red. Estas pueden incluir patrones de tráfico inusuales, conexiones a direcciones

IP o dominios maliciosos que son conocidos y el uso de protocolos o puertos inesperados.

- Host-Based IoC: Indican actividad sospechosa en un sistema. Algunas de estas son actividad de archivos inusual, procesos o servicios sospechosos en ejecución, y cambios inesperados en la configuración del sistema.

- File-Based IoC: Indican la presencia de archivos maliciosos o malware dentro de un sistema. Algunos ejemplos son hash, nombres o rutas de archivos y cadenas de texto.

- Behavioral IoC: Son indicadores que sugieren actividades sospechosas de usuarios en una red o sistema. Estos pueden incluir múltiples intentos fallidos de inicio de sesión, horarios de inicio de sesión inusuales y acceso no autorizado a datos sensibles.

2.3 Información relevante cuando se detecta compromiso

Cuando se detecta un indicador de compromiso, es fundamental tomar medidas inmediatas para minimizar el riesgo y proteger el sistema. Una de las cosas importantes que se deben hacer es analizar lo que se ha detectado y recopilar toda la información relevante del sistema comprometido. Esto incluye identificar su origen, su comportamiento, cuándo se detectó y cualquier otro detalle que pueda ser útil para la investigación.

Una vez que se ha recopilado toda la información relevante, se deben tomar medidas a corto plazo para mitigar el riesgo, y también, encontrar nuevos indicadores de compromiso que no se hayan detectado anteriormente y se puedan encontrar con lo que fue recolectado en el sistema comprometido.

A largo plazo, esta información recolectada del sistema comprometido, y post a un análisis forense, puede ayudar a la creación de nuevos indicadores de compromiso.

Según el CSIRT [8] (Equipo de Respuesta ante Incidentes de Seguridad Informática del Gobierno de Chile) algunas fuentes de información relevante que se debe recolectar dentro de sistema Linux, donde se ha encontrado compromiso relacionado a un ransomware, son:

- Registro de Logs del sistema.
- Usuarios del sistema.
- Cron Jobs o tareas programadas.
- Historial de los comandos en la bash.

Asimismo, existen también otras fuentes de información [9] a las que se acceden mediante comandos de Linux que podrían ayudar a detectar compromiso, tales como:

- Conexiones de red activas.
- Puertos abiertos.
- Últimos archivos modificados.

Toda esta información, y muchas más que no fue listada, contiene el comportamiento del sistema y de los usuarios, con esto se pueden detectar patrones de actividad sospechosa que pueden ser indicativos de actividades maliciosas.

Asimismo, cuando ya se detectó compromiso, toda la información recolectada ayudará a reconstruir lo que sucedió y el alcance del daño causado.

2.4 Herramientas utilizadas

Antes de describir la arquitectura general de la plataforma y el prototipo implementado, es importante explicar las herramientas que se utilizaron para poder diseñar y desarrollarla:

2.4.1 Python [10]

Lenguaje de programación de alto nivel, es ampliamente utilizado en las aplicaciones web, el desarrollo de software, la ciencia de datos y *el machine learning* (ML). Fue elegido debido a que es compatible con gRPC, además de ser versátil, multiplataforma, eficiente y escalable.

2.4.2 gRPC (Remote Procedure Call de Google) [11]

Framework de código abierto, se utiliza para crear servicios de comunicación entre diferentes sistemas de software distribuidos. Las razones por las cuales decidimos usar gRPC son las siguientes:

- Sencillo, ya que son simples llamadas a funciones, y no requiere tanto código.
- Flexible, se puede trabajar con diversos lenguajes de programación, esto gracias a las protocols buffers y compiladores, permitiendo la comunicación independiente del lenguaje de programación y el sistema operativo.
- Rendimiento, gRPC utiliza HTTP/2 para el envió de datos a través de la capa de transporte, reduciendo la latencia de red.
- Seguridad, cuenta con autenticación integrada mediante el uso de SSL/TLS.
- Escalabilidad, gRPC está diseñado para que sea escalable y tolerante a fallos, permitiendo su ejecución en entornos de alta disponibilidad.

En grandes rasgos, la arquitectura de gRPC se presenta en la *figura 3*, donde la comunicación se produce mediante el protocolo de red HTTP/2, y donde todo los mensajes y servicios que están especificados en el *protobuf* (.proto) se compilan en el lenguaje de programación que se vaya a utilizar, pudiendo ser distintos entre servidor y cliente.

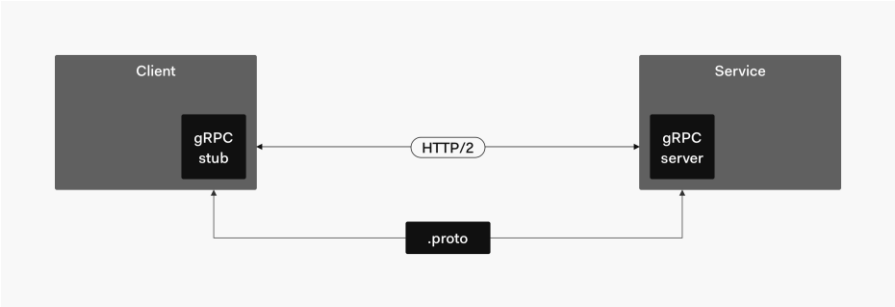


Figura 3: Arquitectura general gRPC [12]

Otro punto por destacar de gRPC, y que es gracias a la utilización del protocolo HTTP/2, son las variantes para llamadas a procedimiento remoto [13], tal como se puede ver en la *figura 4*.

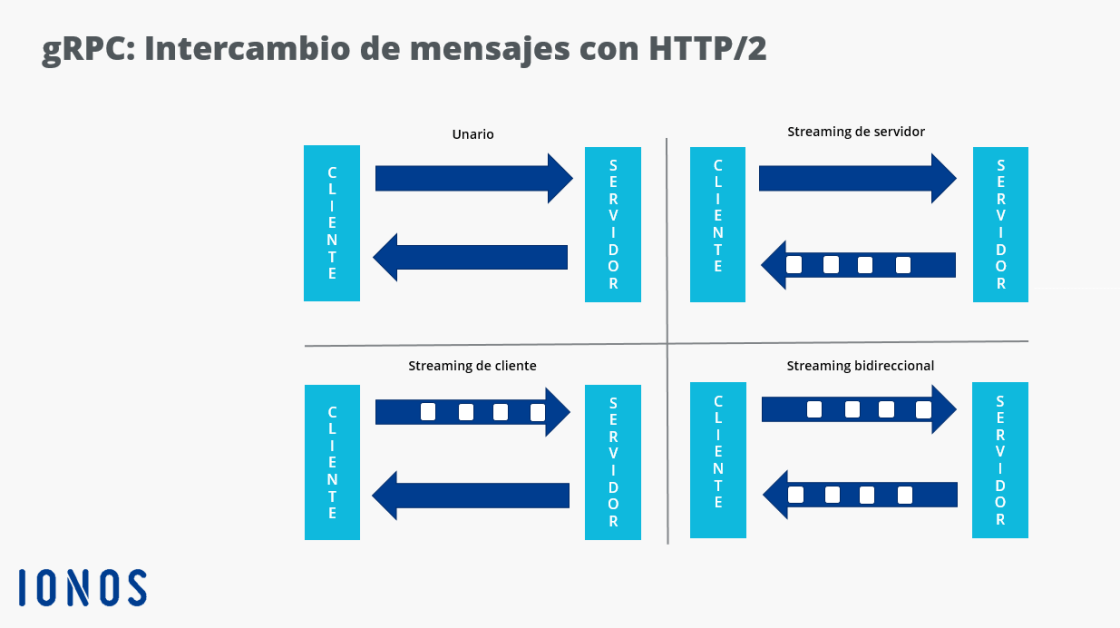


Figura 4: Variantes para llamadas a procedimiento remoto en gRPC

La primera corresponde a una única solicitud del cliente, obteniendo una sola respuesta del servidor.

En segundo lugar, tenemos un *streaming* de parte del servidor, donde el cliente envía una única solicitud al servidor, y este responde con una larga secuencia de mensajes. Asimismo, en el *streaming* del cliente tenemos el proceso invertido.

Por último, en el *streaming* bidireccional, tanto como cliente como servidor envían secuencias de mensajes, donde ambos pueden leer o escribir mensajes sin importar el orden seguido.

Es importante destacar que los clientes son los que siempre inician la comunicación, y que, al conectarse al servidor gRPC, crean los canales de comunicación, donde pueden realizar varias llamadas simultaneas. También existe un límite de llamadas simultaneas, que, cuando se supera, entran en una cola de prioridad.

2.4.3 NAGIOS [14]

Es una herramienta de monitoreo de red open source y gratuita que se utiliza para monitorear y gestionar la infraestructura TI. Es una de las herramientas de monitoreo de red más populares y utilizadas en el mundo. Nagios permite monitorear servidores, dispositivos de red, aplicaciones y servicios, y alerta a los administradores de sistemas cuando se detecta un problema. La principal ventaja y por lo cual fue utilizado en este proyecto es por su adaptabilidad a las necesidades del usuario mediante el uso de plugins, esto ayudara en el futuro para monitorear otros dispositivos en red a través de SNMP.

2.4.4 YARA (Yet Another Recursive Acronym)

Lenguaje basado en reglas que permite crear descripciones de familias de malware basadas en patrones textuales o binarios. Estas reglas están formadas por conjuntos de cadenas de caracteres y expresiones booleanas que las relacionan, y se utilizan en búsquedas sobre los ficheros del equipo sospechoso de haber sido infectado. Un IOC puede incorporar una única regla YARA en su definición, aunque ésta puede ser todo lo compleja que sea necesaria para detectar familias enteras de malware.

En la *figura 5* se puede observar un ejemplo muy simple de una regla de YARA [15], en la cual se detectaría, si se encuentran cadenas de caracteres que coincidan con los *strings* especificados y, además, cumplan las condiciones dadas.

```
rule Example
{
    strings:
        $a = "text1"
        $b = "text2"
        $c = "text3"
        $d = "text4"

    condition:
        ($a or $b) and ($c or $d)
}
```

Figura 5: Ejemplo regla YARA

2.4.5 LOKI [16]

Es una herramienta de análisis de indicadores de compromiso, que funciona bajo cuatro métodos de detección, que se detallan a continuación:

- Nombre del archivo IoC (Coincidencia de expresiones regulares en la ruta o nombre completo del archivo).
- Comprobación de las reglas con YARA.
- Comparación de hashes maliciosos conocidos (MD5, SHA1 y SHA256), mediante el escaneo de archivos.
- Compara los puntos finales de conexión de los procesos con las C2 de IoC.

Con respecto a la pirámide del dolor mostrada anteriormente, LOKI, junto con YARA, son capaces de detectar indicadores de compromiso de varios niveles como hashes, direcciones IP, dominios y artefactos de red.

Si bien hay opciones que tienen mayor cantidad de reglas que definen indicadores de compromiso (*tabla 1*), elegimos esta debido a que es gratis, *open source*, está construida en Python y, además, tiene compatibilidad con Windows, lo que nos permitiría escalar este proyecto a hosts con este sistema operativo.

	LOKI	SPARK Core	SPARK	THOR
Type	Free / Open Source	Free / Registration Required	Enterprise Product	Enterprise Product
Main Use Case	Preventive Scanning / Triage	Preventive Scanning / Triage	Preventive Scanning / Triage	Incident Response / Live Forensics
Platform	Windows (precompiled), Linux / macOS (source with dependencies)	Windows, Linux, macOS	Windows, Linux, macOS	Windows
Size (Binaries)	8 MB	9 MB	9 MB	16 MB
Language	Python	Go	Go	Python
Modules	3	5	9	26
Bundled Signatures	Open Source (~3000 YARA rules)	Open Source (~3000 YARA rules)	THOR's Signature Set (~9000 YARA rules)	THOR's Signature Set (~9000 YARA rules)
Support and Testing	Github README & Issues, Travis-CI	Manual & Github Issues, Internal CI	Manual & Support Portal, Internal CI	Manual & Support Portal, Internal CI
Special Extras	Levenshtein check PESieve check Double Pulsar check	JSON output SYSLOG (tcp/udp/ssl) Scan Throttling	JSON output SYSLOG (tcp/udp/ssl) Scan Throttling	... a lot, see comparison

Tabla 1: LOKI y otras opciones [17]

2.4.6 CFSSL [18]

CFSSL es una herramienta útil para la gestión y el control de la seguridad en entornos de red y nube, permitiendo la creación, firma y verificación de certificados X.509 y SSL/TLS. Con esta herramienta se crearon los certificados necesarios para establecer una comunicación segura entre cliente y servidor, asegurando de esa forma la confidencialidad e integridad de toda la información que se transmite mediante gRPC.

2.4.7 SQLite [19]

Sistema de gestión de datos relacional de código abierto. Para el prototipo, se utilizó SQLite debido a que no necesita servidor, haciéndola fácil de usar, portable, estable y eficiente.

2.4.8 Wireshark [20]

Wireshark es una herramienta para capturar y analizar el tráfico de red en tiempo real, permitiendo ver todos los paquetes de datos que se envían y reciben a través de una red, con toda la información relacionada a ellos, como el protocolo o servicios utilizados.

Sin embargo y debido a que es un *sniffer*, este puede ser utilizado para fines malintencionados, violando la confidencialidad o integridad de la información transmitida en la red, como en un ataque *Man in the Middle*.

2.4.9 Nessus Essentials [21]

Nessus Essentials es un software que permite escanear redes y sistemas con el objetivo de encontrar vulnerabilidades y amenazas de seguridad, proporcionando informes detallados sobre estas. Es una versión gratuita de Nessus limitada para escanear un máximo de 16 direcciones IP por red.

2.5.10 NRPE

Nagios Remote Plugin Excecutor es un agente que se aloja en un host remoto, recibe ordenes de ejecución de plugins de Nagios y devuelve la información al servidor que lo ejecutó en caso de ser necesario, tal como se aprecia en la *figura 6*. Estos plugins pueden realizar varias acciones como revisar el uso de recursos del sistema, ejecutar scripts remotamente, interactuar con aplicaciones externas, entre otros. Con la información obtenida de los plugins ejecutados se pueden desencadenar acciones desde el servidor de monitoreo en caso de ser necesario.

