Universitat Oberta de Catalunya

uoc.edu

Estudio e implementación de un Firewall DNS

Autor: Adrián Navas Ajenjo
Máster Universitario en Ciberseguridad y Privacidad
TFM Seguridad Empresarial

Tutor: Borja Guaita Pérez
Profesor responsable de la asignatura: Víctor García Font

26/05/2022

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Estudio e implementación de un Firewall DNS*
Nombre del autor:	*Adrián Navas Ajenjo*
Nombre del consultor/a:	*Borja Guaita Pérez*
Nombre del PRA:	*Víctor García Font*
Fecha de entrega:	05/2022
Titulación:	*Máster Universitario en Ciberseguridad y Privacidad*
Área del Trabajo Final:	*TFM Seguridad Empresarial*
Idioma del trabajo:	*Castellano*
Palabras clave	*Ciberseguridad, Firewall, OSINT, Phishing, DNS, Cloud*

Resumen del Trabajo

El creciente uso de las Tecnologías de la Información, y específicamente de la navegación web, conlleva también un incremento en el número de criminales que tratan de obtener beneficio ilegal a costa de los usuarios.
Con el fin de frenar este tipo de ataques, se ponen muchos esfuerzos en la implementación de productos de seguridad como firewalls o programas antivirus, pero, sin embargo, el protocolo DNS es comúnmente dejado de lado.
Debido a que este protocolo se utiliza en prácticamente todas las interacciones de la navegación web, se propone el estudio e implementación de un firewall DNS. Mediante el uso de información de IPs y nombres de dominio maliciosos extraídos de fuentes públicas, se filtran las peticiones maliciosas a un servidor web que alerta de este hecho.

Inicialmente, se realiza un estudio del estado del arte, así como de las tecnologías implicadas. Dentro de esta fase se realiza un análisis de viabilidad de implementación de un motor de aprendizaje automático para detectar posibles ataques de *phishing*.

Adicionalmente se propone en este estudio una arquitectura en alta disponibilidad y con una plataforma de recogida y visualización de métricas, con aplicación a entornos empresariales y domésticos y que aportan información relevante para el estudio futuro de las métricas.

Tras las pruebas realizadas sobre el producto se concluye la viabilidad y utilidad de la implementación de un firewall DNS como una herramienta adicional de protección frente a posibles ataques.

Abstract

The increasing use of Information Technologies, and more specifically of the web navigation, leads also to an increment of the number of criminals that try to get illegal profit from these users.
To try to stop this kind of attackers, a lot of efforts are made by the implementation of cybersecurity products as firewalls of antivirus software. On the other hand, the DNS protocol is commonly excluded on the scope of these products.
Due to the highly use of this protocol in almost every web navigation interaction, it is proposed a study and implementation of a DNS firewall. By using malicious IP and domain open-source information, all the malicious requests are filtered out and redirected to an alert web server alerting about this behavior.

First, a state of art and related technologies studies are made. Inside this phase, the implementation viability of a Machine Learning engine to detect phishing attacks is studied.

It is proposed in this study a highly available architecture with a metrics recollection and visualization platform applicable to enterprise or homes and that provides relevant information for the future study of the metrics.

After all the tests done for the final product, it is concluded the viability and usefulness of the implementation of a DNS firewall as an additional tool to protect users against cyberattacks.

Índice

Lista de figuras

Lista de tablas

1. Introducción

1.1 Contexto y justificación del Trabajo

Con el creciente uso de las Tecnologías de la Información, la navegación web se ha convertido en uno de los puntos más utilizados por clientes tanto de escritorio como móviles. Junto con este incremento de uso, crece también el número de criminales que intentan sacar beneficio ilegalmente a costa de estos usuarios mediante diferentes vectores.

Uno de los vectores principales de ataque es a través del uso de la Ingeniería Social. Estos ataques se valen del engaño a las personas para que realicen acciones que no desean como entrar en un enlace que simula una web del banco o conseguir que inserten un dispositivo USB malicioso en su ordenador. Una de las vías principales de este tipo de ataques es el "*phishing*", mediante el cual se envían correos electrónicos que simulan ser entidades conocidas, como bancos o la administración pública, y que llevan al usuario mediante la manipulación a entrar en enlaces a páginas web que simulan ser verídicas y, de esta forma, robarles las credenciales de acceso a un portal web, sus datos bancarios o cualquier otro tipo de información que pueda tener valor o conseguir que se descargue y ejecute software malicioso en los dispositivos de los afectados.

El software malicioso que se introduce puede tener muy variadas funcionalidades y fines, pero cada vez es más frecuente, sobre todo en el ámbito empresarial, que estos sean cripto mineros o "*ransomware"*.

Adicionalmente, los mecanismos de publicidad embebida son también cada vez más frecuentes e intrusivos tanto en la navegación web como en el uso de aplicaciones de escritorio y móviles. Estos mecanismos pueden comprometer la privacidad de los usuarios mediante la recogida encubierta de datos personales o disminuir la calidad de la navegación mediante la inserción intrusiva de anuncios publicitarios.

Todos estos elementos que comprometen la seguridad y la privacidad de los usuarios se suelen tratar de evitar mediante el uso de programas que controlan las acciones de los usuarios y el contenido de ficheros o documentos que se ejecutan o descargan en los dispositivos como, por ejemplo, programas antivirus o bloqueadores de anuncios en navegadores web. Sin embargo, hay otro enfoque menos utilizado, y que puede suponer una mejora complementaria en materia de seguridad y privacidad, que es el uso de un *Firewall DNS*. Este elemento puede utilizarse para, mediante el uso de fuentes públicas de direcciones IP y dominios maliciosos, detectar cuándo un cliente está tratando de acceder a un sitio que puede comprometer su seguridad o privacidad y, o bien bloquear o bien alertarle de esta acción.

El uso de un *Firewall DNS* en este ámbito, siempre de forma adicional a los mecanismos más comunes de seguridad comentados previamente, tiene bastante sentido y utilidad en los casos de ataques comentados previamente ya que todos tienen en común el uso de URLs que necesitan ser resueltas para que los ataques se puedan realizar.

Otro de los problemas principales que se afrontan en materia de seguridad es el de la visualización. Es de vital importancia disponer de los datos correctos y poder visualizarlos de una forma eficaz. En el caso competente a este trabajo, los datos que se pueden recoger, analizar y visualizar son los datos de

peticiones de resolución de dominios a sitios maliciosos y benignos para obtener estadísticas de ellos.
Por ello, en este trabajo se propone el estudio y la implementación de un *Firewall DNS* que utilice fuentes públicas de direcciones IP y dominios maliciosos para detectar y bloquear o alertar a los clientes cuando estos intenten acceder a dominios que puedan estar relacionados con ataques a la seguridad o privacidad.
Se propone adicionalmente la implementación de una plataforma de recogida de métricas que sirva para la visualización y análisis de datos posterior de todos los eventos que suceden en el servidor DNS.

1.2 Objetivos del Trabajo

El objetivo principal de este trabajo es el de estudiar y desarrollar una implementación de un *Firewall DNS* funcional y puesto a disposición pública para su uso que disponga de un elemento de recogida de datos y visualización de estos. Este *Firewall DNS* se desarrollará mediante una implementación de un servidor DNS que consulte fuentes públicas de direcciones IP y dominios maliciosos para alertar cuando los clientes realicen peticiones que apunten a estos.
Si bien estos son los objetivos principales, el trabajo se puede dividir en varios objetivos más acotados:

- Estudio del estado del arte.
- Estudio del protocolo DNS y su implementación en un servidor.
- Estudio de diferentes fuentes públicas de direcciones IP y dominios maliciosos.
- Estudio y elección de una solución de almacenamiento eficiente para este contexto.
- Estudio y elección de una solución visualización eficiente para este contexto.
- Estudio de métricas y visualizaciones útiles en este contexto.
- Estudio de la viabilidad de implementar una prueba de concepto para introducir un modelo de aprendizaje automático que filtre prediga URLs maliciosas basándose en la extracción de parámetros de estas.
- Diseño de la solución para alertar de peticiones a URLs maliciosas.
- Diseño del producto final (servidor DNS, servidor web de redirección y plataforma de recogida de métricas y visualización).
- Implementación de la plataforma y disposición pública de la misma.
- Realización de pruebas de la plataforma.

Como objetivo adicional, se ha propuesto realizar este trabajo mediante el uso de tecnologías *open source* para que sea exportable e implementable por terceras personas de forma sencilla y gratuita.
Como objetivo final, este trabajo se desarrollará buscando mejorar la seguridad y la privacidad de los usuarios, tanto a nivel personal como a nivel empresarial, mediante un enfoque centrado en la resolución de nombres.

1.3 Enfoque y método seguido

El trabajo se divide en tres fases diferentes: Estudio, diseño e implementación.
En la fase de estudio se realiza un análisis de todos los elementos necesarios para el diseño y la implementación mediante la consulta de artículos científicos, material bibliográfico, blogs comunitarios de tecnología y la propia documentación oficial de los productos estudiados.
Una vez realizado el estudio previo y recopilada toda la información necesaria, se procede a realizar el diseño a nivel lógico para disponer de los diferentes componentes lógicos que comprenderán el producto final, así como el diseño a nivel de arquitectura para la infraestructura que soporta este producto. Este diseño se realiza siguiendo un patrón de alta disponibilidad y siguiendo buenas prácticas de arquitectura para tratar de realizarlo de la forma más resiliente posible.
Finalmente, una vez el diseño está finalizado, se procede a realizar la implementación de la infraestructura y del producto a nivel lógico siguiendo este diseño. Al disponer de todo el producto implementado, se realizan pruebas para verificar que el diseño y los objetivos se cumplen.
Tras estas fases, se documentará todo el proceso, investigaciones, implementación, pruebas y conclusiones obtenidas en la memoria del trabajo.
Tal y como se puede observar en el apartado 1.1 Contexto y justificación del Trabajo, las tareas se realizarán en el orden dispuesto en este apartado, pero siempre tratando de paralelizar el mayor número de tareas posible con el fin de agilizar el desarrollo de todas estas y disponer de un margen necesario para realizar pruebas y solucionar los posibles problemas o mejoras que se encuentren.

1.4 Planificación del Trabajo

1.4.1 Listado de tareas

El listado detallado de tareas para la resolución de los objetivos definidos previamente es el siguiente. En este listado se pueden ver sombreados en amarillo los objetivos y sombreadas en gris las tareas correspondientes a cada uno de los objetivos. También se pueden observar sombreadas en gris y en negrita algunas tareas que no entran dentro de ningún objetivo en concreto pero que sí tienen relevancia para el desarrollo del trabajo.