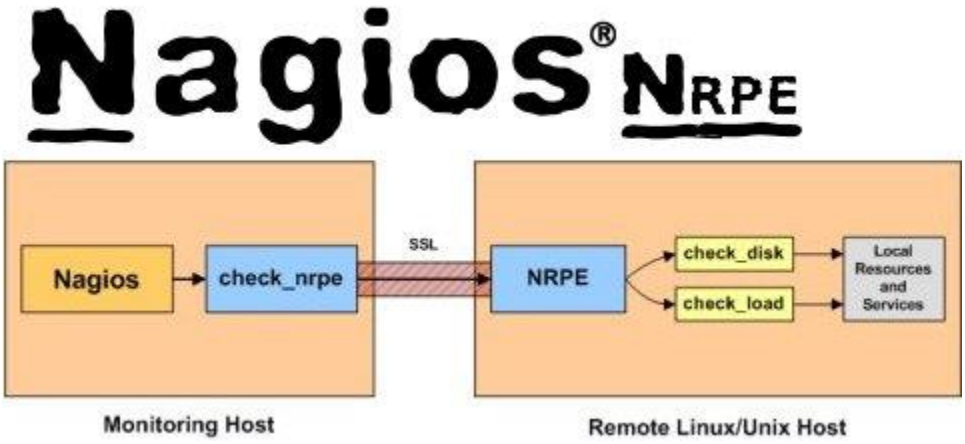


Figura 6: Funcionamiento de NRPE [22]

3. Desarrollo

Dentro de lo desarrollado, se encuentra un diseño de una arquitectura general y de un prototipo, siendo este último implementado como prueba de concepto de este proyecto.

3.1 Arquitectura general

Como se ve en el diseño de arquitectura general en la *figura 7* hay 3 servidores gRPC que reciben la información desde los hosts que estén dentro de la red local, para así guardarla en la base de datos y poder entregarla a aplicaciones de terceros cuando sea solicitada.

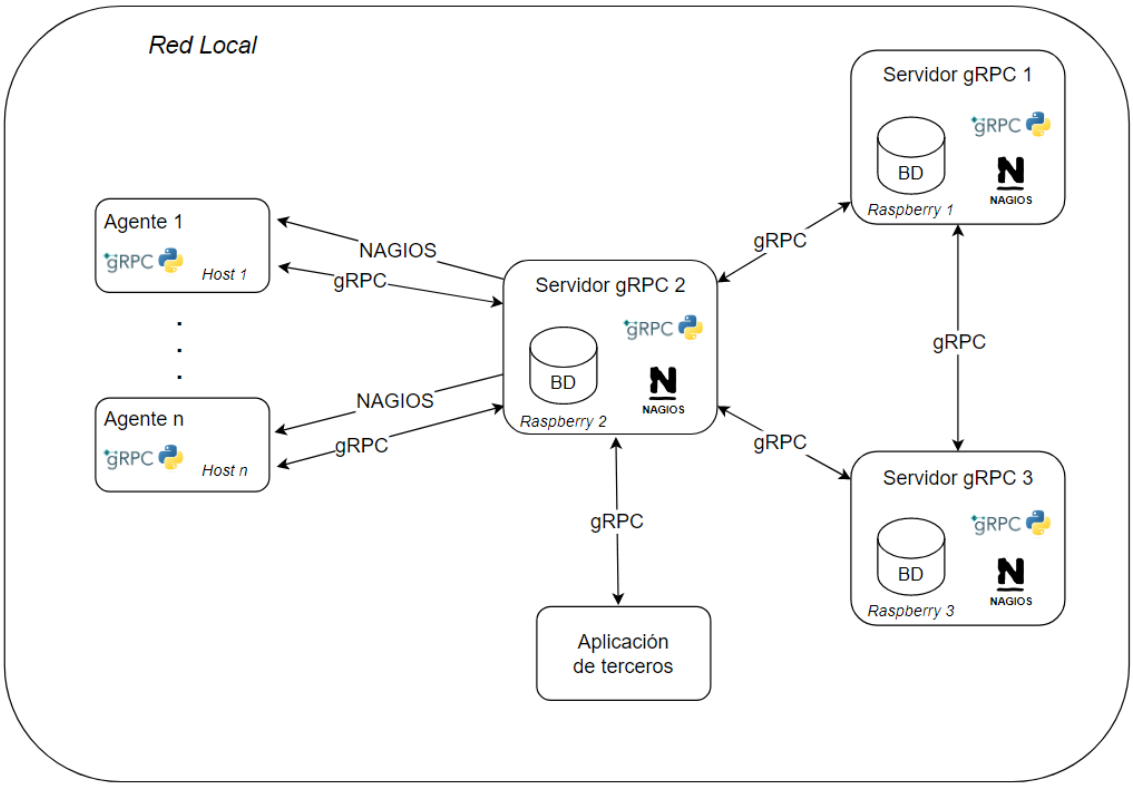


Figura 7: Arquitectura general

Cada uno de los elementos se detalla a continuación:

3.1.1 Host

Estaciones de trabajo y servidores, dentro de cada uno tenemos un agente que mediante gRPC se comunica con servidores gRPC constantemente, a la espera de una solicitud de reporte o detección propia de un indicador de compromiso.

3.1.2 Raspberry PI

Dentro de cada una de ellas 3 tenemos un servidor gRPC, NAGIOS, y una base de datos, donde el primero se encarga de la comunicación constante con los hosts, el segundo de monitorear el comportamiento de los hosts y sus servicios en busca de indicadores de compromiso o anomalías, y la tercera de guardar la información recibida en el servidor gRPC, es importante recalcar que las bases de datos son exactamente iguales en todas.

3.1.3 Aplicación de terceros

Mediante gRPC, el servidor envía la información solicitada por la aplicación de terceros. Esta puede solicitar los reportes que hayan sido recopilados o indicadores

que hayan sido detectados, ya sea en tiempo real, o como una consulta individual de un host.

3.1.4 gRPC

Con el *framework* gRPC se logra una comunicación encriptada (mediante uso de certificados SSL/TLS) entre host, servidores gRPC y aplicaciones de terceros de toda la plataforma.

Gracias a la comunicación encriptada de la información que se transmite dentro de la plataforma, se aseguran la confidencialidad e integridad. Por otro lado, la disponibilidad está cubierta mediante el uso de múltiples Raspberry Pi, donde cada una contiene un servidor gRPC, para que así, cuando alguna de ellas presente algún problema (*figura 8*), otra pueda cubrir rápidamente sus funciones (*figura 9*), entregando redundancia a la arquitectura. Además, toda la información recibida en alguna de ellas y guardada en la base de datos, se replica en las demás.

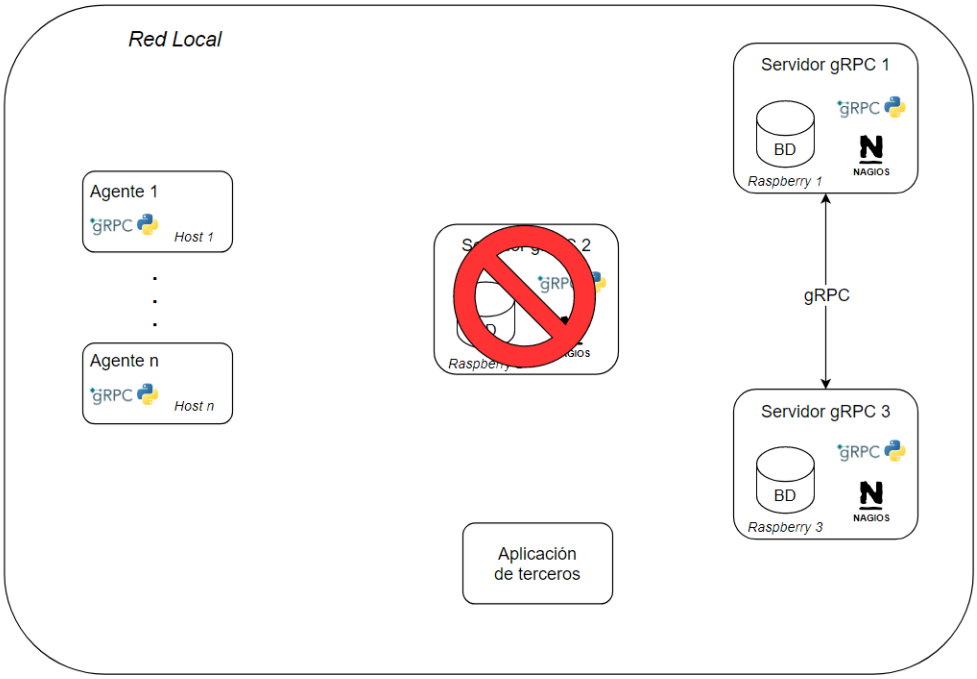


Figura 8: Caída de un servidor gRPC

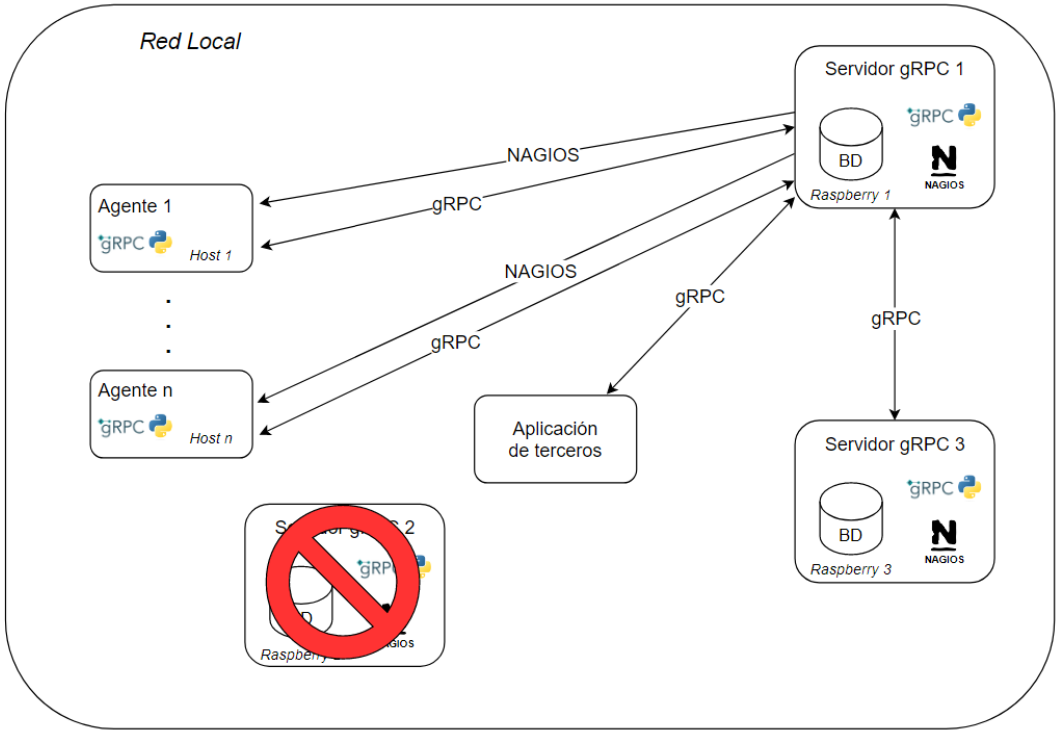


Figura 9: Reconexión plataforma

3.2 Arquitectura prototipo

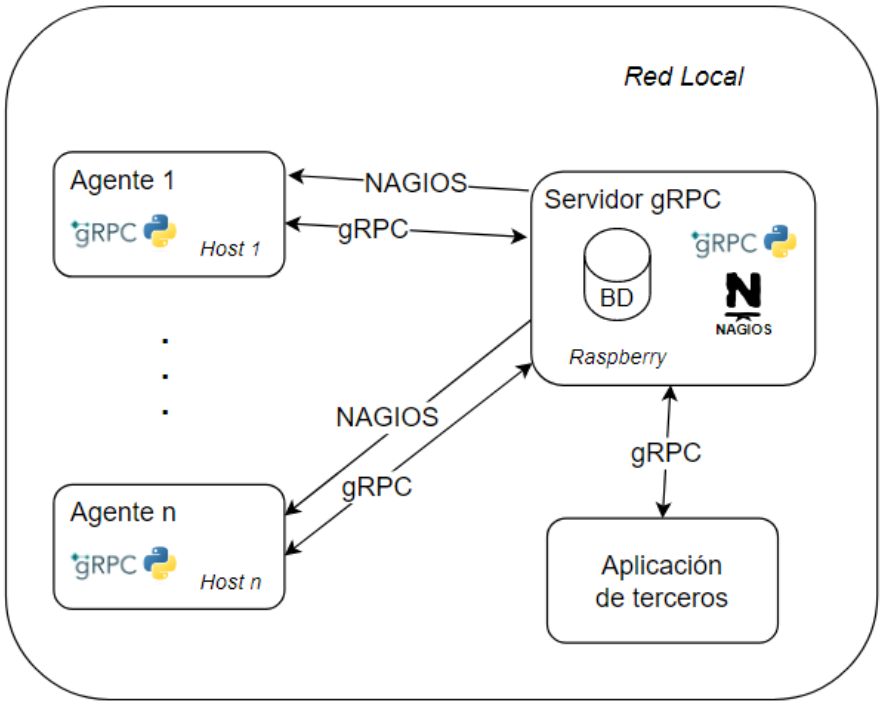


Figura 10: Arquitectura prototipo

Primero, es importante considerar que dentro de la arquitectura del prototipo (*figura 10*) y lo que se implementó, se toma como una primera prueba de concepto de la arquitectura general planteada anteriormente, con los siguientes objetivos:

- Implementación de un canal de comunicación seguro para toda la información que se transmita dentro de la plataforma.

- Monitoreo simple de indicadores de compromiso, y que al momento de detectarse alguno se notifique al servidor.

- Solicitud de reporte con información relevante al host donde se detecte compromiso.

Dado el tiempo limitado disponible para la implementación del prototipo, se optó por priorizar la confidencialidad e integridad dentro de la triada CIA. En consecuencia, el diagrama muestra que, en lugar de utilizar tres Raspberry Pi, se usó solo una. Con esto se pierde la redundancia porque conlleva el riesgo de que, si el dispositivo presenta algún problema, la plataforma estaría inaccesible.

Hay que destacar también, que en este prototipo no se usará un servidor de base de datos local, si no, una base de datos de un solo archivo con SQLite, que estará en el único servidor.