Desarrollo del Plan de Trabajo
Estudio del protocolo DNS y su implementación
Estudio general de servidores DNS y su protocolo
Estudio de la implementación de un servidor DNS mediante Python.
Estudio de fuentes públicas de direcciones IP y dominios maliciosos
Estudio de fuentes públicas y sus periodos de actualización
Automatización de la extracción recurrente de los datos de las fuentes públicas.
Elección de una solución de almacenamiento y una solución visualización eficientes
Estudio de soluciones de almacenamiento para los datos obtenidos de fuentes públicas
Estudio de soluciones de almacenamiento de métricas

Estudio de soluciones de visualización de métricas
Estudio de métricas y visualizaciones útiles en este contexto
Estudio de métricas para servidores DNS, de rendimiento y de seguridad
Estudio de la viabilidad de implementar una prueba de concepto para introducir un modelo de aprendizaje automático que filtre prediga *URLs* maliciosas basándose en la extracción de parámetros de estas
Estudio de patrones y parámetros extraíbles de las URL
Estudio de la viabilidad de implementar esta funcionalidad en el servidor DNS
Diseño de la solución para alertar de peticiones a URLs maliciosas
Estudio de la integración en el servidor DNS de detección de URLs maliciosas
Estudio de la implementación de un servidor web al que redirigir las peticiones a sitios maliciosos
Diseño del producto final (servidor DNS, servidor web de redirección y plataforma de recogida de métricas y visualización)
Diseño lógico
Selección de la plataforma de despliegue
Diseño de la arquitectura
Diseño en alta disponibilidad y con buenas prácticas de seguridad
Implementación de la plataforma y disposición pública de la misma
Implementación del servidor DNS
Integración de los datos extraídos de las fuentes públicas de direcciones IP y dominios maliciosos
Implementación del motor del firewall DNS
Implementación del servidor web de redirección para alertar de accesos maliciosos
Implementación de la plataforma de métricas
Implementación de las visualizaciones
Apertura al público de la plataforma de DNS
Realización de pruebas de la plataforma
Realización de pruebas de uso generales
Realización de pruebas de rendimiento
Realización de pruebas de alta disponibilidad
Desarrollo de la memoria del trabajo

Tabla 1. Listado de tareas.

1.4.2 Planificación temporal

Teniendo en cuenta el listado de tareas del apartado anterior, la planificación temporal estimada es la siguiente:

Fase	Tarea	Tiempo	Fecha inicio	Fecha fin
Inicio	**Desarrollo del Plan de Trabajo**	**13**	**16/02/2022**	**01/03/2022**
Estudio	**Estudio del protocolo DNS y su implementación**	**10**	**25/02/2022**	**07/03/2022**
	Estudio general de servidores DNS y su protocolo	3	25/02/2022	28/02/2022
	Estudio de la implementación de un servidor DNS mediante Python.	7	28/02/2022	07/03/2022

	Estudio de fuentes públicas de direcciones IP y dominios maliciosos	**8**	**07/03/2022**	**15/03/2022**
	Estudio de fuentes públicas y sus periodos de actualización	4	07/03/2022	11/03/2022
	Automatización de la extracción recurrente de los datos de las fuentes públicas.	4	11/03/2022	15/03/2022
	Elección de una solución de almacenamiento y una solución visualización eficientes	**4**	**15/03/2022**	**19/03/2022**
	Estudio de soluciones de almacenamiento para los datos obtenidos de fuentes públicas	2	15/03/2022	17/03/2022
	Estudio de soluciones de almacenamiento de métricas	2	15/03/2022	17/03/2022
	Estudio de soluciones de visualización de métricas	2	17/03/2022	19/03/2022
	Estudio de métricas y visualizaciones útiles en este contexto	**3**	**19/03/2022**	**22/03/2022**
	Estudio de métricas para servidores DNS, de rendimiento y de seguridad	3	19/03/2022	22/03/2022
	Estudio de la viabilidad de implementar una prueba de concepto para introducir un modelo de aprendizaje automático que filtre y prediga *URLs* maliciosas basándose en la extracción de parámetros de estas	**10**	**22/03/2022**	**01/04/2022**
	Estudio de patrones y parámetros extraíbles de las URL	5	22/03/2022	27/03/2022
	Estudio de la viabilidad de implementar esta funcionalidad en el servidor DNS	5	27/03/2022	01/04/2022
	Diseño de la solución para alertar de peticiones a URLs maliciosas	**7**	**01/04/2022**	**08/04/2022**
	Estudio de la integración en el servidor DNS de detección de URLs maliciosas	5	01/04/2022	06/04/2022
	Estudio de la implementación de un servidor web al que redirigir las peticiones a sitios maliciosos	5	03/04/2022	08/04/2022
	Diseño del producto final (servidor DNS, servidor web de redirección y plataforma de recogida de métricas y visualización)	**12**	**08/04/2022**	**20/04/2022**
	Diseño lógico	5	08/04/2022	13/04/2022
	Selección de la plataforma de despliegue	1	08/04/2022	09/04/2022
	Diseño de la arquitectura	10	08/04/2022	18/04/2022
Diseño	Diseño en alta disponibilidad y con buenas prácticas de seguridad	12	08/04/2022	20/04/2022
	Implementación de la plataforma y disposición pública de la misma	**23**	**20/04/2022**	**13/05/2022**
	Implementación del servidor DNS	3	20/04/2022	23/04/2022
	Integración de los datos extraídos de las fuentes públicas de direcciones IP y dominios maliciosos	3	23/04/2022	26/04/2022
Implementación	Implementación del motor del firewall DNS	4	26/04/2022	30/04/2022

	Implementación del servidor web de redirección para alertar de accesos maliciosos	4	30/04/2022	04/05/2022
	Implementación de la plataforma de métricas	5	04/05/2022	09/05/2022
	Implementación de las visualizaciones	3	09/05/2022	12/05/2022
	Apertura al público de la plataforma de DNS	1	12/05/2022	13/05/2022
	Realización de pruebas de la plataforma	**3**	**13/05/2022**	**16/05/2022**
	Realización de pruebas de uso generales	2	13/05/2022	15/05/2022
	Realización de pruebas de rendimiento	2	13/05/2022	15/05/2022
Pruebas	Realización de pruebas de alta disponibilidad	3	13/05/2022	16/05/2022
Documentación	**Desarrollo de la memoria del trabajo**	**91**	**01/03/2022**	**31/05/2022**

Tabla 2. Listado de tareas con tiempos estimados.

De esta tabla, se ha extraído el correspondiente diagrama de Gantt que se puede observar en la **Figura 1** que se muestra a continuación.

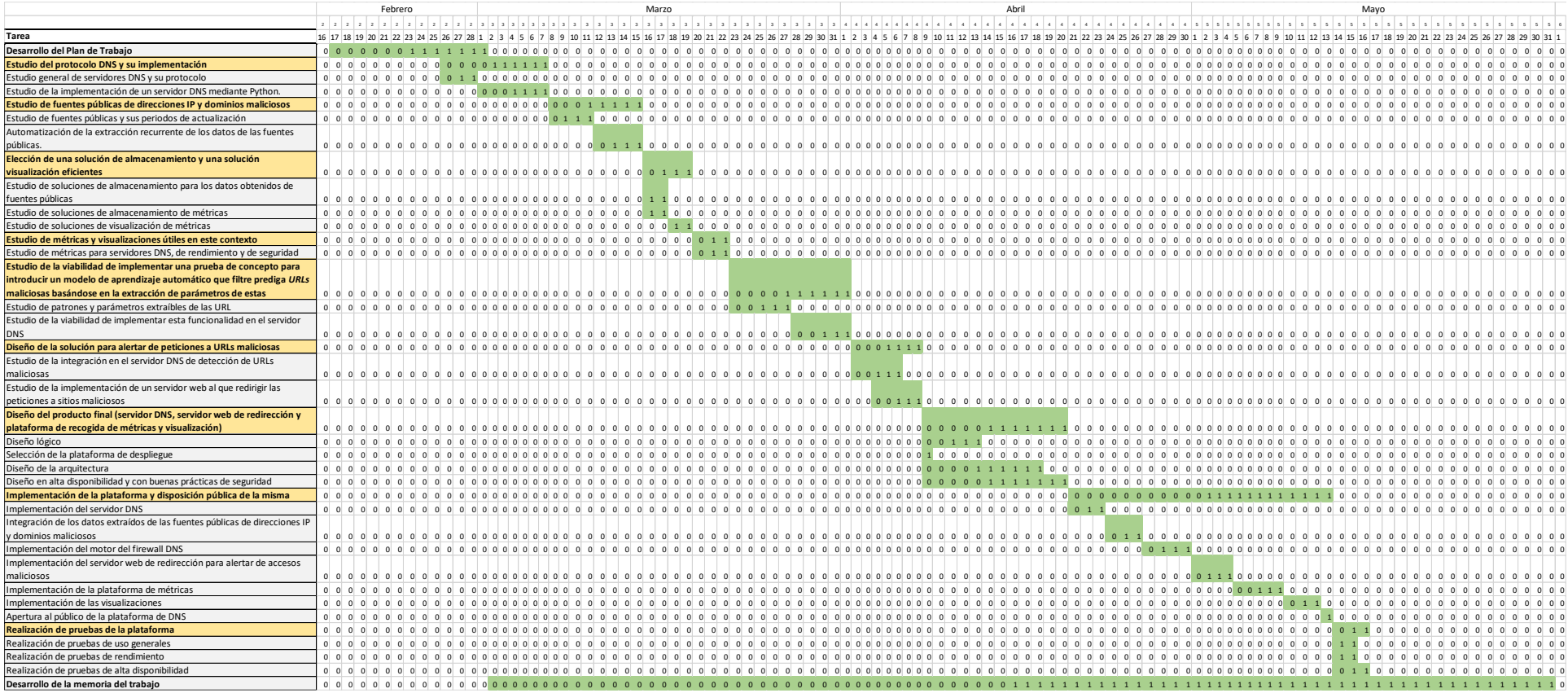

Figura 1. Diagrama de Gantt.

1.4.3 Seguimiento de la planificación

A lo largo del trabajo, se han dispuesto una serie de entregas parciales. El cumplimiento de la planificación en cada una de ellas se encuentra detallado en la Tabla 3.

Entrega	Cumplimiento de objetivos
PEC 1 – Plan de trabajo (01/03/2022)	La única tarea prevista para esta entrega es la elaboración del Plan de trabajo. La tarea se ha completado correctamente en plazo.
PEC 2 – Entrega de seguimiento 1 (29/03/2022)	Esta entrega comprende hasta la tarea de la fase de diseño “Estudio de patrones y parámetros extraíbles de las URL”. Se han completado el 100% de las tareas previstas para esta entrega y se ha avanzado correspondientemente en la tarea de elaboración de la memoria del trabajo.
PEC 3 – Entrega de seguimiento 2 (26/04/2022)	Esta entrega comprende hasta la tarea de la fase de implementación “Integración de los datos extraídos de las fuentes públicas de direcciones IP y dominios maliciosos”. Se han completado todas las tareas previstas para esta entrega y se ha avanzado correctamente en la redacción de la memoria del trabajo.
PEC 4 – Entrega final (31/05/2022)	Esta entrega comprende todas las tareas del proyecto incluyendo la fase de pruebas y finalización de la memoria del trabajo. Se han completado para esta entrega en plazo el 100% de las tareas previstas dentro del trabajo.

Tabla 3. Seguimiento de la planificación.

1.5 Estado del arte

Tal y como se menciona en el capítulo 1.1 Contexto y justificación del Trabajo, las peticiones DNS son un punto muy útil en el que poner el foco de atención para mejorar la seguridad y privacidad de los usuarios. Esto es así ya que, la resolución de nombres es un elemento clave en la navegación por internet y en general en las comunicaciones entre dispositivos y aplicaciones.

Dentro de las implementaciones ya dispuestas encontramos dos tipos diferentes:

- Soluciones comerciales. Como por ejemplo de Infoblox [1] o de EfficientIP SOLIDserver [2] que, comúnmente están basadas en *Response Policy Zones* (RPZ) que hacen uso de fuentes tanto públicas como privadas de dominios maliciosos para detectar peticiones que se consideran peligrosas.
- Implementaciones procedentes de la comunidad o investigaciones. Dentro de estas implementaciones existen implementaciones clásicas mediante el uso de RPZ como por ejemplo [3], pero también se estudian otros tipos de implementaciones como los basados en aprendizaje automático para la detección de dominios maliciosos, como por ejemplo [4].