3.3 Prototipo implementado

3.3.1 Base de datos implementada

La base de datos implentada (*figura 11*), utilizando SQLite como herramienta, es la siguiente:

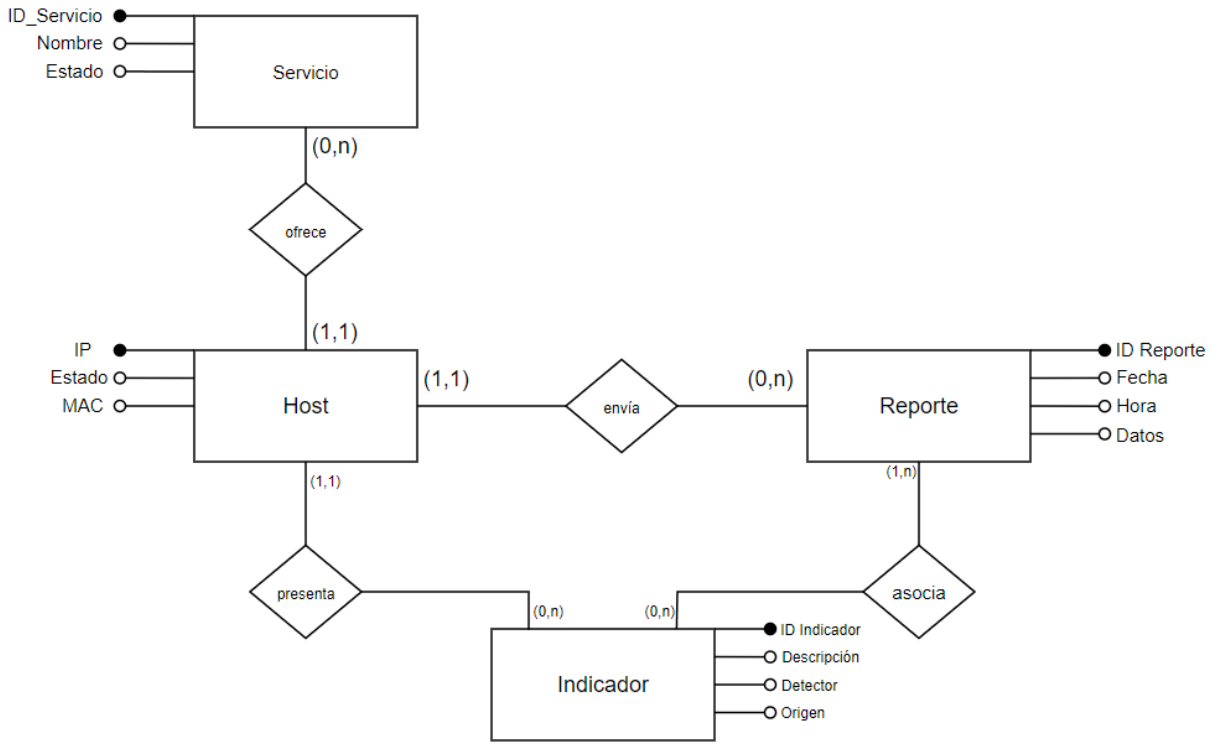


Figura 11: Base de datos implementada

- **Indicador:** Estos son creados cuando un indicador de compromiso es encontrado durante el monitoreo. Estos contienen un ID, una descripción (se especifica el indicador de compromiso detectado) y el detector (que es LOKI). Se relaciona con el host donde fue encontrado y también con los reportes que se pidan a partir de su detección.

- **Host:** Esto corresponde a cada estación de trabajo o servidor que tenga un agente para comunicarse con el servidor gRPC, estos tienen una dirección IP, un estado (disponible o no disponible) y una dirección MAC. Los hosts pueden tener indicadores o reportes asociados.

- **Reporte:** El reporte es solicitado a un host cuando se encuentra un indicador de compromiso en el mismo o cuando hay una petición global de estos, este reporte tiene un ID, fecha y hora de creación y los datos del reporte, que es la información que se recolectó y servirá posteriormente para la creación o detección de indicadores de compromiso. Un reporte está asociado al host

donde se recolectó la información que lo describe, y también al indicador o indicadores que desencadenaron su petición.

- **Servicio:** Cada host puede tener de 0 a n servicios (las estaciones de trabajo no tienen, solo los servidores), los cuales son monitoreados por NAGIOS, y cuando sucede alguna anomalía o caída de estos, se cambiaría el estado de este, y se le pediría reporte al host.

3.3.2 Información recolectada en los reportes

Se recolecta información que puede ser relevante para concluir si es que un sistema ha sido comprometido, mediante información del sistema extraída a través de comandos de la terminal de Linux, que se detalla a continuación:

- **export LC_ALL=C**: Salida de la consola en inglés.
- **netstat**: Lista de todas las conexiones activas en el sistema.
    - **netstat -antup | grep 'ESTABLISHED'**
      Conexiones TCP y UDP activas en el estado ESTABLISHED.
    - **netstat -antup | grep 'LISTEN'**
      Conexiones TCP y UDP activas en el estado LISTEN.
    - **netstat -antup | grep 'TIME_WAIT'**
      Conexiones TCP y UDP activas en el estado TIME_WAIT
    - **netstat -tulpna**
      Lista de todos los puertos abiertos en el sistema con los procesos que los están utilizando.
- **ss**: Información detallada de las conexiones de red activas.
    - **ss | grep ssh**
      Conexiones relacionadas con SSH.
- **/etc/passwd**: Archivo que contiene la información de los usuarios registrados en el sistema.
    - **cut -d: -f1 /etc/passwd**
      Listado de los nombres de usuario registrados en el sistema.
- **/var/log/auth.log**: Archivo que registra todas las actividades de autenticación, con esto se pueden monitorear posibles intentos de intrusión.
    - **cat /var/log/auth.log**
      Archivo /var/log/auth.log en salida estándar.
- **history**: Listado de los últimos comandos utilizados.
- **last**: Listado de los últimos inicios de sesión.
- **ps**: Lista de procesos que se están ejecutando.
    - **ps auxf**
      Listado jerárquico de todos los procesos en ejecución, con información detallada de cada uno, con su relación padre-hijo.
- **crontab**: Programar tareas para su ejecución automática en un momento especifico.

    - **crontab -l**
      Lista todas las tareas programadas del usuario actual.
- **/var/log/syslog**: Archivo que almacena mensajes y eventos del sistema, con él se pueden llegar a identificar actividades sospechosas o maliciosas, ya que contiene mucha información sobre el equipo y lo que sucede en él.
    - **cat /var/log/syslog**
      Archivo /var/log/syslog en salida estándar.

Toda esta información recolectada, se estructura dentro de Python como un diccionario, para posteriormente ser enviada a través de gRPC, mediante el método RPC *SubmitReport* al servidor, y ser guardado en la base de datos.

La solicitud de reportes de parte del servidor gRPC a los agentes en los hosts en la red se establece en los siguientes casos:

- Detección de indicador(es) de compromiso en un host por LOKI.
- Anomalía en el comportamiento de un servicio de un host monitoreado por NAGIOS.
- Cada cierto tiempo establecido.

Cabe destacar, que en los primeros 2 casos, solo se le solicita reporte en el host donde se detectó compromiso o comportamiento anómalo, y en el último, es de forma global a todos los hosts en la red.

3.3.3 gRPC y un canal de comunicación seguro

Toda la plataforma se comunica utilizando los mensajes y servicios de gRPC especificados anteriormente, lo cual establece un canal seguro mediante conexión SSL (*figura 12*). Se crearon los certificados con la herramienta CFSSL para cada servidor gRPC y host, además de NAGIOS y aplicaciones de terceros.

Estos certificados, que son creados bajo un certificado raíz, utilizan el cifrado RSA-2048 (asimétrico, donde la clave es de 2048 bits ya que es lo recomendado por el NIST [23]), donde cada cliente y servidor tiene sus propias claves públicas y privadas, permitiendo una autenticación de parte del cliente. También, hay que destacar que estos certificados tienen fecha de expiración, que, en esta implementación, se estableció en un año.

Figura 12: Conexión SSL

También, como autenticación adicional, se cifró la clave privada del host mediante cifrado simétrico, utilizando una librería de Python llamada Fernet [24], lo que significa que para poder iniciar la conexión con el servidor gRPC, es necesario ingresar la contraseña establecida para descifrar la clave privada del host.

Lo anterior se puede apreciar en el fragmento de código de la *figura 13*, donde se crea la llave con la contraseña ingresada por consola, y con esto se desencripta el certificado encriptado, pudiendo generar las credenciales necesarias para iniciar la comunicación.

```
llave = input() #Se pide la llave para desencriptar la clave privada
f = Fernet(llave) #Se crea el objeto de la llave
with open("certificates/host-key-encrypted.pem","rb") as encrypted_file:
    encrypted = encrypted_file.read() #Se lee el archivo encriptado
decrypted =  f.decrypt(encrypted) #Se desencripta el archivo
credentials = grpc.ssl_channel_credentials(open('certificates/ca.pem','rb').read(),
decrypted,open('certificates/host.pem','rb').read()) #Se crean las credenciales
```

Figura 13: Código encriptación de la clave privada del host

Por último, al iniciar la comunicación, se realiza una verificación de los archivos Python ocupados por la plataforma de parte del host, mediante el método RPC *ServerComprobationMD5*, donde si no coinciden con los de la base de datos que posee el servidor gRPC, se notifica como indicador de compromiso, pidiéndole reporte al agente y posterior a eso cerrando la comunicación.

Todo lo anterior, asegura confidencialidad e integridad en todos los pasos de la comunicación dentro del prototipo, lo cual es esencial.

3.3.4 Concurrencia de gRPC

Dentro de gRPC, las solicitudes de parte de los clientes conectados se realizan de forma simultánea, o, dicho de otra manera, cada solicitud se maneja como un subproceso, y todos estos ejecutan tareas de forma concurrente. Lógicamente, existe un límite máximo, que cuando es sobrepasado, las solicitudes que no se puedan realizar entran en una cola de prioridad, resolviéndose cuando exista disponibilidad.

Asimismo, en la implementación realizada, y debido a que la cantidad de clientes gRPC es baja (máximo 3, que incluyen NAGIOS, el host y la aplicación de terceros), la cantidad máxima de solicitudes simultáneas se estableció en 10 (*figura 14*).

```
server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
```

Figura 14: Cantidad máxima de solicitudes simultaneas

3.3.5 Mensajes en .proto

Dentro de gRPC, los protobuffers (proto) se utilizan para definir los mensajes y servicios con sus métodos RPC que se utilizarán en una aplicación distribuida. Con estos mensajes y servicios, se envían y se reciben los datos entre servidor y cliente utilizando llamadas remotas.

Los mensajes son declarados en el *protobuf*, y están estructurados como se ve en la *figura 15,* donde se le entrega un nombre, y dentro contiene campos de información, los cuales tienen un tipo (*string, bool, int, float, double,* entre otros), un nombre y un tag o identificador.

```
message ClientMessage{
  string ip = 1; // IP del host que envia el mensaje
  string message = 2; // Mensaje a enviar
}

message SoftwareMessage{ ···
}
```

Figura 15: Ejemplo de mensaje en el protobuf

Los mensajes declarados en el archivo *communication.proto* para la implementación del prototipo son los siguientes, donde el tipo de los campos siempre es *string*:

ClientMessage: Utilizado por los hosts (clientes gRPC) que se conectan al servidor gRPC, específicamente para la comunicación bidireccional, la cual es cada 1 minuto.

1. **ip:** Dirección IP del host donde se encuentra el agente.

2. **message:** Mensaje para enviar al servidor, que puede ser, por ejemplo, “No hay problema” o “Tengo un problema”.

ServerMessage: Utilizado por el servidor gRPC para responder a los hosts o aplicaciones de terceros.

1. **message:** Mensaje de respuesta del servidor tras recibir un mensaje, que puede ser “Ok” si es que no se ha detectado un indicador de compromiso o “Envíame tu reporte” si es que se recibió uno, entre otros.

ReportMessage: Utilizado por el cliente para enviar los reportes que le solicita el servidor, como también por el servidor para enviar los reportes tras consultas de una aplicación de terceros.

1. **ip:** Dirección IP del host al que pertenece la información.
2. **json:** JSON del reporte, que contiene toda la información recolectada en el host, incluido un *TimeStamp* de cuando se recolectó. Esto se detallará a profundidad en la sección 3.3.4.

IndicatorMessage: Utilizado por el cliente para enviar los indicadores de compromiso que detecte, como también por el servidor para enviar los indicadores tras consultas de una aplicación de terceros.

1. **ip:** Dirección IP del host donde fue detectado el indicador.
2. **timestamp***: TimeStamp* de cuando fue detectado el indicador.
3. **indicator:** Descripción del indicador, donde se provee la información necesaria para saber el riesgo del indicador de compromiso detectado.
4. **detector:** Detector del indicador de compromiso, que puede ser LOKI o MD5, este último en caso de que la verificación del hash falle.

ReportXIndicador: Utilizado para relacionar reportes con indicadores.

1. **idReport:** ID del reporte que fue generado al detectar el indicador de compromiso.
2. **idIndicator:** ID del indicador que desencadeno la petición del reporte al host.

ComprobationMD5: Utilizado por el cliente para enviar el hash de un archivo, con el objetivo de que el servidor compruebe que es el correcto.

1. **ip:** Dirección IP del host donde se comprobará el hash.
2. **md5:** MD5 del archivo a comprobar.
3. **file:** Archivo por comprobar.

SpecificRequest: Utilizado por una aplicación de terceros para realizar consultas específicas sobre indicadores o reportes en el servidor.

1. **ip:** Fecha de inicio de la petición específica.
2. **start:** Fecha de fin de la petición específica.
3. **end:** IP del host que se quiere revisar.

SoftwareMessage: Utilizado por una aplicación de terceros para entregar su nombre, que se utilizará como identificador en el servidor y así responder a sus peticiones.

1. **name:** Nombre de la aplicación de terceros que hace la petición.

3.3.6 Servicios y métodos RPC en .proto

Los servicios son definidos en el *protobuf* y contienen métodos RPC, los cuales están compuestos de un nombre, un mensaje de entrada y uno de salida *(figura 16)*. Estos métodos RPC son llamados por el cliente gRPC, donde el servidor responde dependiendo de la solicitud.

```
service Communication {

  rpc SubmitReport (ReportMessage) returns (ServerMessage); // Envia un reporte al servi
  rpc BidirectionalCommunication (stream ClientMessage) returns (stream ServerMessage);
  rpc IndicatorReport (IndicatorMessage) returns (ServerMessage); // Envia un reporte de
  rpc SaveIndicatorReport (ReportXIndicator) returns (ServerMessage); // Envia ids para
  rpc ServerComprobationMD5 (ComprobationMD5) returns (ServerMessage); // Envia un md5 p
  rpc StreamingServerIndicator  (SoftwareMessage) returns (stream IndicatorMessage); //
  rpc StreamingServerReport (SoftwareMessage) returns (stream ReportMessage) ; // Envia
  rpc IndicatorRequest (SpecificRequest) returns (stream IndicatorMessage); // Envia un
  rpc ReportRequest (SpecificRequest) returns (stream ReportMessage); // Envia un report
}
```

Figura 16: Métodos RPC definidos en el protobuf

Especificados en el *communication.proto* al igual que los mensajes, los métodos RPC definidos dentro del servicio gRPC son los siguientes:

BidirectionalCommunication: Utilizado para la comunicación constante entre servidor y agente en el host.