Teniendo en cuenta estos dos enfoques, en este trabajo se pretende estudiarlos más a fondo. Se tendrá en cuenta principalmente en enfoque de las RPZ mediante el uso de listas negras de dominios y direcciones IP, pero también se pretende realizar una prueba de concepto para estudiar la viabilidad de implementar un modelo de aprendizaje automático que funcione de manera complementaria.
Para la recopilación de direcciones IP y dominios maliciosos se estudiará y hará uso de diferentes fuentes públicas dentro de un gran número de ellas que la comunidad pone a disposición pública para una protección comunitaria.
Observando ambos enfoques, se observa que los productos comerciales sí que disponen de ciertas visualizaciones con las que el administrador del producto pueda observar ciertas métricas, pero las versiones provenientes de investigaciones y productos *open source* se centran en la funcionalidad y no proveen métricas.

1.6 Recursos necesarios

Para la realización de este trabajo, adicionalmente al dispositivo que se utilice para su investigación, diseño e implementación, es necesario disponer de la infraestructura que lo soporte. Esta infraestructura se desplegará en un proveedor de nube pública, más concretamente en Microsoft Azure con una cuenta de estudiante.

1.7 Presupuesto del proyecto

Al ser uno de los objetivos de este trabajo su realización mediante el uso de tecnologías *open source*, el presupuesto se limita al coste de la infraestructura que soportará el producto final. Al no tener dispuesto el diseño aún, y al no haber decidido aún el proveedor de nube pública que se utilizará, no se puede dar una estimación de precio. Sin embargo, algunos proveedores de nube pública disponen de créditos gratuitos a estudiantes para la realización de proyectos por lo que, al poder hacer uso de ellos para este proyecto, el presupuesto total final será de 0€.

2. Estudio preliminar

2.1 Servidores DNS

Los servidores DNS (*Domain Name Server*) son una pieza clave en el uso de Internet hoy en día, principalmente en la navegación web.

Estos servidores implementan el protocolo con el mismo nombre y se encargan, a muy alto nivel, de traducir nombres de dominio a direcciones IP de forma jerárquica y descentralizada. La implementación de este protocolo está definida principalmente en los RFC 1034 [5] y RFC 1035 [6], aunque después se ha ido expandiendo su funcionalidad con otros RFC posteriores.

Cada cliente dispone de uno o varios servidores DNS configurados, locales o públicos, hacia los cuales redirige sus peticiones. Estos servidores, en caso de no ser el servidor final del dominio y no disponer de la respuesta almacenada en caché, reenvían la petición de forma recursiva a otros servidores hasta conseguir la respuesta. En caso de no conseguirla porque no exista o no se encuentre, devolverían un mensaje de "dominio no encontrado" (*NXDOMAIN*).

En la Figura 2 se puede observar un ejemplo del flujo genérico de una petición DNS.

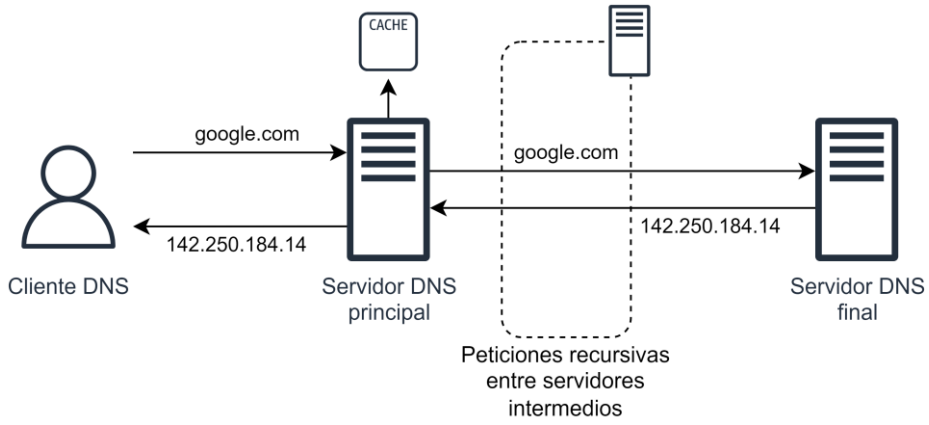

Figura 2. Ejemplo de flujo DNS.

El protocolo definido en los RFC 1034 [5] y RFC 1035 [6] describe tres elementos principales:

- ***Domain Name Spaces*** (espacios de nombres de dominio) y ***Resource Records*** (*RR*, registros de recurso)
- ***Name Servers*** (servidores de nombres)
- ***Resolvers*** ("resolutores")

El espacio de nombres de dominio es árbol estructurado y dividido en cuatro niveles de profundidad (o jerarquías) y en el que cada nodo contiene una etiqueta que se utiliza para describir el dominio. Los cuatro niveles de profundidad en los que se divide son:

1. Nodo raíz
2. *Top level domain* (*TLD*) que está a nivel regional.
3. *Second level domain* que está a nivel local.
4. *Subdomain* que está a nivel de la red local.

Tal y como se puede observar en la Figura 3, el nombre resultante en cada nodo es el equivalente a concatenar las etiquetas de cada nodo en orden inverso al nivel de profundidad.

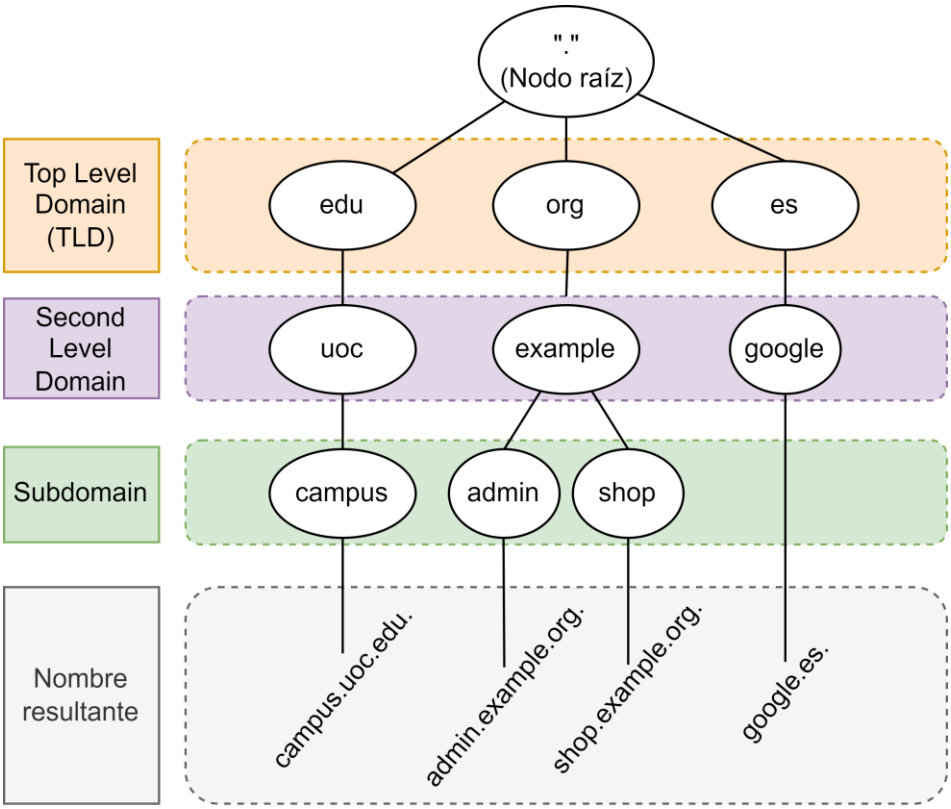

Figura 3. Ejemplo de árbol de espacio de nombres de dominio.

Cada uno de los nodos del árbol de un espacio de dominio de nombres define un nombre de dominio y está compuesto por diferentes registros de recursos (*RRs*). Cara registro de recursos está compuesto por la siguiente información:

- *NAME* (nombre): que contiene el nombre del propietario del registro.
- *TYPE* (tipo): que contiene el código del tipo de RR en cuestión. Los diferentes tipos se pueden observar en la Tabla 4.
- *CLASS* (clase): que contiene el código de la clase del RR en cuestión. Las diferentes clases se pueden observar en la Tabla 5.
- *TTL (time to live,* tiempo de vida): que contiene el tiempo en segundos que ese registro debe almacenarse en la caché del cliente.
- *RDLENGHT (*longitud de *RDATA*): que contiene la longitud en octetos del campo *RDATA* del registro.
- *RDATA (resource data*, datos del recurso): que contiene una cadena de texto con la información del recurso. El tipo de información que contiene depende del tipo y de la clase del registro.

Tipo	Descripción	Contenido de *RDATA*
A	Dirección de host	Dirección IP
NS	Servidor de nombres autoritativo	Nombre de host
CNAME	Nombre canónico para de un alias	Nombre de dominio
SOA	Inicio de zona de autoridad	Varios campos
PTR	Puntero a un nombre de dominio	Nombre de dominio
HINFO	Información de host	Tipo de CPU y sistema operativo
MX	Intercambio de email	Preferencia (numérica) y nombre de host
TXT	Texto	Texto

Tabla 4. Tipos de RR (TYPE)

Tipo	Descripción
IN	Internet
CH	*CHAOS*

Tabla 5. Clases de RR (CLASS)

Los **servidores de nombres** son los servidores que almacenan la información del árbol de los espacios de nombres de dominio y sus elementos y disponen de una memoria caché para almacenar las entradas el tiempo estipulado en el campo *TTL*. Adicionalmente, estos servidores son capaces de distribuir las peticiones e información hacia otros servidores.
Los ***resolvers* o "resolutores"** son los programas encargados de realizar peticiones a los servidores de nombres para extraer la información de los nombres de dominios que se consulten.

2.1.1 Servidores DNS en Python

Al estar este trabajo implementado en Python (versión 3), el servidor DNS y la gestión de sus peticiones y respuestas se hace a través del uso de la librería dnslib [7]. Esta librería implementa el protocolo DNS abstrayendo toda la gestión de los paquetes en formato binario y simplificando así toda la gestión del servidor. Debido a esta abstracción y las ventajas que ella supone, se considera útil su uso a lo largo del trabajo. Adicionalmente, esta librería está actualmente soportada y en desarrollo con actualizaciones recientes.
Se hace uso de las librerías "socket" y "*threading*" para la creación del servidor UDP que recibe y envía los paquetes a los clientes. Por cada cliente, se lanza un hilo con una función de gestión de peticiones DNS que implementa la funcionalidad del firewall DNS mediante el uso de los métodos de la librería dnslib.

2.2 Fuentes públicas de direcciones IP y dominios maliciosos

Debido al gran incremento de ciberataques, las comunidades empresariales, científicas y de individuos cada día están más concienciadas al respecto y comparten información sobre los ataques que han recibido para, en conjunto,

evitarlos en la mayor medida posible. La colección, puesta en público y consulta de esta información se engloba dentro del término "*Cyber Threat Intelligence (CTI)*" o inteligencia de ciber amenazas. El objetivo de la *CTI* [8] es el de recolectar indicadores de compromiso y depositarlos en una red de "conocimiento compartido" con el fin de permitir a los usuarios de esa red ser capaces de detectar y prevenir métodos y procesos de ataque que han sucedido en otros espacios. También, tal y como se menciona en [9], los dos problemas principales que se abordan mediante el uso de la *CTI* son la detección de vectores de ataque y la detección de indicadores de ataque.
Con lo que respecta al ámbito de este trabajo, la detección de los indicadores de ataque es de gran utilidad debido a que, para un *firewall DNS*, lo datos necesarios son IPs y dominios maliciosos. Estos se pueden obtener de fuentes de inteligencia abiertas de la red compartida de *CTI* en la que usuarios que han sufrido ataques por parte de estos, publican sus datos.
Para este trabajo, se han estudiado diferentes fuentes públicas y se han seleccionado las correspondientes a la Tabla 6.

Fuente	URL del proyecto	Tipo	Descripción
URLHaus de Abuse.ch	https://urlhaus.abuse.ch/about/	Dominios (*malware*)	Lista de dominios que se utilizan para la distribución de *malware*.
Phishing.Database	https://github.com/mitchellkrogza/Phishing.Database	IPs y dominios (*phishing*)	Lista de IPs y dominios que se utilizan en ataques de *phishing*.
IPsum	https://github.com/stamparm/ipsum	IPs (*malware*)	Lista de IPs de servidores maliciosos. Se utilizan más de 30 fuentes de CTI para su creación.
OpenPhish	https://openphish.com	Dominios (*phishing*)	Lista de dominios que se utilizan en ataques de phishing. Se hace uso de la lista en versión *community.*
notracking/hosts-blocklists	https://github.com/notracking/hosts-blocklists	Dominios (*trackers*)	Lista de dominios de *trackers* de anuncios. Contiene también algunos dominios que apuntan a sitios maliciosos.
Dan.me.uk Tor Nodes	https://www.dan.me.uk/tornodes	IPs (nodos tor)	Lista de IPs de nodos identificados de la red TOR.
FireHOL criptomineros y red Bitcoin	http://iplists.firehol.org/	IPs (nodos bitcoin y criptomineros)	Lista de IPs de nodos identificados de la red Bitcoin y de servidores utilizados por criptomineros ilegales.

Tabla 6. Listado de fuentes públicas de IPs y dominios maliciosos.

2.3 Bases de datos

Dentro de las soluciones de almacenamiento de datos, existe una gran variedad de ellas con diferentes propósitos. Este trabajo necesita una solución de almacenamiento tanto para los datos recolectados de fuentes públicas como para las métricas que se recogen. Estos datos, necesitan ser consultados para los fines del trabajo: filtrar IPs y dominios maliciosos y consultar las métricas de forma gráfica. Debido a esta necesidad, el tipo de soluciones de almacenamiento más indicado son las bases de datos por su definición nativa de disponer de métodos para realizar consultas a los datos de forma eficiente.

Dentro de las bases de datos, existen dos tipos principales en función a su modelo de datos [10]: relacionales y no relacionales.

Las bases de datos relacionales organizan la información en tablas reducidas que se relacionan con las demás formando un modelo de datos completamente interconectado mediante el uso de identificadores. Las bases de datos relacionales más conocidas son MySQL (y MariaDB), PostgreSQL, Oracle DB y Microsoft SQL Server.

Las bases de datos no relacionales, por el contrario, no utilizan identificadores para las entradas que almacenan y por lo tanto no disponen de los datos relacionados entre sí. Los datos que se almacenan pueden ser de diferentes tipos como: clave-valor, documentos o grafos. Las bases de datos no relacionales más conocidas son MongoDB, Redis y ElasticSearch.

2.4 Visualización de métricas

En el campo de las herramientas de visualización de métricas existen numerosas soluciones. Al ser este un campo en auge, las herramientas disponibles y funcionalidades de estas varía en poco tiempo. Dentro de las soluciones *open source* también existe una gran variedad siendo Prometheus, Graphite, Grafana y Kibana las más utilizadas en el mundo empresarial tal y como se describe en [15]. Dentro de estas soluciones, Graphite y Prometheus están diseñadas únicamente como herramientas de visualización de métricas numéricas en series de tiempo.

Prometheus no es simplemente una herramienta de visualización de métricas en series de tiempo, es una pila de servicios que incluye, además de la visualización, la recogida de estas métricas. Dispone de una amplia aceptación casi estandarizando la extracción de métricas de aplicativos al ser el único proyecto graduado por la Cloud Native Cloud Foundation [16] en el apartado de monitorización. Sin embargo, se trata de un proyecto utilizado en su mayoría como elemento que estandariza y recolecta las métricas sin tener un gran uso como plataforma de visualización de métricas.

Graphite, solo acepta, al igual que Prometheus, datos numéricos en series de tiempo y, por ello, dispone de una menor popularidad en la comunidad debido a la competencia que le ofrece.

Tanto Kibana como Grafana por su parte aceptan además de datos numéricos otras fuentes de información como texto en estático para visualizar tablas.

Kibana por su parte dispone de una integración nativa con ElasticSearch ya que es un producto desarrollado por la propia compañía.

Grafana dispone de integraciones nativas tanto con ElasticSearch como con otras soluciones, entre ellas Prometheus. Grafana, además, dispone de una

fuerte comunidad que la sustenta y genera materiales para su reutilización como paneles de visualización e integraciones con servicios de alerta.

2.5 Detección de *phishing* mediante aprendizaje automático

Además de la detección de posibles ataques mediante listas negras de direcciones IP y dominios maliciosos, otro enfoque que está ganando popularidad en el campo de la ciberseguridad para la detección de amenazas es el uso de modelos de aprendizaje automático que modelen el comportamiento nominal o el anómalo con el fin de detectar patrones que se salgan de esos modelos nominales o que coincidan con los modelos anómalos.
Dentro de una solución de firewall DNS la aplicación de estos modelos puede ser de gran utilidad ya que, supone una forma de detección complementaria de posibles amenazas. Sin embargo, esta detección solo puede aplicarse a nombres de dominio debido a que las direcciones IP no tienen patrones que se puedan extraer por su característica dinámica. Un estudio interesante que propone una serie de características a extraer es el correspondiente a [11]. En él se proponen ciertos atributos a extraer en el contexto del léxico de la URL y de ciertos atributos a extraer en el contexto del contenido HTML de las webs a las que redirigen y extraído de webs de confianza de dominios como Alexa Rank.
Otro estudio similar, [12], se proponen también una serie de atributos que se pueden extraer de las URLs, a nivel léxico, así como obtener de las páginas web a las que redirigen, a nivel de petición HTTP.
En el contexto de un servidor DNS, en el cual se requiere rapidez a la hora de procesar las peticiones, tienen mayor sentido y utilidad las características puramente léxicas y que no dependan de peticiones HTTP a las webs finales ni a otras terceras. Por ello, se estudian únicamente las características léxicas que se pueden extraer de las URL y que no requieran de peticiones ni utilidades externas. Adicionalmente, en el contexto de un servidor DNS, solo se pueden tener en cuenta las características que estudien o valoren propiedades del nombre de dominio, por lo que es necesario descartar también todas las características en cuyo fin se utilice cualquier parte del dominio.
Las características léxicas que se pueden extraer de las URLs y que se proponen en el artículo [12] coinciden y aumentan las características del artículo [11]. Entre ellas, descartando las que no aplican en el caso de un servidor DNS están:

- Número de puntos.
- Número de niveles de subdominio.
- Longitud del nombre de dominio.
- Número de guiones en el dominio (“-“).
- Número de caracteres numéricos.
- Número de palabras sensibles "secure", "account", "login", "banking", "confirm", "bank", "pass", "bitcoin", "coin")***** en el nombre de dominio.
- Si existe un nombre de marca en el nombre de dominio ("amazon", "santander", "bbva", "ing", "caixa", “paypal”, "adidas", "nike", "facebook", "instagram", "tiktok", "twitter", "whatsapp", "youtube", "google", "gmail", "vodafone", "movistar", "apple", "samsung", "microsoft")******.

***** La lista de palabras sensibles ha sido seleccionada del propio artículo y se han añadido algunas palabras adicionales.

****** La lista de marcas ha sido propuesta a modo ejemplo para la redacción a este trabajo basándose en las marcas más populares con presencia en Internet en España.

Para el estudio de la viabilidad de su aplicación en el firewall DNS se ha hecho uso de estas características, así como la base de datos proporcionada en [13] que contiene URLs maliciosas (*phishing*) y URLs normales.

Primero, se ha aplicado una limpieza a esta base de datos eliminando toda la información no correspondiente al nombre de dominio y se han tenido en cuenta únicamente las características dominio y clase. Posteriormente, se han calculado los valores para cada dominio de cada una de las características mencionadas previamente. Haciendo uso de un clasificador "*Random Forest*", particionando la base de datos en diferentes conjuntos de entrenamiento y prueba aleatoriamente y promediando, las métricas obtenidas han sido las siguientes:

		Real	
		Normal (-)	**Phishing (+)**
Predicho	**Normal (-)**	5962	2585
	Phishing (+)	1349	5462

Tabla 7. Matriz de confusión de la prueba de clasificación.

Métrica	**Valor**
Precisión	74.38%
Ratio de verdaderos positivos, *TPR* (*Recall*)	67.87%
Ratio de verdaderos negativos, *TNR*	81.55%

Tabla 8. Métricas de la prueba de clasificación.

La información detallada junto con el código de estas pruebas se encuentra y se puede consultar en el 1.

Como se observa en las métricas, los valores no indican un buen comportamiento en cuanto a la detección. Si bien, disponen de un 74% de precisión detectando tanto URLs reales como de *phishing*, este valor no es notablemente alto. Además, solo se detectan un 68% de los casos positivos dejando un 32% de los casos que realmente son *phishing* sin detectar como tal. También, el 18.5% de los casos que no son *phishing* se detectan como tal generando bastante ruido.

Sin embargo, este motor de búsqueda puede ser un buen complemento a la hora de detectar casos de *phishing* si reducimos la cantidad de falsos positivos que son los que pueden afectar en gran medida a la experiencia de navegación de los usuarios.

3. Diseño

3.1 Diseño general

La arquitectura general de la solución se puede observar en la Figura 4.
Tal y como se puede observar, existen dos actores principales que son el cliente DNS que hace peticiones al servidor para ser alertado cuando acceda a una página sospechosa y el operador de seguridad que consulta las visualizaciones generadas por las métricas recogidas y recibe las alertas de disponibilidad que se generen.
Todos los desarrollos necesarios, implementados en Python, se despliegan como ejecutables individuales en sus correspondientes servidores mediante servicios de *systemd* para su inicio automático con los servidores. Para facilitar la ejecución de todos ellos, se hace uso de la librería argparse para la recogida de parámetros y la creación de la ayuda de cada uno de ellos.
Dentro del diseño general, se dispone de los siguientes elementos principales:

- **Balanceador de carga público**: al cual llegarán las peticiones de resolución de nombres y las peticiones del servidor web sumidero de alerta.
- **Grupo de servidores "Firewall DNS"**: el cual ejecuta el propio servidor DNS y, periódicamente mediante el uso de *crontab*, actualiza las fuentes de IPs y dominios maliciosos.
- **Grupo de servidores "Servidor web de alerta"**: el cual ejecuta el servidor web sumidero al cual se redirigen todas las peticiones DNS que redirigen a una IP o dominio considerado malicioso, que muestra el mensaje de alerta y, permite al usuario continuar aceptando los riesgos.
- **Balanceador de carga interno**: al cual llegan todas las peticiones de la plataforma de recogida de logs y almacenamiento. Entre esta información se incluye el envío de los logs y métricas recolectadas en los servidores y las consultas a la plataforma de almacenamiento de IPs y dominios maliciosos.
- **Grupo de servidores "Logstash y ElasticSearch"**: la cual se encarga de recolectar los logs y métricas, procesarlos y almacenarlos. También se encarga de almacenar la información de IPs y dominios maliciosos. La elección de esta plataforma se justifica en el apartado 3.3 Solución de almacenamiento.
- **Servidor Grafana**: el cual se encarga de ofrecer visualizaciones de métricas seleccionadas y enviar las alertas.