SubmitReport: Utilizado para el envió del reporte con la información del host desde el agente en el host al servidor.

IndicatorReport: Utilizado para el reporte de un indicador, ya sea desde el agente en el host o NAGIOS hacia al servidor.

SaveIndicatorReport: Relaciona los indicadores con los reportes de un host, que son enviados desde el agente en el host al servidor.

ServerComprobationMD5: Utilizado para el *hash* de los programas (cliente.py && LOKI.py) en el host, donde el servidor al recibirlos comprueba esta información consultando a la base de datos si es correcta, si no lo es corta la comunicación y se notifica como un claro indicador de compromiso.

StreamingServerIndicator: Utilizado para el reenvío en tiempo real de los indicadores recibidos en el servidor gRPC a la aplicación de terceros.

StreamingServerReport: Utilizado para el reenvío en tiempo real de los reportes recibidos en el servidor gRPC a la aplicación de terceros.

IndicatorRequest: Tras una solicitud desde la aplicación de terceros, se envían todos los indicadores que cumplan los requisitos solicitados.

ReportRequest: Tras una solicitud desde la aplicación de terceros, se envían todos los reportes que cumplan con los requisitos solicitados.

3.3.7 Funcionamiento de LOKI

LOKI, como se mencionó anteriormente, es una herramienta de análisis de indicadores de compromiso que está construida sobre Python.

Esta herramienta se utiliza en cada host dentro de la red, y cada cierto tiempo establecido, se realiza un análisis del sistema en busca de indicadores de compromiso. Al momento de detectar alguno(s), LOKI lo exporta como una línea de texto en un archivo de texto llamado *indicadores.log*, donde cada una de ellas se ve de esta forma:

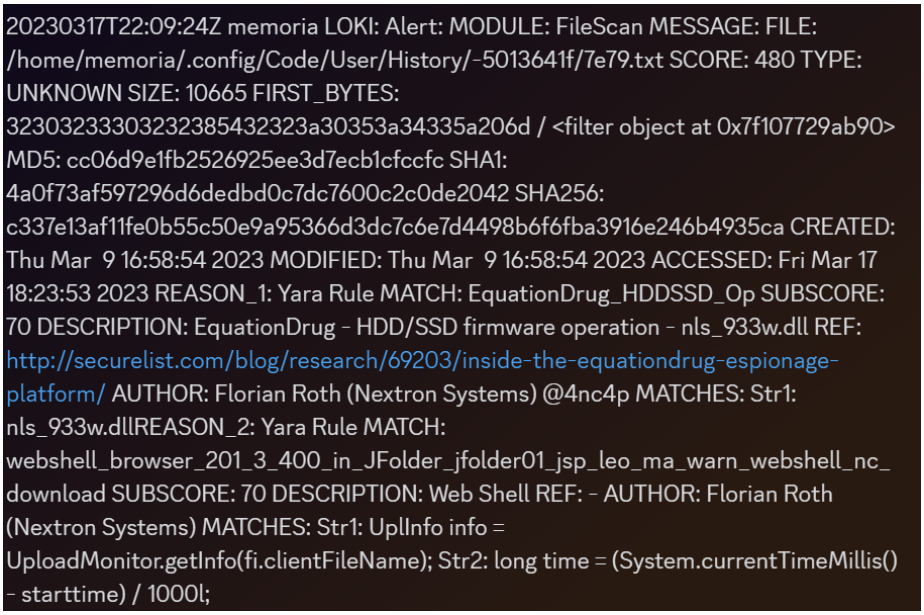


Figura 17: Indicador de compromiso detectado por LOKI

Por el lado del agente en el host, busca, cada cierto intervalo de tiempo, nuevos indicadores de compromiso detectados por LOKI en el archivo de texto, y cada uno de ellos, con una previa organización de la información que se compone como *JSON* (*figura 18*), es enviado como indicador al servidor gRPC, donde es guardado en la base de datos, y además se le solicita un reporte al host.

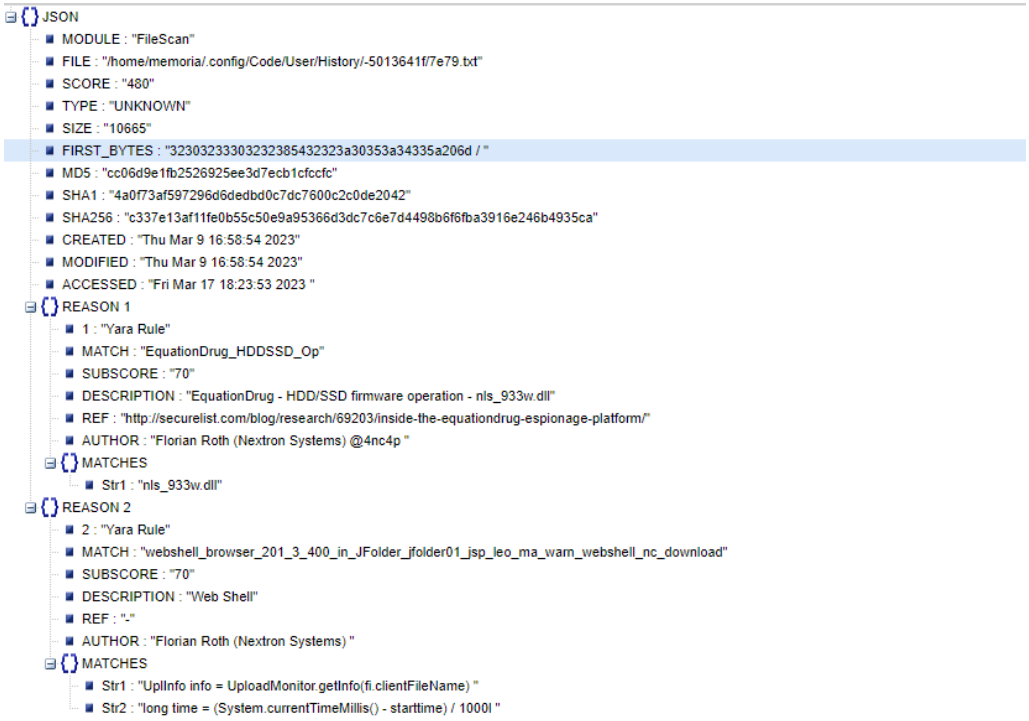


Figura 18: Indicador de compromiso organizado como JSON

3.3.8 Funcionamiento de NAGIOS

NAGIOS, dentro del prototipo implementado, no detecta indicadores de compromiso, solo monitorea hosts y sus servicios utilizando NRPE.

Si se encuentra alguna anomalía al monitorear los servicios, mediante un script creado (*communicationNagios.py*) y utilizando el método RPC *IndicatorReport*, le notifica al servidor gRPC, el cual solicita reporte al host donde se detectó.

3.3.9 Funcionamiento API para aplicaciones de terceros

Esta API tiene como objetivo poder recibir, en tiempo real, los indicadores detectados y los reportes recopilados, o realizar consultas sobre los indicadores y reportes almacenados en la base de datos utilizando el canal de comunicación ya existente por gRPC.

Para recibir en tiempo real los indicadores y reportes, se crearon los métodos RPC *StreamingServerIndicator* y *StreamingServerReport*, respectivamente. Dentro de la implementación de los métodos, de parte del servidor gRPC, se crean diccionarios (*figura 19*), donde las claves son los nombres de la aplicación de terceros, y el valor es una lista, para encolar los indicadores o reportes que vayan llegando al servidor gRPC.

Estos revisan constantemente si la lista donde se encolan los indicadores o reportes está vacía, si no lo está, se envía lo que esté encolado en la respectiva lista a la aplicación de terceros.

```
global dicIndicadoresAPI
dicIndicadoresAPI = {}

global dicReportesAPI
dicReportesAPI = {}
```

Figura 19: Definición de los diccionarios para la API

Por otro lado, para las consultas de indicadores y reportes a base de datos, se crearon los métodos RPC *IndicatorRequest* y *ReportRequest*, respectivamente.

Estos métodos reciben una IP de un host, una fecha de inicio y de fin para realizar la petición. Estos datos son enviados al servidor gRPC, el cual le consulta a la base de datos, y con lo recolectado se reconstruyen los mensajes de indicadores o reportes que correspondan, enviándolos a la aplicación de terceros.

3.3.10 Formas de detección de indicadores de compromiso

Los métodos o herramientas utilizadas para detectar indicadores de compromiso se detallan en la siguiente tabla, donde se especifica las categorías o niveles de la pirámide del dolor que cubren:

Método de detección de indicadores de compromiso	Pirámide del dolor	Categorías
LOKI	Hash de archivos Direcciones IP Dominios Artefactos de red (limitado a archivos) Herramientas	File-Based Network
Verificación hash	Hash de archivos	File-Based

Tabla 2: Formas de detección de indicadores de compromiso según la pirámide del dolor y categorización

4. Experimentos y Resultados

4.1 Características hardware utilizadas

El prototipo implementado fue probado en 2 computadores, uno como servidor gRPC utilizando una Raspberry Pi, y otro como host utilizando un computador de escritorio. A continuación, se detallan sus características hardware y el sistema operativo utilizado de cada uno de ellos:

4.1.1 Servidor gRPC

- **Modelo:** Raspberry Pi 4b.
- **Memoria RAM:** 4 GB RAM.
- **Procesador:** 64-bit quad-core Cortex-A72.
- **Almacenamiento:** 128 GB (microSD).
- **Red:** Wifi 802.11ac de 2.4 GHz y 5.0 GHz.
- **Sistema Operativo:** Raspberry PI OS (Debian 11).

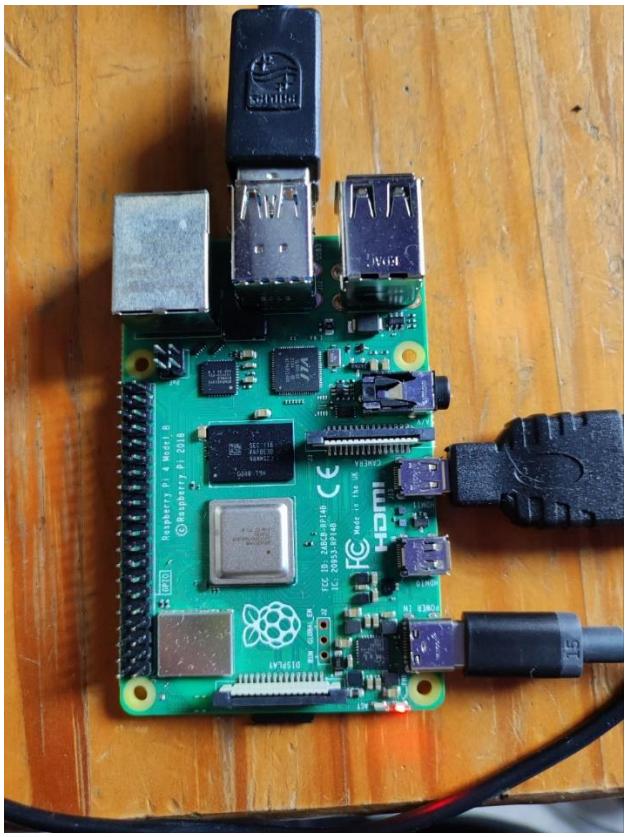


Figura 20: Raspberry utilizada

4.1.2 Host

- **Memoria RAM:** 16 GB.
- **Procesador:** Intel Core i7-8700.
- **Almacenamiento:** SSD 512 GB.
- **Red:** Ethernet Gigabit.
- **Sistema Operativo:** Ubuntu OS.

Figura 21: Computador utilizado como host

4.2 Experimentos relacionados con la tríada CIA

Con el objetivo de probar que la confidencialidad y la integridad de la información trasmitida y almacenada por la plataforma están aseguradas, se realizaron los siguientes experimentos.

4.2.1 Experimento con herramienta de *sniffer* (Wireshark)

Para probar que existe confidencialidad e integridad de toda la información que se transmite por la plataforma, se utilizó la herramienta Wireshark (*figura 22*) como *sniffer* para monitorear la red.

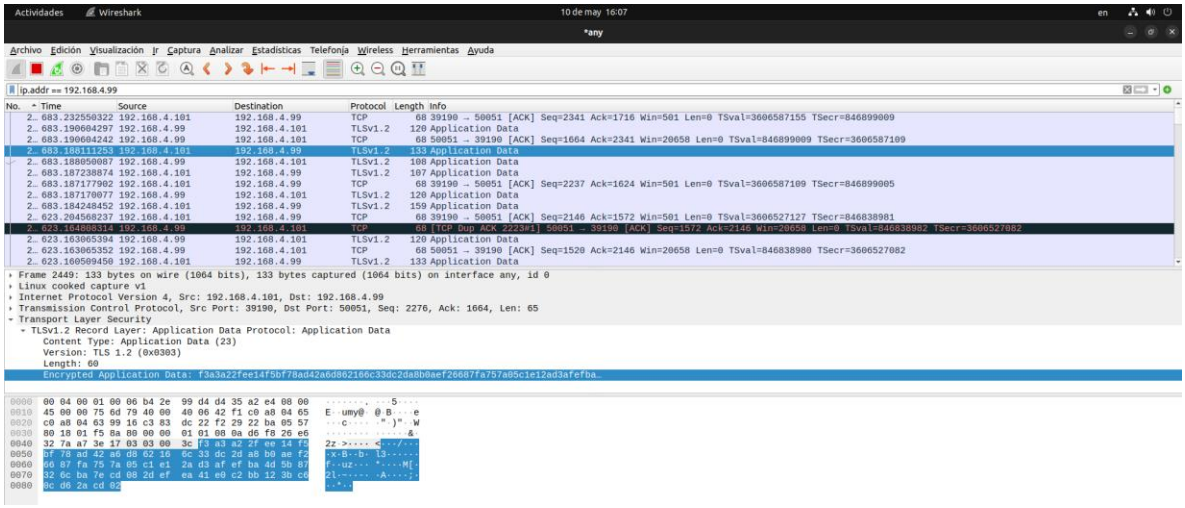


Figura 22: Wireshark

Se filtraron los paquetes enviados y recibidos por el equipo con dirección IP 192.168.4.99 que corresponde al servidor gRPC, en la *figura 23* se puede observar que los protocolos utilizados son TCP y TLSv1.2, siendo el último el utilizado para enviar la información dentro de la plataforma.