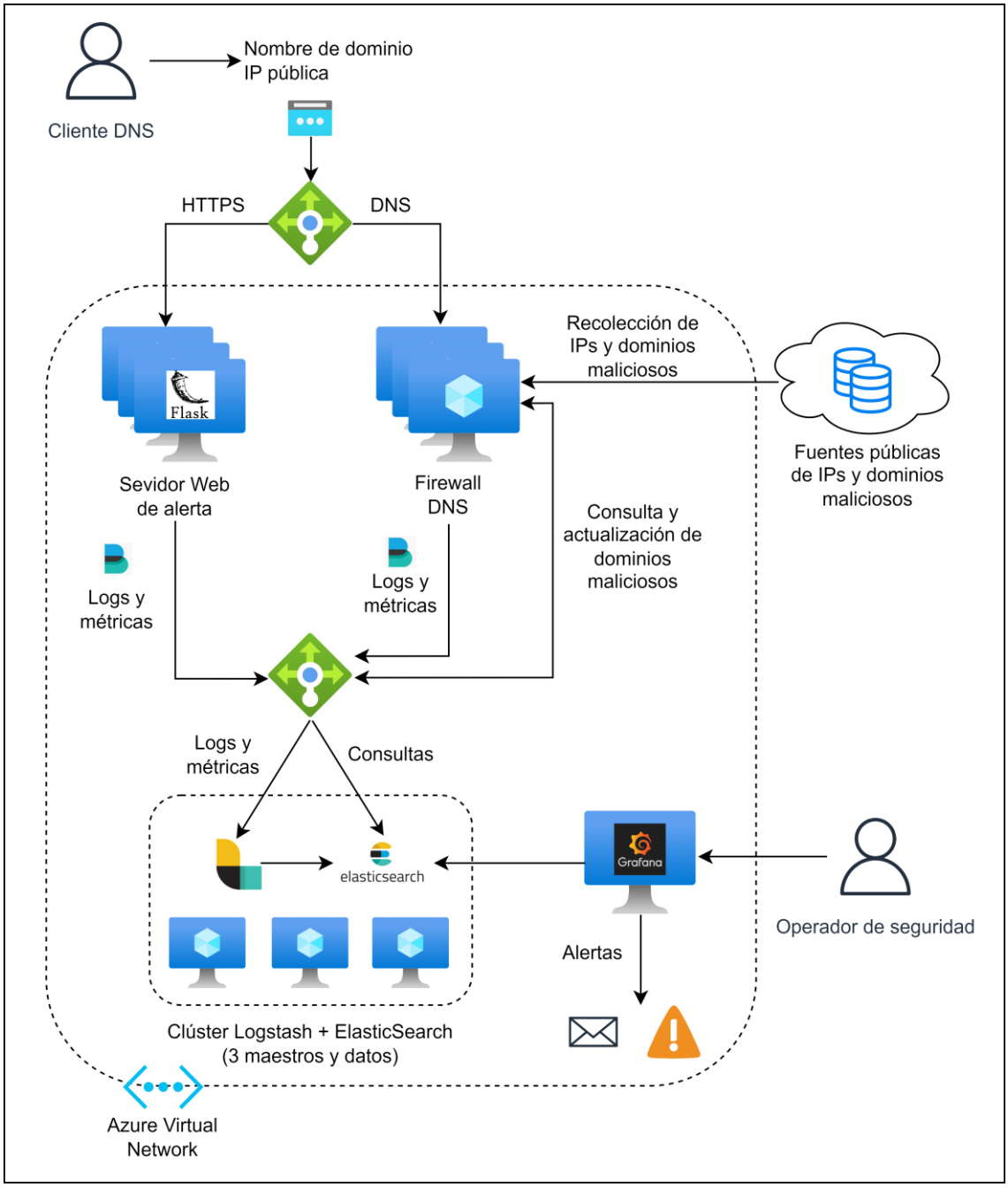

Figura 4. Arquitectura general.

Para el desarrollo de este trabajo se ha hecho uso de un nombre de dominio real y estático: "***firewalldns.systems***". Dado el objetivo de realizar el trabajo haciendo uso de recursos gratuitos, este dominio se ha registrado en el proveedor de servicios DNS Name.com el cual ofrece dominios de forma gratuita a los participantes del programa GitHub Student Developer dentro del paquete de recursos gratuitos que ofrecen.
Este nombre de dominio apunta a la dirección IP pública del balanceador de carga público y es el encargado de redirigir todo el tráfico entrante tanto al servidor DNS como al servidor web de alertas.

3.2 Recolección automática de datos

Disponiendo de las fuentes seleccionadas en el apartado de estudio y haciendo uso de Python como a lo largo de todo el trabajo, se automatiza su recogida mediante el uso de un script. Este script, utiliza un fichero de configuración en el cual se añaden todas las fuentes con los siguientes datos:

- Nombre: nombre de la fuente
- URL: enlace al fichero en formato texto plano que contiene las entradas separadas línea por línea.
- Expresión regular: utilizada para procesar cada línea de la fuente y extraer de ella los datos necesarios (dirección IP o dominio).
- Tipo: indica si la fuente es de tipo IP (para IPs maliciosas) o de tipo dominio (para dominios maliciosos).
- Categoría: indica la categoría de amenaza que presenta cada fuente, como por ejemplo *phishing* para dominios y *malware* para IPs.

El script, que se lanza automática y periódicamente, una vez al día a las 03:00, mediante el uso de *crontab* en los servidores que ejecutan el firewall DNS, lee este fichero de configuración, descarga las fuentes mediante HTTP (librería *requests*), procesa los archivos línea a línea y, finalmente, los envía a la solución de almacenamiento.

Los detalles de este script se encuentran en el Anexo 2 para su consulta. Este script hace uso de un fichero de configuración del que se leen las fuentes. El fichero utilizado en este trabajo se encuentra en el Anexo 3 para su consulta.

3.3 Solución de almacenamiento

Al tener la necesidad de disponer de una solución que actúe como plataforma de recogida de métricas, ElasticSearch es una buena opción, comúnmente utilizada en la industria con este fin, *open source*, y con un motor de búsqueda muy eficiente ya que utiliza Apache Lucene para ellas. También es más eficiente que las búsquedas con motores de bases de datos relacionales con los datos indexados tal y como se observa en [14].

Adicionalmente, ElasticSearch dispone de características nativas para su despliegue de forma distribuida para así garantizar la alta disponibilidad y diferentes herramientas con integración nativa para la recolección, procesamiento, consulta y visualización de estas métricas.

Al necesitar también una plataforma de almacenamiento de IPs y dominios maliciosos, se considera que ElasticSearch es una buena opción por las mismas bondades: alta eficiencia en búsquedas y configuración en alta disponibilidad nativa. Además, al hacer uso de ElasticSearch para la plataforma de recogida de métricas, se considera una buena opción de diseño unificar todos los datos dentro de la misma plataforma.

Para la recogida de las métricas y logs de los aplicativos desarrollados se hace uso de Filebeat en cada uno de los *hosts* que ejecute un aplicativo. Este los envía a Logstash en el cual se hace un procesamiento especial para cada tipo de dato y, finalmente, se envían a ElasticSearch para su indexación.

La recolección de datos de IPs y dominios maliciosos, realizada de forma automatizada, almacena los datos en ElasticSearch mediante su indexación directa.

3.3.1 Diseño de ElasticSearch

Para garantizar la disponibilidad de ElasticSearch en todo momento, se ha desplegado y configurado en modo clúster con tres nodos actuando en rol de maestro y de datos al mismo tiempo. Se configuran tres para que, en caso de que un nodo deje de funcionar, la alta disponibilidad siga estando garantizada. Adicionalmente, teniendo en cuenta que la infraestructura está desplegada en la nube pública de Azure, cada uno de los tres nodos está desplegado en una zona de disponibilidad diferente dentro de la misma región.
Para la configuración del clúster de ElasticSearch se hace uso de la propia funcionalidad que dispone implementada el propio producto, mediante la cual existen dos roles diferentes: nodos maestros y nodos de datos. Los nodos maestros se encargan de la distribución de los datos y de gestionar las peticiones de manera distribuida. Los nodos de datos se encargan de almacenar, indexar los datos recibidos y realizar búsquedas en ellos. En este despliegue, al no considerarse necesario disponer de un número mayor de nodos, los tres nodos cuentan con ambos roles.
Para una mejor seguridad dentro de la plataforma, para la conexión de aplicativos con ElasticSearch se hace uso de *API-keys* con los permisos mínimos necesarios para ello. Esto es, permisos de lectura para los aplicativos que solo requieran realizar lecturas a ciertos índices y permisos de lectura y escritura para los aplicativos que requieran realizar lecturas e indexaciones.
Adicionalmente, junto con el despliegue de la plataforma, se realiza una rotación de todas las contraseñas de los usuarios por defecto de ElasticSearch y se crea un usuario administrador diferente del usuario por defecto (*elastic*) ya que este es un usuario conocido y presente en diferentes filtraciones de contraseñas y diccionarios.

3.3.2 Diseño de índices de ElasticSearch

Los índices de ElasticSearch son la agrupación lógica que dispone para la separación de datos. Por ello, para la implementación de toda la arquitectura del Firewall DNS se hace uso de un índice para cada una de las funcionalidades requeridas. Estos índices y sus funcionalidades se pueden observar en la Tabla 9.

Nombre de índice	Descripción	Políticas de ciclo de vida
malicious-domain	Índice que contiene los dominios maliciosos obtenidos de las fuentes públicas. Este índice se actualiza de forma periódica mediante el uso del script que actualiza los datos. Adicionalmente contiene los datos de la categoría y el nombre de la fuente.	Fase activa: 10GB o 1 día. Fase de eliminación: 1 día. En total, 2 días de periodo de vida útil.
malicious-ip	Índice que contiene las IPs maliciosas obtenidas de las fuentes públicas. Este índice se actualiza de forma	Fase activa: 10GB o 1 día. Fase de eliminación:

	periódica mediante el uso del script que actualiza los datos. Adicionalmente contiene los datos de la categoría y el nombre de la fuente.	1 día. En total, 2 días de periodo de vida útil.
webserver-bypass	Índice que contiene los documentos con los dominios mapeados con las URLs de bypass generadas por el servidor web para los usuarios que quieren aceptar los riesgos y continuar a dominios considerados maliciosos.	Fase activa: 1GB o 1 minuto Fase de eliminación: 14 minutos. En total, 15 minutos de periodo de vida útil.
general-metrics	Índice que contiene los logs de métricas del firewall DNS, del servidor web de alerta y del actualizador de datos.	Fase activa: 10GB o 1 día. Fase de eliminación: 30 días. En total, 31 días de periodo de vida útil.
general-logs	Índice que contiene los logs que no provienen de métricas del firewall DNS, del servidor web de alerta y del actualizador de datos.	Fase activa: 10GB o 1 día. Fase de eliminación: 90 días. En total, 91 días de periodo de vida útil.
metricbeat	Índice que contiene las métricas de las comprobaciones de salud (*health checks*) de los servidores y sus procesos críticos para permitir un alertado cuando se detecten periodos de caída.	Fase activa: 10GB o 1 día. Fase de eliminación: 90 días. En total, 91 días de periodo de vida útil.
exceptions	Índice que contiene las excepciones del firewall DNS para realizar el proceso de *bypass* solventando el problema de certificados tal y como se comenta más adelante.	Fase activa: 1GB o 1 minuto. Fase de eliminación: 9 minutos. En total, 10 minutos de periodo de vida útil.