Source	Destination	Protocol	Length	Info
192.168.4.101	192.168.4.99	TCP	68	39190 → 50051 [ACK] Seq=2341 Ack=1716 Win=501 Len=0 TSval=3606587155 TSecr=846899009
192.168.4.99	192.168.4.101	TLSv1.2	120	Application Data
192.168.4.99	192.168.4.101	TCP	68	50051 → 39190 [ACK] Seq=1664 Ack=2341 Win=20658 Len=0 TSval=846899009 TSecr=3606587109
192.168.4.101	192.168.4.99	TLSv1.2	133	Application Data
192.168.4.99	192.168.4.101	TLSv1.2	108	Application Data
192.168.4.101	192.168.4.99	TLSv1.2	107	Application Data
192.168.4.101	192.168.4.99	TCP	68	39190 → 50051 [ACK] Seq=2237 Ack=1624 Win=501 Len=0 TSval=3606587109 TSecr=846899005
192.168.4.99	192.168.4.101	TLSv1.2	120	Application Data
192.168.4.101	192.168.4.99	TLSv1.2	159	Application Data
192.168.4.101	192.168.4.99	TCP	68	39190 → 50051 [ACK] Seq=2146 Ack=1572 Win=501 Len=0 TSval=3606527127 TSecr=846838981

Figura 23: Filtrado de los paquetes según la dirección IP del servidor gRPC en Wireshark.

Al seleccionar alguno de estos paquetes que utilizan el protocolo TLSv1.2, nos encontramos con que la información de la aplicación se encuentra encriptada, tal cual como se puede apreciar *figura 24*, probando de esta manera que la información que se transmite dentro de la plataforma no es visible y no puede ser modificada, confirmando confidencialidad e integridad de la comunicación.

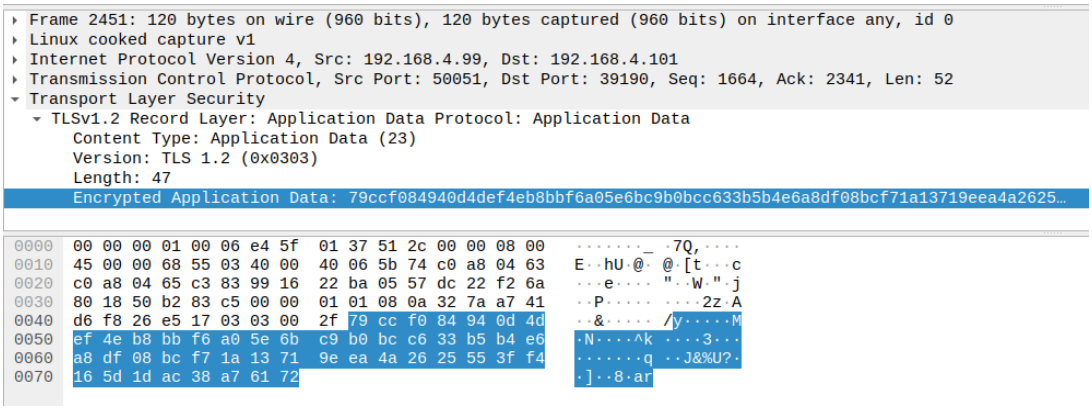


Figura 24: Visualización de la información encriptada transmitida por gRPC

4.2.2 Análisis de vulnerabilidades con Nessus

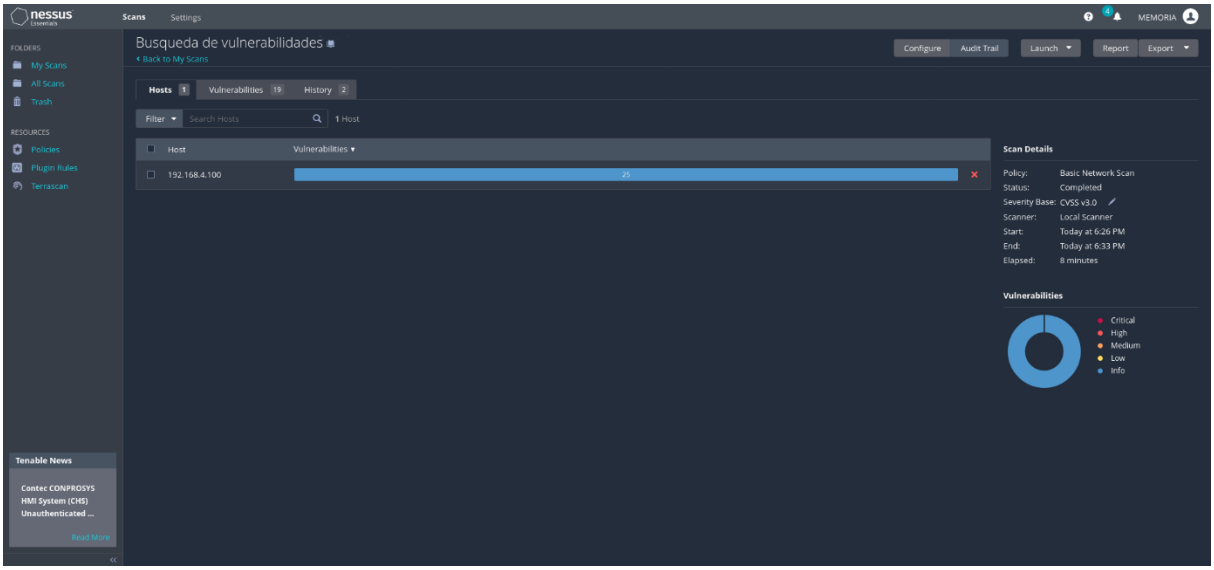


Figura 25: Análisis de vulnerabilidades en el servidor gRPC.

Al servidor gRPC con la dirección IP *192.168.4.100* (se hizo un cambio de Raspberry con respecto al anterior experimento, debido a una falla) se le realizó un análisis para encontrar vulnerabilidades utilizando la herramienta Nessus Essentials (*figura 25*), en la cual solo se encontraron algunos puertos abiertos.

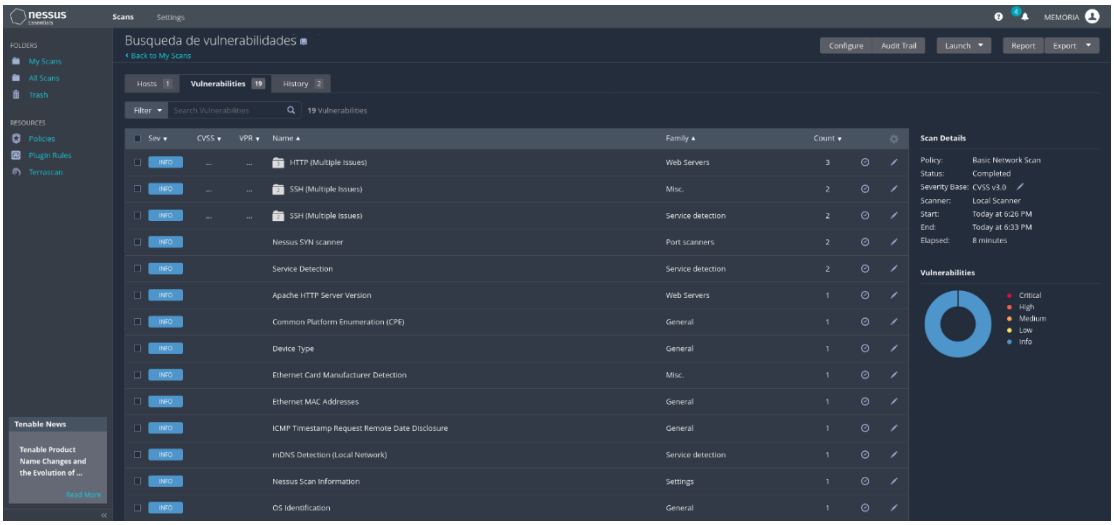


Figura 26: Listado de vulnerabilidades encontradas

Dentro de lo que se ve listado en la *figura 26*, lo relevante es la información relacionada SSH, mDNS y HTTP.

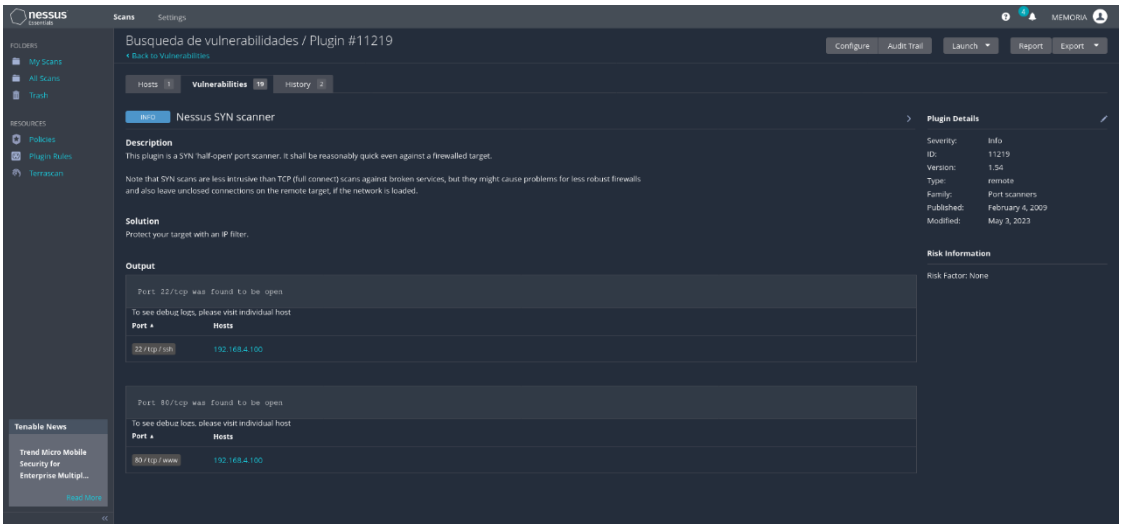


Figura 27: Puertos 22 y 80 abiertos

Se encontraba abierto el puerto 22, utilizado para conexiones SSH, y el puerto 80, conexiones HTTP (*figura 27*).

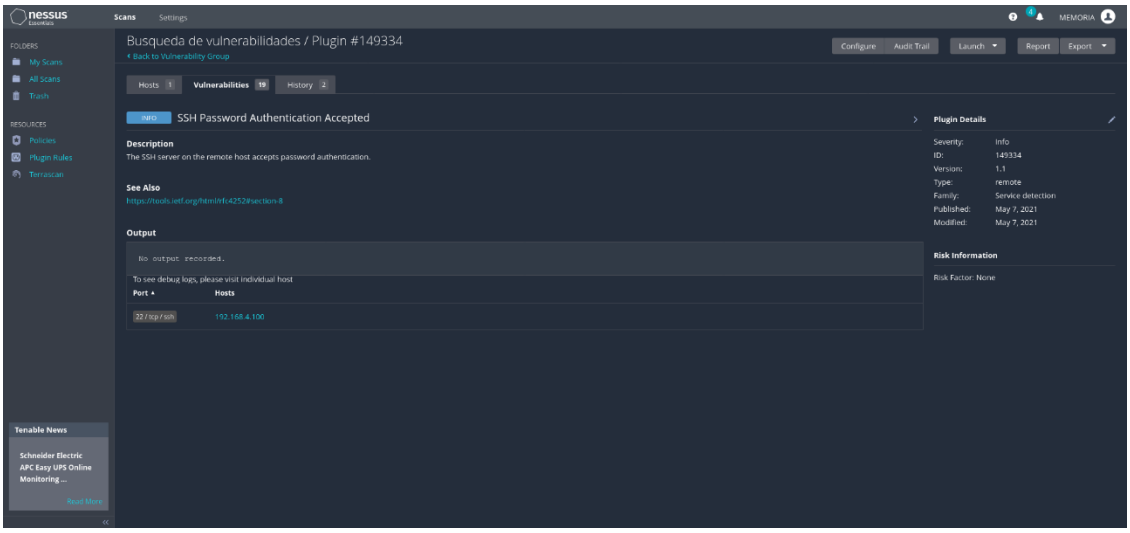


Figura 28: Información sobre SSH

También, se encontraba activa la autenticación por contraseña para el servidor SSH (*figura 28*).

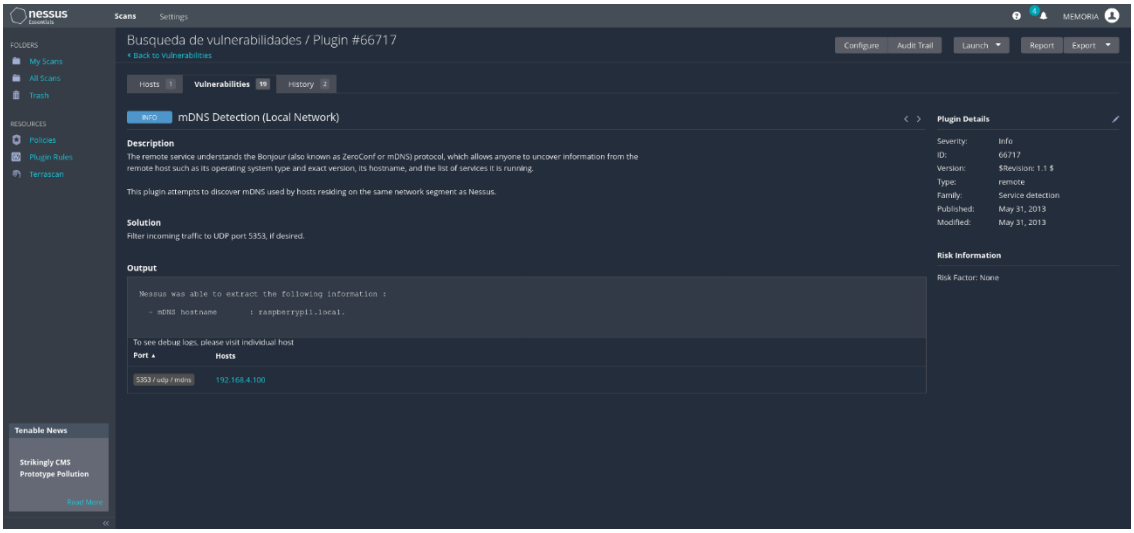


Figura 29: Información sobre mDNS

Por último, como se ve en la *figura 29*, se detectó que el puerto 5353 estaba abierto, el cual es utilizado para mDNS, y puede ser vulnerable a ataques [25].

Para corregir todo lo anteriormente detallado, se configuró el firewall en el servidor gRPC, utilizando *ufw* (herramienta en consola para la configuración de firewall en Linux), para dejar solamente disponibles los puertos utilizados por NAGIOS con NRPE (puerto 5666) y gRPC (50051). En la *figura 30* se puede ver esta configuración al consultar el estado del firewall.

```
pi1@raspberrypi1:~/Downloads/MonitoreoIoC-main $ sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
50051                      ALLOW       Anywhere
5666                       ALLOW       Anywhere
50051 (v6)                 ALLOW       Anywhere (v6)
5666 (v6)                  ALLOW       Anywhere (v6)
```

Figura 30: Configuración de firewall del servidor gRPC

De esta manera, al realizar nuevamente un análisis de vulnerabilidades con Nessus Essentials (*figura 31*), se confirma que se realizó correctamente el proceso de hardening en el sistema del servidor gRPC.

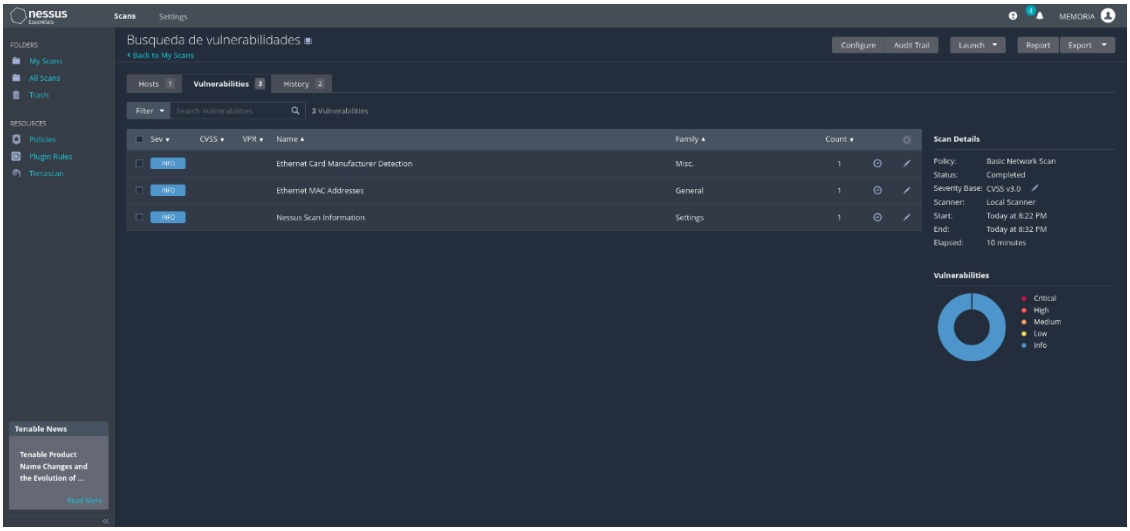


Figura 31: Listado de vulnerabilidades encontradas luego de configurar firewall

Para finalizar, utilizando la herramienta *Nmap* con el comando de la *figura 32* y con el objetivo de escanear todos los puertos del servidor gRPC, se obtuvo que solamente están disponibles los puertos 5666 (cerrado ya que al momento de ejecutar el análisis no se estaba utilizando) y el 50051, reafirmando que la configuración del firewall fue exitosa, tal como se ve en la *figura 33*.

```
memoria@memoria:~$ sudo nmap -p 1-65535 192.168.4.100
```

Figura 32: Comando para escanear todos los puertos del servidor gRPC

```
Nmap scan report for 192.168.4.100
Host is up (0.012s latency).
Not shown: 65533 filtered tcp ports (no-response)
PORT      STATE  SERVICE
5666/tcp  closed nrpe
50051/tcp open   unknown
MAC Address: E4:5F:01:37:50:E2 (Raspberry Pi Trading)

Nmap done: 1 IP address (1 host up) scanned in 958.26 seconds
```

Figura 33: Resultados del escaneo con Nmap

4.3 Pruebas de funcionamiento del prototipo

De acuerdo con el prototipo implementado, se realizaron pruebas para evidenciar su correcto funcionamiento. Se utilizó un *data set* de *malware* [26] para la detección de indicadores de compromiso en el host.

Al momento de iniciar el prototipo, de parte del cliente o host se solicita la contraseña para desencriptar el certificado de este, se inicia un análisis con LOKI y luego empieza la comunicación con el servidor por la verificación del hash del archivo *communicationClient.py*.

4.3.1 Detección, envío, almacenamiento y consulta de indicadores y reportes

1. **Si no coincide** con el que se encuentra almacenado en la base de datos, se levanta un indicador de compromiso en el servidor gRPC (*figura 36*), se le solicita reporte al host y luego se cierra la comunicación (*figura 34 y 35*), ya que no es confiable continuarla.

```
memoria@memoria:~/Documentos/GitHub/MonitoreoIoC$ sudo python communicationClient.py
Ingrese clave para desencriptar certificado: M0O8zY9aU6HoKx9BHWWaO3sFXgRtUNo7scny1HTYJjc=
message: "El archivo ha sido modificado"

Se ejecutara un analisis
nohup: se añade la salida a 'nohup.out'
Respuesta: Server solicita tu reporte
Respuesta: Cerrar conexion
Se cerrara la conexion
memoria@memoria:~/Documentos/GitHub/MonitoreoIoC$
```

Figura 34: Terminal del host cuando el hash no coindice

```
pi1@raspberrypi1:~/Downloads/MonitoreoIoC-main $ python communicationServer.py
Server Started
Se comprobara el hash del archivo client en el host 192.168.4.101
Solicitud de 192.168.4.101: No pasa nada
Se recibio el reporte de 192.168.4.101
Solicitud de 192.168.4.101: No pasa nada
```

Figura 35: Terminal del servidor gRPC cuando el hash no coincide

```
20  "2023-05-21 21:41:05.924633": {
21    "DESCRIPTION": "File modified",
22    "FILE": "communicationClient.py",
23    "ipHost": "192.168.4.101",
24    "detector": "MD5"
25  },
```

Figura 36: Indicador de compromiso levantado tras no coincidir el hashing realizado [27]

2. **Si coincide** con el que se encuentra almacenado en la base de datos, se continua la comunicación, y luego de 20 minutos se revisa el análisis en busca de indicadores de compromiso realizado por LOKI, y en caso de

detectar uno o más, se notifica al servidor gRPC, el cual responde con una solicitud de reporte al host.
Esto se puede ver en la *figura 37* donde, utilizando los archivos de un ransomware llamado Pay2Kitten pertenecientes al *dataset*, se detectaron indicadores de compromiso en el host luego de cumplir el tiempo de análisis con LOKI. Posterior a eso, los indicadores de compromiso fueron enviados al servidor gRPC, el cual respondió con una solicitud de reporte (*figura 38*).

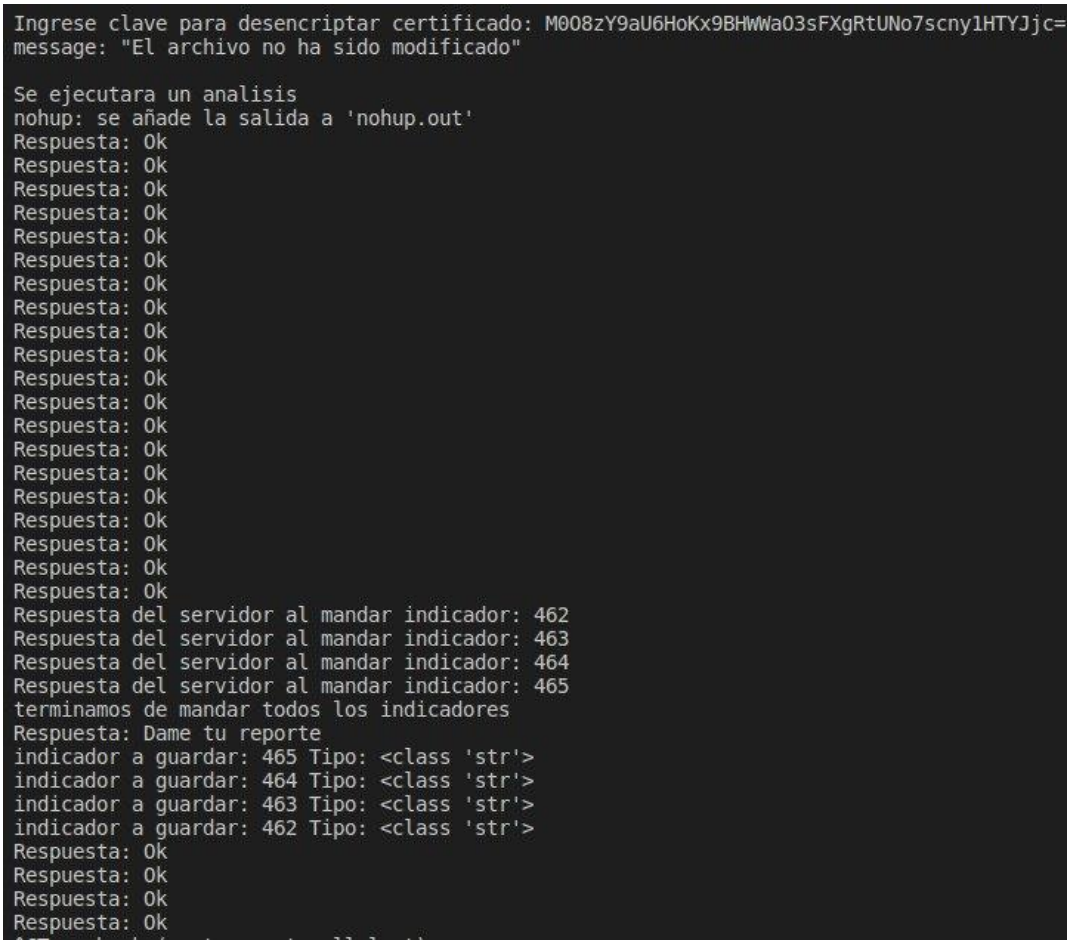


Figura 37: Terminal del host al detectar indicadores de compromiso

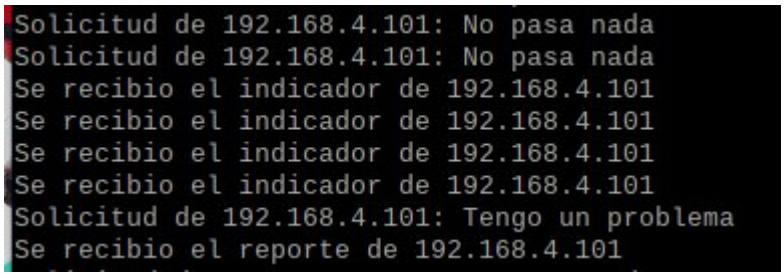


Figura 38: Terminal del servidor al recibir indicadores y solicitar reporte

Luego, para comprobar los indicadores recibidos, tras una consulta a la API, usando el script implementado para realizar pruebas (*figura 39*), se obtienen los 4 indicadores de compromiso que fueron detectados y enviados al servidor gRPC (*figura 40*), donde todos estos se encuentran vinculados a los archivos del ataque ransomware utilizado (*figura 41*).

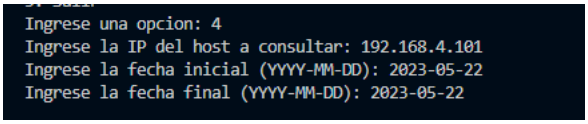


Figura 39: Consulta de indicadores a la API

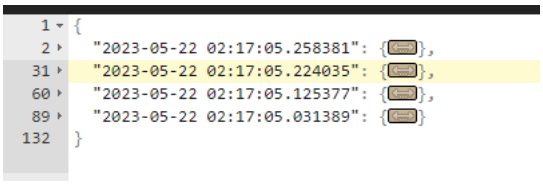


Figura 40: Resultado de la consulta

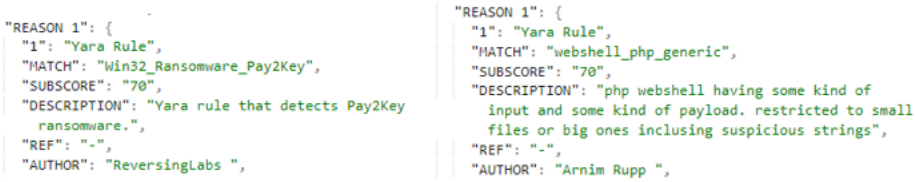


Figura 41: Indicadores de compromiso relacionados al ransomware

También, al consultar el reporte solicitado (*figura 42*), se obtiene lo que se muestra en las *figuras 43 y 44*.