Tabla 9. Índices de ElasticSearch.

ElasticSearch dispone de un módulo integrado de gestión del ciclo de vida de los índices. Este módulo permite disponer de los datos de los índices en cuatro

fases diferentes: almacenamiento caliente, templado, frío y eliminación. Mediante el uso creación de políticas de ciclo de vida de índices y de plantillas de índices para enlazar los índices a estas políticas, es posible configurar una rotación efectiva de los datos de estos. Mediante el uso de esta funcionalidad, cada uno de los índices dispone de un alias, que es el nombre que se puede observar en la Tabla 9, y para realizar inserciones o consultas a ellos, es suficiente con apuntar a este índice. En realidad, estos índices son tratados de forma interna por ElasticSearch con una nomenclatura:

<nombre del índice>-XXXXXX

Dónde XXXXXX es un número entero incremental que aumenta con cada índice que se crea en cada rotación y dónde el primer índice tendría el id 000001. Este número por convención se autoajusta con ceros a la izquierda para completar los huecos. Cuando un índice acaba su periodo de vida en almacenamiento caliente, bien por haber llegado al tamaño de almacenamiento máximo establecido o bien por haber llegado al tiempo de vida establecido, este se bloquea y pasa a la siguiente fase y se crea otro índice activo con el id siguiente al actual que pasa a ser el que recibe los datos cuando se indexan. Sin embargo, cuando se realizan consultas, estas se siguen realizando a todos los datos disponibles de todos los índices independientemente de la fase en la que se encuentren. Una vez los datos llegan a la fase de eliminación y termina su periodo de vida, estos son eliminados de forma permanente.
En el caso del Firewall DNS, los periodos de vida de cada fase en cada índice se pueden consultar en la Tabla 9.

De forma adicional, para garantizar la alta disponibilidad de los datos, aun cuando uno de los nodos se caiga, se hace uso de la funcionalidad de partición de datos integrada en ElasticSearch. Esta funcionalidad permite particionar los índices en diferentes particiones o “*shards*” y, además, permite disponer de estos “*shards*” replicados un número determinado de veces actuando uno de forma activa y el resto de forma pasiva hasta ser necesarios. Para garantizar la alta disponibilidad al cien por cien de los datos de todos los índices, se han configurado para que cada uno disponga de tres “*shards*” y de un factor de replicación de tres. Se puede observar un ejemplo de la arquitectura del particionado del índice *webserver-bypass* a lo largo de los tres nodos del clúster de ElasticSearch en la Figura 5. En ella se puede ver cómo los datos están completamente replicados en los tres nodos, y en caso de que se caiga cualquiera de ellos, los datos seguirían estando completamente disponibles.

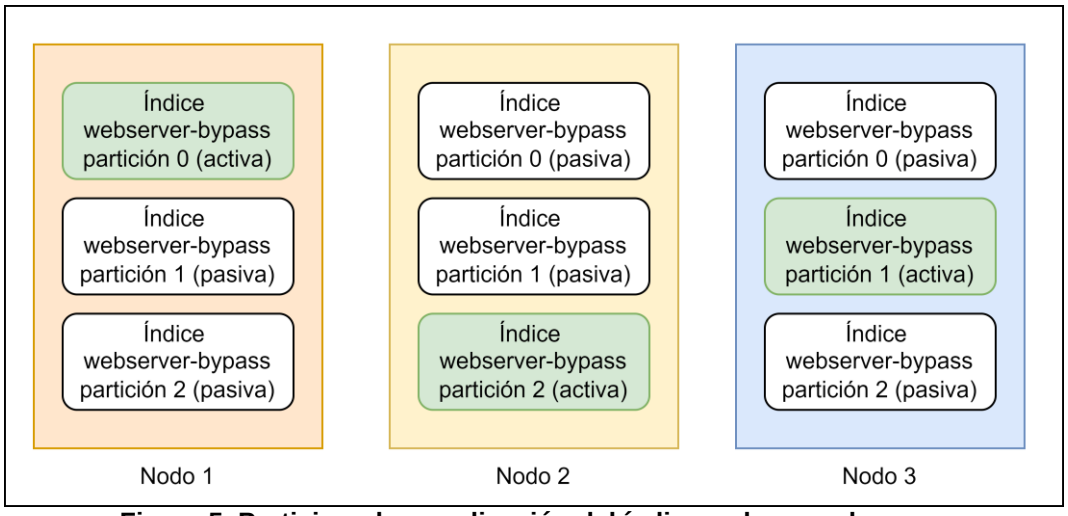

Figura 5. Particionado y replicación del índice webserver-bypass.

3.4 Solución de visualización de métricas

Teniendo en cuenta las soluciones estudiadas en el capítulo anterior, se consideran las soluciones más adecuadas para este proyecto Kibana y Grafana al aceptar datos que no sean numéricos dando la posibilidad así de enriquecer los paneles de visualización con información adicional.

Observando que se hace uso de ElasticSearch para este proyecto, Kibana se considera más adecuada debido a su nativa integración con ElasticSearch y las opciones adicionales de gestión que ofrece. Sin embargo, la versión *open source* de ElasticSearch no permite la creación y el envío de alertas de forma externa a la plataforma, únicamente permite la creación de alertas en un índice. Grafana sí que lo permite en su versión *open source* y, además, dispone de numerosas integraciones como envío de emails o mensajes a Telegram.

Por esto, la solución de visualización de métricas que se considera idónea para este trabajo es Grafana.

3.5 Diseño del servidor DNS

El servidor DNS, implementado en Python, utiliza la librería *socket* para exponer el puerto 53 y recibir peticiones en él a través del protocolo UDP. Para cada una de las peticiones, y mediante el uso de la librería *threading*, se crea un hilo para la gestión de su petición, y, una vez la petición es gestionada, este hilo es cerrado.

El diagrama de flujo de una petición en el servidor DNS es el correspondiente a la Figura 6.

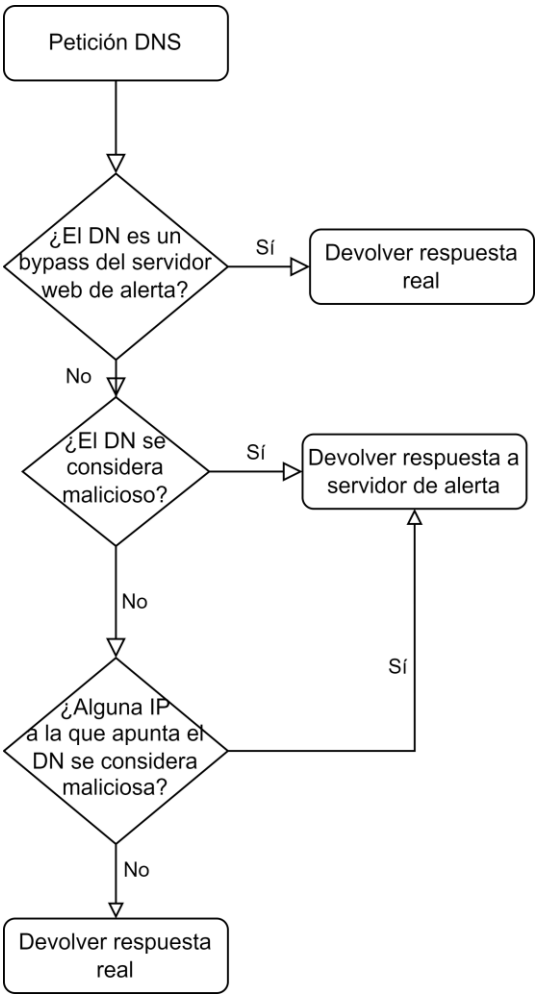

Figura 6. Diagrama de flujo petición DNS tentativo.

Las peticiones detectadas como normales, que no son dominios maliciosos ni apuntan a IPs detectadas como maliciosas se devuelven de forma normal. Para la resolución de estas peticiones se hace uso de una lista de servidores DNS recibidos como parámetro en la ejecución del servidor, como por ejemplo los servidores DNS públicos de Google 8.8.8.8 y 8.8.4.4. Las peticiones se realizan de forma consecutiva y en orden aleatorio en cada uno de ellos hasta encontrar una solución en uno de ellos que es devuelta. En caso de no encontrar una solución en ninguno de ellos se devolvería un mensaje de nombre de dominio no encontrado (*NXDOMAIN*). Para mitigar los problemas que podría causar la caída de alguno de estos servidores DNS, se establece un tiempo de espera muy pequeño con el fin de, en caso de no recibir una respuesta en él, considerar el servidor como caído y pasar al siguiente servidor para que el rendimiento se vea afectado el menor tiempo posible.

Cuando una petición apunta a un dominio o una IP considerada maliciosa, se devuelve una respuesta apuntando al servidor web de alerta, en el cual se le muestra al cliente un mensaje de alerta indicando que está intentando acceder a un dominio considerado malicioso. En esta alerta, el cliente puede observar también la categoría en la que se ha catalogado el dominio o la IP como maliciosa y una opción para aceptar el riesgo y continuar al dominio real.

Para la transmisión al servidor web de alerta, se hace uso de una redirección IP para que el cliente sea redirigido al servidor web. Para ello se hace uso de un registro CNAME que un nombre especial para dejar constancia de esta

redirección cuando se visualizan los datos de las peticiones DNS. Por lo tanto, las peticiones redirigidas al servidor web de alerta se envían mediante esta vía con el siguiente formato:

```
<DN malicioso>.firewalldns.systems
```

Por ejemplo, teniendo en cuenta que el servidor web de alerta está expuesto en "*firewalldns.systems*" y se dispone de una petición entrante para resolver el dominio "*malicioso.org*":

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 433
;; flags: rd; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0
;; QUESTION SECTION:
;malicioso.org.                 IN      A
```

Y este dominio se considerase como malicioso por el firewall DNS, este devolvería al cliente un CNAME apuntando al dominio "*malicioso.org.firewalldns.systems*" y su resolución a la dirección IP del servidor web de alerta:

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 43488
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0,
ADDITIONAL: 0
;; QUESTION SECTION:
;malicioso.org.                     IN      A
;; ANSWER SECTION:
malicioso.org.              0       IN      CNAME
malicioso.org.firewalldns.systems.
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 43488
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL:
0
;; QUESTION SECTION:
; malicioso.org.firewalldns.systems.  IN      A
;; ANSWER SECTION:
malicioso.org.firewalldns.systems 300  IN   A    <IP Servidor Web>
```

El servidor web, una vez recibe una petición con el formato correcto, este procesa el nombre de dominio malicioso al que se ha redirigido, tal y como se explica en el apartado siguiente, y extrae de la fuente de almacenamiento la categoría de este dominio para mostrarle una alerta con mayor nivel de información al cliente.
Tras este mensaje de alerta, se le muestra al cliente un mensaje de aceptación de los riesgos para los casos en los que los clientes deseen continuar igualmente hacia esos dominios. Este mensaje de aceptación de los riesgos incluye un enlace para hacer clic en él y continuar al dominio. Para el diseño de esta funcionalidad, se han encontrado una serie de puntos para tener en cuenta:

- No se puede redirigir directamente al nombre de dominio real ya que esta petición volverá a ir al firewall DNS y la volverá a catalogar como maliciosa y se creará un bucle infinito.
- No se pueden utilizar sub-rutas dentro del mismo servidor web ya que estas no se incluyen en las peticiones DNS.