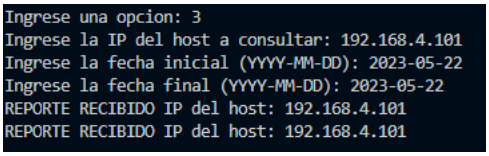


Figura 42: Consulta de reportes a la API

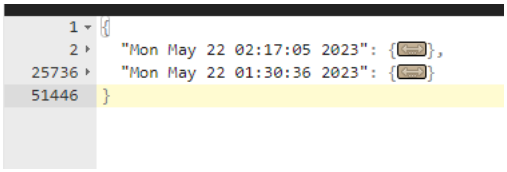


Figura 43: Resultado de la consulta

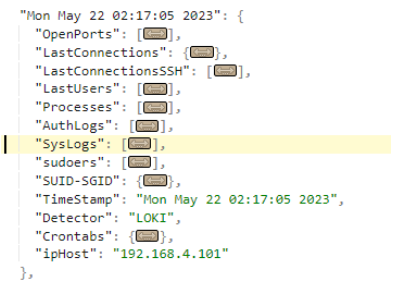


Figura 44: Reporte relacionado con los indicadores

4.3.2 Reporte global

Cada vez que se inicia el servidor gRPC, pasado 5 minutos se realiza una solicitud de reportes a todos los hosts que se encuentren conectados. Luego esto se repite cada cierto tiempo, que, para no generar un exceso de reportes quedo en 4 horas. Esto se puede ver en las *figuras 45 y 46*, donde el cliente, recibe en la sexta respuesta una solicitud de reporte desde el servidor, especificando que es el global.

```
memoria@memoria:~/Documentos/GitHub/MonitoreoIoC$ sudo python communicationClient.py
[sudo] contraseña para memoria:
Ingrese clave para desencriptar certificado: M0O8zY9aU6HoKx9BHWWaO3sFXgRtUNo7scny1HTYJjc=
message: "El archivo no ha sido modificado"

Se ejecutara un analisis
nohup: se añade la salida a 'nohup.out'
Respuesta: Ok
Respuesta: Ok
Respuesta: Ok
Respuesta: Ok
Respuesta: Ok
Respuesta: Solicitud de reporte global
Respuesta: Ok
```

Figura 45: Reporte global desde la terminal del host

```
pi1@raspberrypi1:~/Downloads/MonitoreoIoC-main $ sudo python3 communicationServer.py
Server Started
Se comprobara el hash del archivo client en el host 192.168.4.101
Solicitud de 192.168.4.101: No pasa nada
Solicitud de 192.168.4.101: No pasa nada
Solicitud de 192.168.4.101: No pasa nada
Solicitud de 192.168.4.101: No pasa nada
Solicitud de 192.168.4.101: No pasa nada
Solicitud de 192.168.4.101: No pasa nada
Se recibio el reporte de 192.168.4.101
Solicitud de 192.168.4.101: No pasa nada
```

Figura 46: Reporte global desde la terminal del servidor

Asimismo, si consultamos por el reporte global utilizando la API, se obtiene lo expuesto en las *figuras 47 y 48*, donde en la última se especifica que el motivo fue una solicitud global.

```
(base) C:\Users\FelipeHR\Documents\GitHub\MonitoreoIoC>python communicat
ionAPI.py
1. Recibir reportes
2. Recibir indicadores
3. Consulta especifica de reportes
4. Consulta especifica de indicadores
5. Salir
Ingrese una opcion: 3
Ingrese la IP del host a consultar: 192.168.4.101
Ingrese la fecha inicial (YYYY-MM-DD): 2023-05-28
Ingrese la fecha final (YYYY-MM-DD): 2023-05-28
REPORTE RECIBIDO IP del host: 192.168.4.101
REPORTE RECIBIDO IP del host: 192.168.4.101
```

Figura 47: Consulta del reporte global

```
{
  "Sun May 28 00:24:34 2023": {},
  "Sun May 28 00:09:32 2023": {
    "OpenPorts": [],
    "LastConnections": {},
    "LastConnectionsSSH": [],
    "LastUsers": [],
    "Processes": [],
    "AuthLogs": [],
    "SysLogs": [],
    "sudoers": [],
    "SUID-SGID": {},
    "TimeStamp": "Sun May 28 00:09:32 2023",
    "Detector": "GLOBAL",
    "Crontabs": {},
    "ipHost": "192.168.4.101"
  }
}
```


Figura 48: Reporte correspondiente a la solicitud global

4.3.3 Solicitud de reporte con NRPE y NAGIOS

Para probar NAGIOS con NRPE, se monitoreó el servicio de carga de CPU dentro del host, el cual al superar el 50% entraría a un estado CRITICAL (*figura 49*), teniendo como consecuencia la solicitud de reporte al host de parte del servidor gRPC, luego de ser alertado por NAGIOS.

```
command[check_load]=/usr/local/nagios/libexec/check_load -w 0.45,0.4,0.35 -c 0.7,0.6,0.5
```

Figura 49: Configuración del servicio de carga de CPU

De esta manera, al realizar una prueba de estrés al procesador, se consiguió pasar ese límite, obteniendo un estado CRITICAL en el servicio (*figura 50*).

Figura 50: Estado CRITICAL del servicio

Al alcanzar el estado CRITICAL, se ejecuta el script *communicationNagios.py* (*figura 51*), alertando al servidor gRPC sobre el estado del servicio (*figura 52*), el que a su vez responde pidiendole un reporte al host respectivo.

Figura 51: Ejecución del script al alcanzar estado CRITICAL

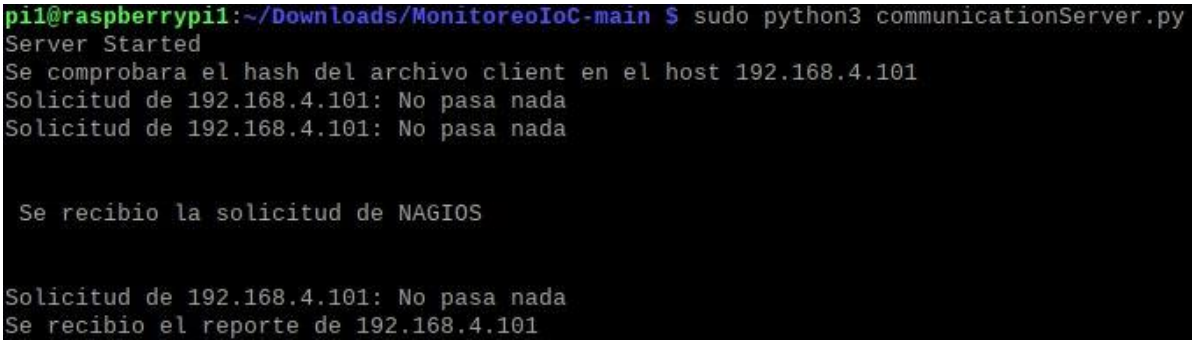


Figura 52: Alerta de NAGIOS recibida en el servidor gRPC y solicitud de reporte exitosa

Finalmente, para revisar el reporte que fue solicitado, se hizo una consulta a la API, teniendo como resultado lo que se muestra en la *figura 53*, donde se especifica que NAGIOS fue su detector, es decir, quien lo solicitó.

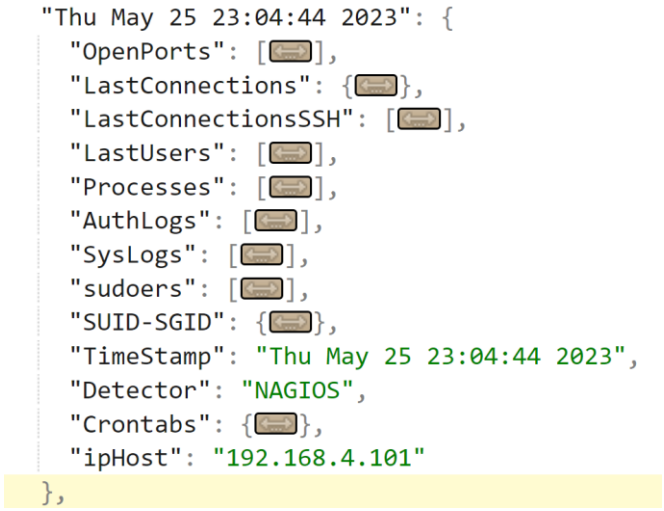


Figura 53: Reporte recibido por el servidor gRPC tras la alerta de NAGIOS

4.3.4 Recepción en tiempo real de indicadores y reportes

Dentro de los métodos RPC implementados se encontraban el de *StreamingServerReport* y *StreamingServerIndicator*, los cuales, a partir de una petición de reportes o indicadores de parte de una aplicación de terceros, los reenvían en tiempo real al ser recibidos en el servidor gRPC.

Para verificar el correcto funcionamiento de estos servicios, se utilizó el script *communicationAPI.py*, que anteriormente en este documento se usó para las consultas específicas de indicadores y reportes.

Se ejecutó el servidor gRPC (*figura 54*) y el host (*figura 55*), y tras 20 minutos de funcionamiento se le solicitaron 2 reportes al host y el servidor recibió 8 indicadores de compromiso.

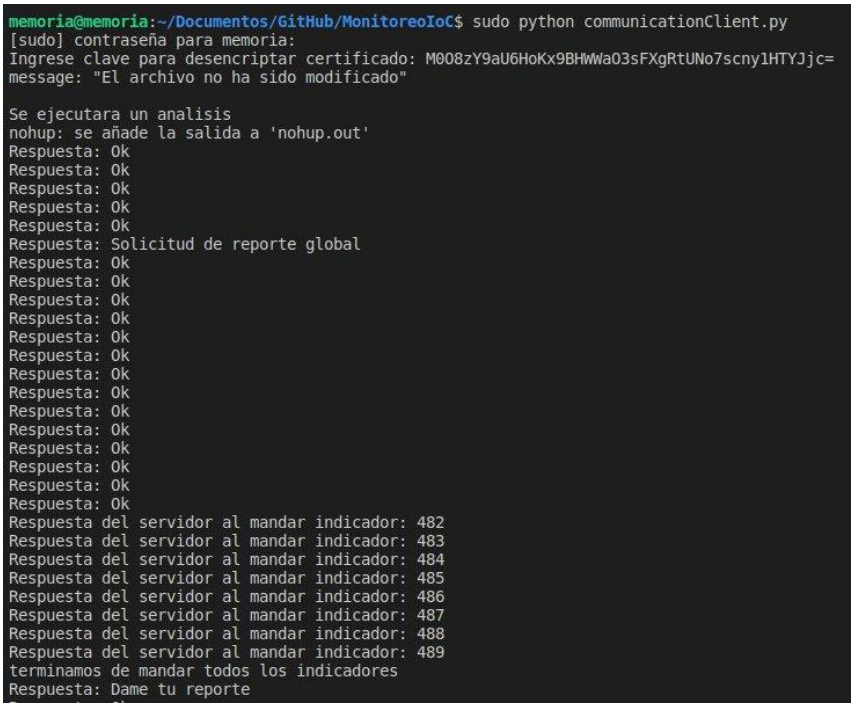


Figura 54: Ejecución host

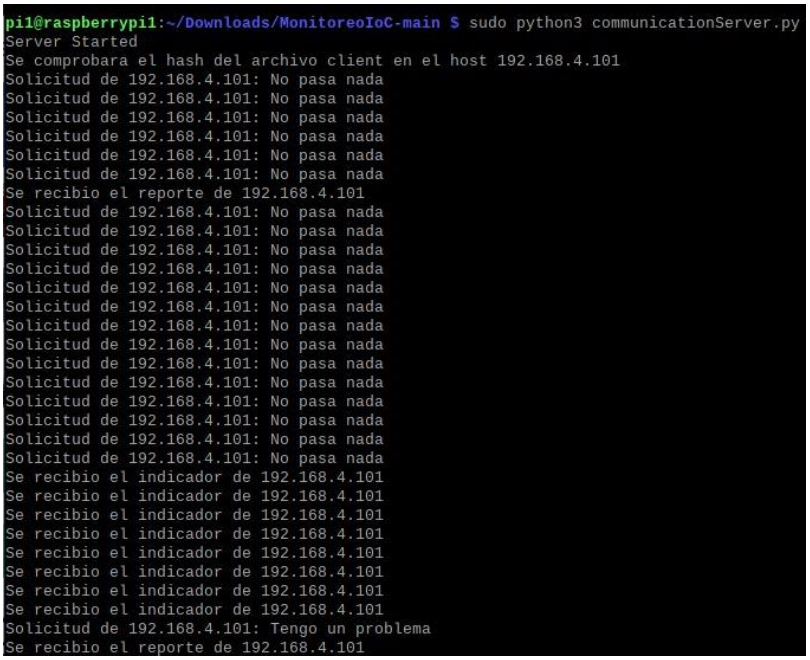


Figura 55: Ejecución servidor gRPC

Simultáneamente, en otro computador, se ejecutó el script para consultar a la API los indicadores y reportes en tiempo real.

Con respecto a los reportes, utilizando el método RPC *StreamingServerReport* en el script implementado (*figura 56*), se recibieron y almacenaron 2 reportes (*figura 57*), coincidiendo con la cantidad que solicitó el servidor gRPC al host.

```
(base) C:\Users\FelipeHR\Documents\GitHub\MonitoreoIoC>python communi
cationAPI.py
1. Recibir reportes
2. Recibir indicadores
3. Consulta especifica de reportes
4. Consulta especifica de indicadores
5. Salir
Ingrese una opcion: 1
REPORTE RECIBIDO
   IP del host: 192.168.4.101

REPORTE RECIBIDO
   IP del host: 192.168.4.101
```

Figura 56: Consulta de reportes en tiempo real

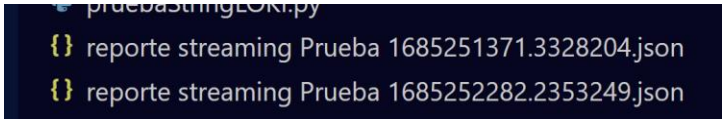


Figura 57: Almacenamiento correcto de los reportes recibidos

Ahora si comparamos los *TimeStamp* de los reportes solicitados por el servidor gRPC con los almacenados, tal como se muestra en la *figura 58*, vemos que la variación más alta es de 10 segundos.

Figura 58: Comparación de TimeStamp de los reportes [28]

Por otro lado, utilizando el método RPC *StreamingServerIndicator* en el script implementado (*figura 59*), se recibieron y almacenaron 8 indicadores (*figura 60*), coincidiendo también con la cantidad de indicadores enviados al servidor gRPC por el host.

```
(base) C:\Users\FelipeHR\Documents\GitHub\MonitoreoIoC>python communi
cationAPI.py
1. Recibir reportes
2. Recibir indicadores
3. Consulta especifica de reportes
4. Consulta especifica de indicadores
5. Salir
Ingrese una opcion: 2
INDICADOR DE COMPROMISO DETECTADO
   IP donde se detecto: 192.168.4.101
   Timestamp: 1685252271.6831
   Detector: LOKI

INDICADOR DE COMPROMISO DETECTADO
   IP donde se detecto: 192.168.4.101
   Timestamp: 1685252271.6427166
   Detector: LOKI

INDICADOR DE COMPROMISO DETECTADO
   IP donde se detecto: 192.168.4.101
   Timestamp: 1685252271.6083767
   Detector: LOKI
```

Figura 59: Consulta de indicadores en tiempo real

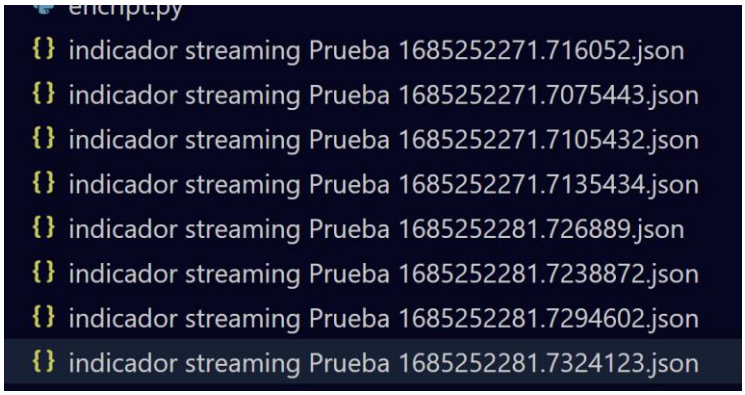


Figura 60: Almacenamiento correcto de los indicadores recibidos

Para finalizar, si comparamos los *TimeStamp* del primer y último indicador enviado por el host con los almacenados, tal como se muestra en la *figura 61*, vemos que la variación más alta es de 10 segundos.

Figura 61: Comparación de TimeStamp de los indicadores

4.4 Resumen de las pruebas realizadas

En la siguiente tabla se detallan las pruebas realizadas, con el resultado esperado y si es que fue logrado:

Tipo de prueba	Prueba realizada	Resultado esperado	¿Logrado?
Tríada CIA	Análisis de los paquetes trasmitidos por la plataforma usando Wireshark	Verificar la comunicación cifrada al utilizar SSL/TLS	**Si**
	Análisis de vulnerabilidades con Nessus al servidor gRPC	Disminución de la cantidad de vulnerabilidades encontradas	**Si**
Pruebas de funcionamiento del prototipo	Verificación del hash de un archivo enviado por el host al servidor	Solicitud de reporte, generación del indicador y cierre de conexión con el host	**Si**
	Detección y envío de indicadores de compromiso desde el host al servidor gRPC	Detección, envío y recepción del indicador en el servidor gRPC	**Si**
	Solicitud de reporte global	Petición de reporte al generar una solicitud de reporte global al host y recepción de este en el servidor gRPC	**Si**
	Solicitud de reporte al host	Petición de reporte al detectar compromiso en un host y recepción de este en el servidor gRPC	**Si**
	Solicitud de reporte con NAGIOS	Petición de reporte al detectar que un servicio entra en estado crítico al ser monitoreado con NAGIOS al host correspondiente y recepción de este en el servidor gRPC	**Si**
	Streaming de reportes hacia la aplicación de 3eros	Reenvío de los reportes hacia la aplicación de 3eros al llegar al servidor gRPC	**Si**
	Streaming de indicadores hacia la aplicación de 3eros	Reenvío de los indicadores hacia la aplicación de 3eros al llegar al servidor gRPC	**Si**
	Consulta especifica de reportes con la aplicación de terceros	Streaming de los reportes hacia la aplicación de terceros que coincidan con la consulta en el servidor gRPC	**Si**
	Consulta especifica de indicadores con la aplicación de terceros	Streaming de los reportes hacia la aplicación de terceros que coincidan con la consulta en el servidor gRPC	**Si**

Tabla 3: Resumen de las pruebas realizadas

5. Conclusiones

Para terminar, es importante mencionar que, si bien los indicadores de compromiso son una herramienta muy útil dentro de la ciberseguridad, no son preventivos, y tal como lo dice su nombre, indican que existe compromiso dentro del sistema, o, en otras palabras, el ataque ya ocurrió y dejó evidencia.

Teniendo lo anterior en cuenta, y que el 95% de las incidencias en ciberseguridad se deben a errores humanos [29], es vital dentro de una organización y/o empresa capacitar y concientizar periódicamente a sus empleados sobre buenas prácticas y prevención de ciber amenazas, como también la adopción de políticas de seguridad, siguiendo los estándares correspondientes.

Ahora, volviendo al contexto de lo diseñado e implementado, se pueden destacar los siguientes puntos:

- gRPC es un *framework* de comunicación muy útil para esta plataforma, debido a su seguridad (SSL/TLS), flexibilidad (integración aplicaciones de terceros que no estén implementadas en Python) y rendimiento (al utilizar HTTP/2).

- Utilizando reglas de YARA, y mediante la integración de aplicaciones de terceros a la plataforma, se podrían construir nuevos indicadores de compromiso, aumentando así la base de éstos que integraba LOKI.

- Usar las Raspberry Pi como servidores gRPC representa una ventaja, debido al bajo costo de ellas.

Para finalizar, se destaca que todo el trabajo realizado se encuentra en un repositorio de GitHub [30].

6. Trabajo Futuro

Tras una exitosa prueba de concepto de la arquitectura general lograda con el prototipo implementado, queda mucho trabajo por delante, que se puede resumir en lo que se detalla a continuación.

6.1 Integración del concepto de disponibilidad

Como se discutió anteriormente, la disponibilidad fue el concepto que quedó fuera de la implementación del prototipo, sin embargo, durante la primera etapa del proyecto, se realizó una investigación sobre como integrarla, y se ve reflejada en la arquitectura general del sistema.

Consta de tener 3 Raspberry PI funcionando como servidores gRPC, dándole al sistema la facultad de seguir funcionando en caso de que uno de ellos fallara. Lo último sería gracias a una comunicación constante mediante gRPC entre los servidores, permitiéndoles saber la disponibilidad de cada uno.

Por otro lado, dentro de cada uno de los servidores gRPC, se encontraría una copia exacta de la base de datos del servidor gRPC que esté recibiendo la información. Esto es posible debido a que todo lo almacenado se va replicando en las demás bases de datos en tiempo real [31].

6.2 Integración de aplicaciones de terceros

En el prototipo implementado fue desarrollada una API para aplicaciones de terceros, por la cual se facilita la información de los reportes recibidos o indicadores detectados. El motivo de esto fue principalmente añadir más herramientas a la plataforma que ayuden a aumentar la cantidad de indicadores de compromiso que pueden ser detectados.

Por ejemplo, una aplicación que analice los reportes almacenados en la base de datos, encontrando indicadores de compromiso que no han sido detectados, o herramientas que pueden caracterizar nuevos indicadores de compromiso, como lo podría hacer RABS, ya mencionada anteriormente, beneficiarían mucho a la plataforma.

De igual forma, esta API se utilizaría con un panel táctico, el cual despliega en tiempo real el estado de seguridad de los hosts en la red y los problemas que van surgiendo, que fue desarrollado por Sergio Cifuentes en la implementación de su prototipo.

Sumando R-ABS y el panel táctico, la arquitectura general de la plataforma quedaría especificada en la *figura 62*.

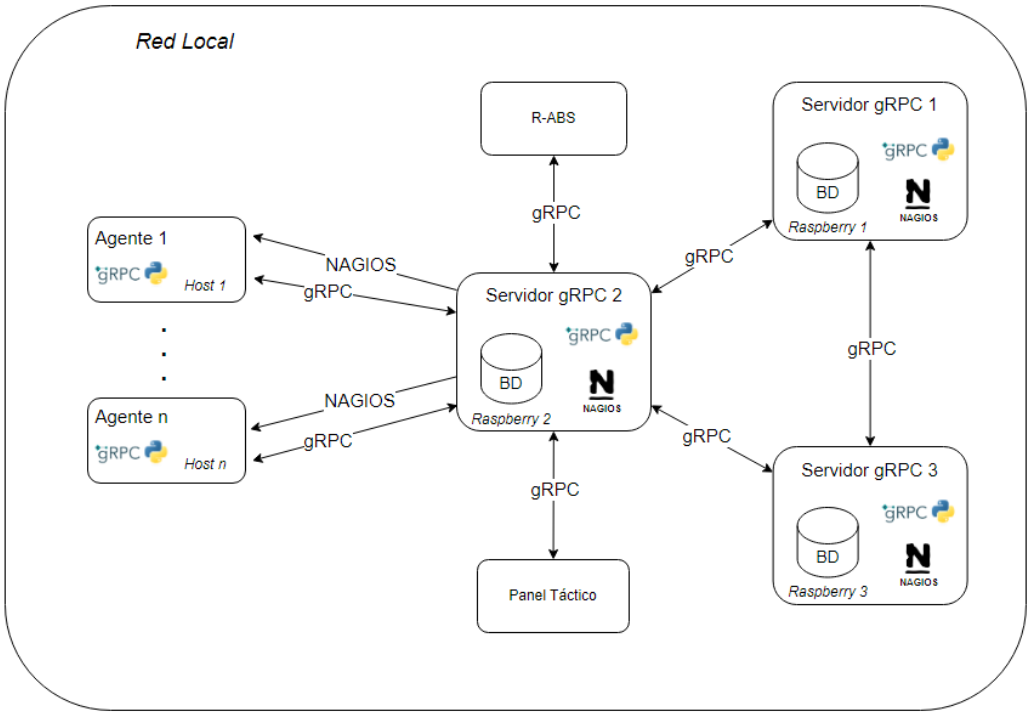


Figura 62: Arquitectura general integrando R-ABS y panel táctico

6.3 Detección de Indicadores de compromiso con NAGIOS

Mediante la utilización de NAGIOS con NRPE [32], en el futuro se podrían crear reglas para detectar indicadores de compromiso vinculados a los servicios que tienen los servidores.

De igual manera, mediante el protocolo SNMP se puede extraer información de dispositivos en red que sean compatibles con él, por ejemplo, routers y switches.

6.4 Otros

- **Integración del sistema operativo Windows:** Durante la implementación del prototipo, solo se consideraron hosts con sistemas operativos Linux (probado en Debian con las Raspberry PI, y Ubuntu con el PC de escritorio). Sin embargo, integrar hosts con Windows a la arquitectura no sería muy complicado, y solo requeriría de la recolección de información para el reporte en este sistema operativo.

- **Rediseño de la arquitectura:** Por otro lado, se quiere realizar un rediseño de la arquitectura presentada anteriormente, bajo un enfoque *Secure by Desing* [33], el cual diseña pensando en la seguridad. En resumen, trata de integrar seguridad durante todas las etapas del desarrollo, para no dejar la seguridad como un aspecto secundario.

7. Referencias

[1] Check Point Research reports a 38% increase in 2022 global cyberattacks. (2023, 5 de enero). Check Point Blog; Check Point Software. https://blog.checkpoint.com/2023/01/05/38-increase-in-2022-global-cyberattacks/

[2] WeLiveSecurity. (2021, 22 de febrero). ¿Qué son los indicadores de compromiso? Y qué evidencia puedes tener de que has sido víctima de malware. https://www.welivesecurity.com/la-es/2021/02/22/que-son-indicadores-compromiso-evidencia-puedes-haber-sido-victima-malware/

[3] National Institute of Standards and Technology (NIST). (2014). Framework for Improving Critical Infrastructure Cybersecurity. https://www.nist.gov/system/files/documents/cyberframework/cybersecurity-framework-021214.pdf

[4] Instituto Nacional de Ciberseguridad (INCIBE). (2018, 28 de junio). El valor de los indicadores de compromiso en la industria. Blog de INCIBE-CERT. https://www.incibe-cert.es/blog/el-valor-los-indicadores-compromiso-industria

[5] Interbel. La Triada del Cid. https://www.interbel.es/triada-cid/

[6] Cisco. (2019). Cybersecurity series 2019: Threat hunting. https://www.cisco.com/c/dam/global/es_mx/solutions/security/pdf/cybersecurity-series-2019-threat-hunting.pdf

[7] IETF. (2018). Indicators of Compromise (IoCs) and Their Role in Attack Defence. https://www.ietf.org/archive/id/draft-ietf-opsec-indicators-of-compromise-02.html

[8] CSIRT. (2020). AN2-2020-12: Vulnerabilidades en VPNs SSL/TLS y cómo mitigarlas. https://www.csirt.gob.cl/media/2020/07/AN2-2020-12.pdf

[9] Brock, B. (2021, 27 de enero). How to Determine if Linux is Compromised. Linux Hint. https://linuxhint.com/determine_if_linux_is_compromised/

[10] Python Software Foundation. Python. https://www.python.org/

[11] Noelia Martín. (2022). gRPC, ¿qué es y cómo funciona? Paradigmadigital.com. https://www.paradigmadigital.com/dev/grpc-que-es-como-funciona/

[12] Chiarelli, A. (2019, 11 de diciembre). How to use gRPC to build efficient .NET core 3.1 microservices. Auth0 - Blog. https://auth0.com/blog/implementing-microservices-grpc-dotnet-core-3/

[13] Know How. (2020). gRPC: pionero de la comunicación cliente-servidor del futuro. Ionos. https://www.ionos.es/digitalguide/servidores/know-how/que-es-grpc/

[14] Nagios Enterprises LLC. Nagios. https://www.nagios.org/

[15] KeepCoding. (2023). ¿Qué son las reglas YARA? KeepCoding Blog. https://keepcoding.io/blog/que-son-las-reglas-yara/

[16] Neo23x0. (2021). Loki. GitHub. https://github.com/Neo23x0/Loki

[17] THOR lite - nextron systems. (2018, 18 de mayo). Nextron-systems.com; Nextron Systems GmbH. https://www.nextron-systems.com/thor-lite/

[18] Cloudflare. (2021). cfssl: CFSSL - Cloudflare's PKI toolkit. GitHub. https://github.com/cloudflare/cfssl

[19] SQLite. SQLite Home Page. https://sqlite.org/index.html

[20] Wireshark. Wireshark · Go Deep. https://www.wireshark.org/

[21] Tenable. Nessus Essentials. https://es-la.tenable.com/products/nessus/nessus-essentials
[22] Galstad, E. NRPE - Nagios remote plugin executor - Nagios Exchange. Nagios.org. https://exchange.nagios.org/directory/Addons/Monitoring-Agents/NRPE--2D-Nagios-Remote-Plugin-Executor/details
[23] National Institute of Standards and Technology (NIST). (2019). Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar2.pdf
[24] Cryptography.io. Fernet — Cryptography 3.4.8 documentation. https://cryptography.io/en/latest/fernet/
[25] HackTricks. UDP Multicast DNS (mDNS). https://book.hacktricks.xyz/network-services-pentesting/5353-udp-multicast-dns-mdns
[26] MalwareSamples. (2021). Malware-Feed. Github. https://github.com/MalwareSamples/Malware-Feed
[27] JSON Online. JSON Viewer. https://jsononline.net/json-viewer
[28] Epoch Converter. https://www.epochconverter.com
[29] Micke Ahola. (2021). The Role of Human Error in Successful Cyber Security Breaches. uSecure Blog. https://blog.usecure.io/the-role-of-human-error-in-successful-cyber-security-breaches
[30] Felipe Henriquez y Sergio Cifuentes. (2023). MonitoreoIoC. GitHub. https://github.com/FelipeHR/MonitoreoIoC
[31] Smith, J. (2018). High Availability MySQL Cluster with Load Balancing using HAProxy and Heartbeat. Towards Data Science. https://towardsdatascience.com/high-availability-mysql-cluster-with-load-balancing-using-haproxy-and-heartbeat-40a16e134691
[32] Antonios S. Andreatos & Nikolaos Chatzipantou. (2021). Using Nagios on a Raspberry Pi to Monitor Anetwork with Emphasis on Security. ResearchGate. https://www.researchgate.net/publication/348895200_Using_Nagios_on_a_Raspberry_Pi_to_Monitor_Anetwork_with_Emphasis_on_Security
[33] Laotshi. (2020). Secure by Design | Fundamentos. Medium. https://laotshi.medium.com/secure-by-design-fundamentos-b4d25ed3a9ac