- No se pueden utilizar redirecciones a través de sub-rutas con alguna palabra clave ya que, los atacantes que conozcan este mecanismo podrían enviar enlaces de phishing utilizando este mecanismo para que el servidor DNS no los catalogase como maliciosos. Un ejemplo de este tipo de redirecciones utilizando el ejemplo anterior sería aceptar en el servidor DNS toda URL que comience por "*secure*" como "*secure.alertwebserver.*com/" y redirigirlo a "*malicioso.org*". De esta forma, cualquier atacante tendría la posibilidad de engañar a los clientes a través de *phishing*.

Como una posible solución, se ha implementado el uso de una sub-ruta del servidor web de alerta, pero en vez de incluyendo el nombre real del dominio, incluyendo una cadena aleatoria de caracteres y números que se mapean a su dominio real en ElasticSearch. De esta forma, cuando un usuario accede a una URL considerada maliciosa, se genera un enlace de la siguiente manera:

1. Se escoge un nombre aleatorio de las siguientes opciones:
   a. *bypass*
   b. *firewalldns*
   c. *insecure*
   d. *risk*
   e. *danger*
   f. *alert*
2. Se escoge un número aleatorio entre 10000000 y 99999999 (de 8 dígitos).
3. Se genera la URL con el formato:

```
<nombre de dominio del servidor>/bypass/<nombre>-<id>

Ejemplo: firewalldns.systems/bypass/bypass-12345678
```

4. Se almacena en un índice de ElasticSearch (*webserver-bypass*) como un documento cuyo id es el número aleatorio escogido y que tiene dos parámetros:
   a. "domain" que almacena el nombre de dominio real considerado malicioso al que apunta esta URL.
   b. "name" que almacena la palabra aleatoria escogida.

Después, cuando un usuario trata de acceder a una URL de forma escapada, el servidor DNS, antes de procesar las consultas, busca si esta proviene de un nombre de dominio con la sintaxis de escape. Si proviene con esta sintaxis, se extraen la palabra y el número aleatorios y se realiza una búsqueda en ElasticSearch. Si esta búsqueda resulta exitosa, se devuelve la URL real al usuario sin pasar por los filtros del Firewall DNS.

Haciendo uso de las políticas de ciclo de vida de los índices de ElasticSearch, y tal y como se detalla en la Tabla 9, los datos del índice dónde se almacenan estos mapeos, solo perduran durante 15 minutos en ElasticSearch, por lo que el usuario solo podrá acceder durante los próximos 15 minutos a la URL y posteriormente esta volverá a estar disponible para un nuevo mapeo. Este tiempo se considera tiempo más que suficiente para garantizar una navegación cómoda a la par que segura ya que, en caso de mantener estas URLs como permanentes, los usuarios podrían comenzar a hacer uso de ellas y los atacantes dispondrían cada vez de más probabilidades para realizar ataques de *phishing* con esta sintaxis.

De esta forma, incluso cuando un posible atacante dispone del mecanismo de generación de URLs de escape, para realizar un intento de *phishing* que eluda la protección del firewall, debe primero hacer una petición al servidor DNS con su URL maligna y después dispone de una probabilidad

$$Probabilidad_{total} = Prob_{cadena} \, x \, Prob_{id} \, ,$$

dónde

$$Prob_{cadena} = 1/6; \; Prob_{id} = 1/89999999 \Rightarrow$$

$$\boldsymbol{Probabilidad_{total} = 1{,}85185 * 10^{-9}.}$$

Como se puede observar, la probabilidad de que un posible atacante coincida con la URL adecuada es ínfima. De igual modo, en caso de considerar que se trate de una probabilidad lo suficientemente elevada, el rango de números se podría ampliar todo lo necesario. Este número tiene un límite de extensión de 512 bytes [17] (4096 bits), lo cual permite un número de ID máximo de $2^{4096}-1$.

3.6 Diseño del servidor web de alertas

El servidor web de alerta, implementado en Python haciendo uso de la librería Flask, está diseñado para funcionar en formato sin estado o *stateless* por lo que, para garantizar su alta disponibilidad, se disponen de tres servidores exactamente replicados, expuestos detrás de un balanceador de carga. Estos servidores, para una mayor garantía de alta disponibilidad están desplegados en zonas de disponibilidad diferentes dentro de una misma región de Azure. Este grupo de servidores exponen el *backend* de Flask a través de Apache y mediante el uso de su módulo WSGI. No se hace uso del propio servidor web que implementa Flask ya que este no está recomendado por la propia librería para entornos productivos.

El servidor web de alertas dispone de tres rutas mapeadas:

- Ruta “/about” la cual redirige a una página de inicio básica en la que se da información básica del servidor DNS y su método de configuración tanto en Windows como en Linux.
- Ruta “/bypass/<nombre-id>” la cual comprueba que el nombre e id recibidos sean correctos, y añade una excepción para el cliente para el dominio detectado como malicioso de ese *bypass*.
- Ruta “/query/<dominio>” la cual captura el dominio que recibe en la ruta de la URI de la petición HTTP para generar una alerta.

Para la implementación de este servidor web de alerta, se plantea el problema del método de comunicación con el Firewall DNS. En caso de realizar una comunicación síncrona, el rendimiento del servidor DNS se vería reducido en gran medida lo cual no es deseable por los tiempos de respuesta tan reducidos que se requieren. Por lo tanto, se hace uso de un mecanismo de comunicación asíncrona, tal y como se menciona en el apartado anterior. Mediante este mecanismo, se permite mantener una eficacia aceptable en términos de rendimiento en el servidor DNS, así como disponer de una solidez y seguridad apropiadas en las respuestas del servidor web de alerta.

En caso de que el servidor reciba una petición para alertar de una detección de acceso a una web maliciosa, se recibe como nombre de host el dominio detectado como malicioso. En este momento se dispone de dos casuísticas:

- Se recibe una petición con un nombre de host detectado como malicioso.

- Se recibe una petición con un nombre de host no detectado como malicioso pero que apunta a una dirección IP detectada como maliciosa.

En la primera casuística, el servidor web realiza una búsqueda a ElasticSearch y, en caso de encontrar el dominio como malicioso, genera una URL de *bypass*, utilizando el método descrito en el apartado anterior, la indexa en ElasticSearch y se la devuelve al cliente.
En el caso de la segunda casuística, el servidor web primero comprueba a qué IPs apunta el nombre de dominio leído del nombre de host con el que se realiza la petición. Haciendo uso de estas IPs, se realiza una búsqueda en ElasticSearch, y si alguna de ellas se encuentra como maliciosa, genera una URL de *bypass* y se la muestra al cliente, pero siempre haciendo uso del nombre de dominio para una mayor transparencia y claridad para el cliente.
En la Figura 7 se puede observar un ejemplo de alerta del servidor web de alertas.

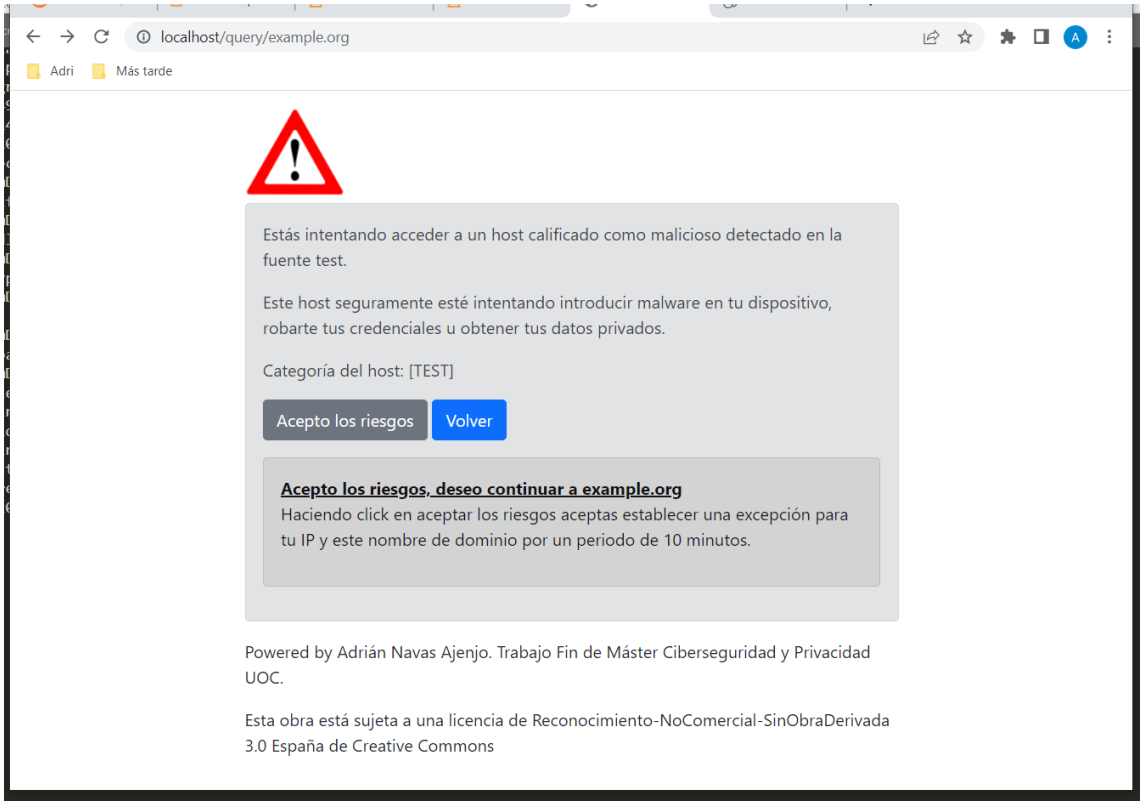

Figura 7. Ejemplo de alerta del servidor web de alertas.

En caso de recibir una petición al servidor web con un nombre de host no encontrado en ElasticSearch y cuyas IPs no se encuentran en los índices de maliciosas, se muestra un error 404 y se guarda un log de ese error identificando al cliente.
En el diseño del servidor web se plantea también la problemática de que un atacante trate de realizar un atacante de denegación de servicio realizando un número masivo de peticiones a URLs no detectadas como maliciosas, ni por dominio ni por IPs resultantes, o a URLs detectadas como maliciosas que general URLs de *bypass*. En el caso de que realizase peticiones correctas, además de un problema de denegación de servicio en el propio servidor, también podría realizar una denegación de servicio en ElasticSearch ya que, con cada petición correcta, se almacena una entrada de la URL de *bypass* en él.

Para evitar este riesgo, se hace uso de la herramienta fail2ban. Esta herramienta, a través del uso de filtros y políticas, comprueba cuando los clientes realizan acciones determinadas y, en base a umbrales en las políticas, cuando estos se superan por un mismo cliente, se le bloquea el acceso al sistema durante un tiempo determinado en la propia política. Este proceso se realiza mediante el uso de *iptables* de Linux.
Para el caso del servidor, se implementan dos filtros diferentes:

1. Cuando un cliente ha realizado más de 10 peticiones correctas, entendiendo correctas como peticiones a dominios o IPs maliciosas que efectivamente se encuentran en ElasticSearch, en un plazo de 1 minuto. Este filtro aplica un bloqueo de 60 minutos al cliente.
2. Cuando un cliente ha realizado más de 10 peticiones erróneas en un plazo de 1 minuto. Este filtro aplica un bloqueo de 60 minutos al cliente.

Dentro del diseño del servidor web se ha tenido también en cuenta la implementación de este a través de HTTPS para una mayor seguridad y confianza por parte de los clientes. Dada la naturaleza investigadora del proyecto y, bajo la premisa de hacer uso de recursos *open source* y gratuitos para él, se hace uso de un certificado firmado por la autoridad certificadora de *Let's Encrypt*. Se hace uso de esta entidad ya que, se trata de una entidad en la que confían la gran mayoría de navegadores web del mercado y, además, emite los certificados de manera gratuita con un periodo de validez de 30 días. Adicionalmente, la renovación de estos por periodos siempre de 30 días es gratuita también.
Al hacer uso de Apache para el enrutamiento de las peticiones a Flask, se configura el cifrado SSL mediante el uso del certificado de *Let's Encrypt* en él. Al estar dispuesto el servidor a través de un balanceador de carga, los tres servidores disponen del mismo certificado que está firmado para el nombre de dominio del balanceador de carga y a través del cual acceden los clientes.

3.7 Problemática de certificados SSL

Teniendo en cuenta que los navegadores web no modifican el parámetro "*Host*" original de las peticiones lanzadas, cuando un cliente realice una petición a un dominio detectado como malicioso, tanto por su nombre de dominio o por su IP resultante, el servidor DNS le devuelve la IP del servidor web de alerta. Cuando la petición llega al servidor web de alerta, este ofrece el certificado SSL obtenido mediante *Let's Encrypt* firmado para el dominio "firewalldns.systems" y como el dominio solicitado no coincide con el firmado, el cliente va a recibir un mensaje alertando de que dominio y certificado no coinciden.
De forma adicional, cuando los clientes utilicen el mecanismo de *bypass* para acceder a una URL detectada como maliciosa, la URL del "*Host*" de la petición HTTP sigue el formato *<bypass>-<id>.firewalldns.systems* y al resolverse con la IP del dominio real al que apunta, el certificado que se obtiene está firmado para el nombre de dominio del dominio real y el navegador web mostrará una alerta con esta disparidad de certificados.
Este comportamiento, por el lado de la seguridad de los navegadores es algo común y deseable ya que, solventa la problemática de hacer uso de servidores DNS maliciosos que traten de redireccionar a los clientes a páginas web no legítimas dónde poder robar sus datos. Pero del lado de la usabilidad del firewall DNS objeto de este trabajo, esto hace que la navegación sea compleja para el cliente ya que, para acceder al mensaje de alerta debe aceptar un

certificado problemático y en el caso de acceder a la página a través del método de *bypass*, también debe aceptar otro certificado problemático en el lado del navegador web. Este comportamiento incluso puede dar a la inoperatividad del firewall DNS en algunas compañías que, por política, tengan deshabilitada la opción de aceptar certificados SSL problemáticos.
El problema del certificado no coincidente con el nombre de "*Host*" al acceder a una alerta del servidor web de alertas no se puede solventar debido a que, por definición, la única manera de redireccionar a una web para mostrar la alerta sin utilizar programas de terceros y haciendo uso únicamente del protocolo DNS, es modificar la IP de un host malicioso por la IP del servidor web de alerta. Sin embargo, de cara a la aceptación de riesgos por parte de los clientes y su redirección a la web detectada como maliciosa, dónde con el método actual es necesario aceptar de nuevo un certificado no coincidente, existe una aproximación para solventar el error:
Es posible, mediante el uso de un índice auxiliar en ElasticSearch, añadir excepciones que mapean IP del cliente y nombre de dominio malicioso. Haciendo uso de este índice, en el lado del servidor DNS, antes de buscar si un nombre de dominio es malicioso, lanzar una búsqueda al índice de excepciones con este y con la IP del cliente que realiza la petición. En caso de que se encuentren en el índice de excepciones, el servidor DNS responde a la petición con la IP real del dominio detectado como malicioso.
Para hacer este proceso lo más transparente posible para el cliente y garantizar la seguridad del proceso, se puede hacer uso del ya implementado método de generación de URLs de *bypass*. Este proceso es:

1. El servidor web de alerta recibe una petición con una cabecera "*Host*" diferente de "firewalldns.systems".
2. Mediante *mod_rewrite* se reescribe la petición para que se envíe a firewalldns.systems/<Host original>.
3. Se comprueba que el host recibido como URI se encuentre en el índice de dominios maliciosos o que apunte a una IP detectada como maliciosa. En caso contrario se devuelve error.
4. Se genera una URL de *bypass* siguiendo el método detallado más atrás.
5. Se le muestra la alerta al cliente y se le indica que puede aceptar el riesgo haciendo clic en un enlace a esta URL de *bypass*.
6. En caso de hacer clic en ella, el servidor DNS resuelve la petición de forma normal, apuntando a la IP del servidor web de alerta. En este punto se modifica el comportamiento del servidor DNS que cuando recibe una URL de *bypass* ya no devuelve la IP real del dominio detectado como malicioso.
7. Cuando el servidor web de alerta recibe una petición con el formato de URL de *bypass*, en caso de que esta URL sea correcta y se encuentre indexada, añade una excepción en el índice de excepciones indicando el nombre de dominio y la IP del cliente y lanza una redirección 302 al dominio original. En este punto se modifica también el comportamiento del servidor DNS ya que, ahora antes de resolver cualquier petición, se añade un paso adicional mediante el cual comprueba si el dominio y cliente se encuentran en el índice de excepciones.

De esta forma, se consigue modificar el funcionamiento del *bypass* de dominios maliciosos sin necesidad de aceptar la alerta de certificado no coincidente por parte de los navegadores web en este paso.

Para establecer un periodo máximo de tiempo de las excepciones, se hace uso de los ciclos de vida de índices de ElasticSearch y se configura estos para que se mantengan en fase activa 1 minuto y pasen a fase de eliminación después de otros 9 minutos. De esta forma, las excepciones duran 10 minutos en total.
Tras esta modificación, el diagrama de flujo de una petición DNS al Firewall DNS es el correspondiente a la Figura 8 (flujo del lado del servidor DNS), Figura 9 (flujo del lado del servidor web de alerta al recibir una petición de alerta) y Figura 10 (flujo del lado del servidor web de alerta al recibir una petición de añadir excepción temporal).

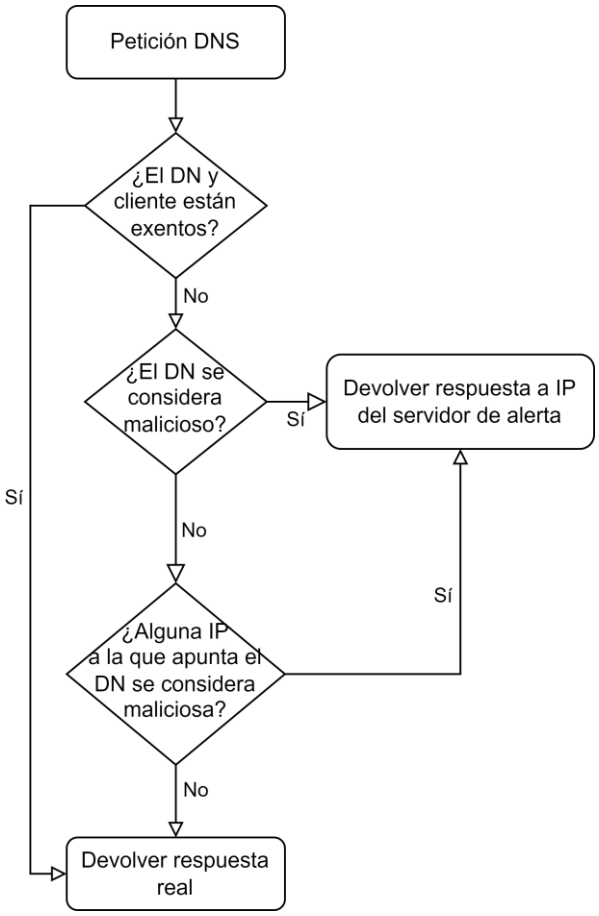

Figura 8. Diagrama flujo petición DNS con solución de certificado SSL.

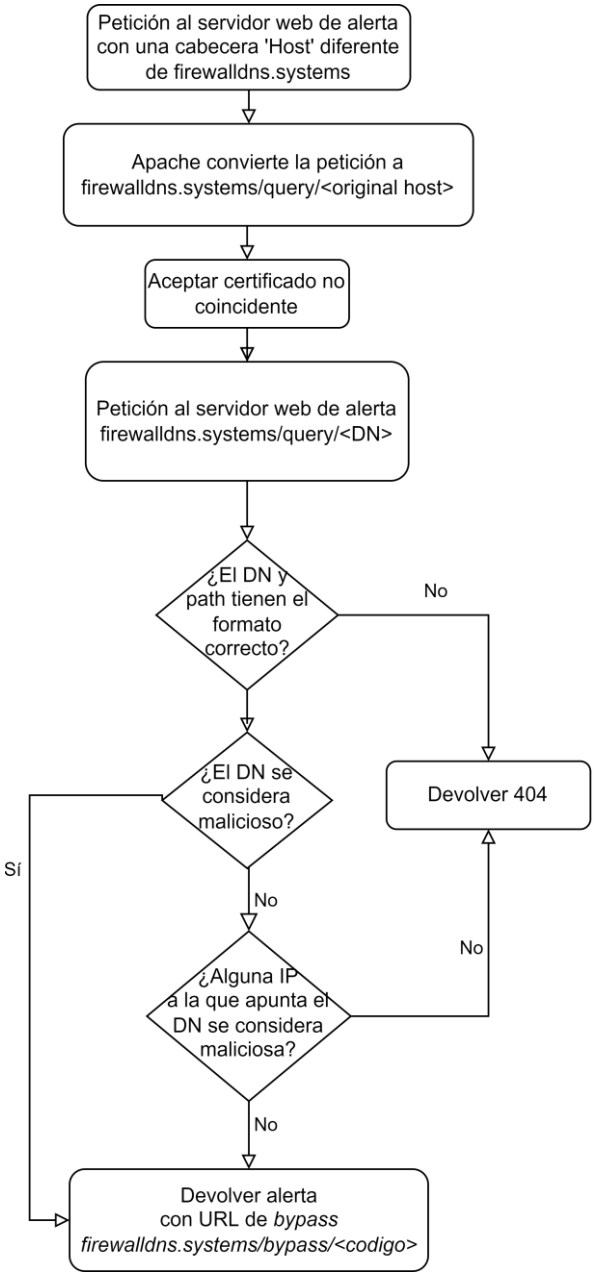

Figura 9. Diagrama flujo petición servidor web de alerta: Generación de URL de *bypass*.

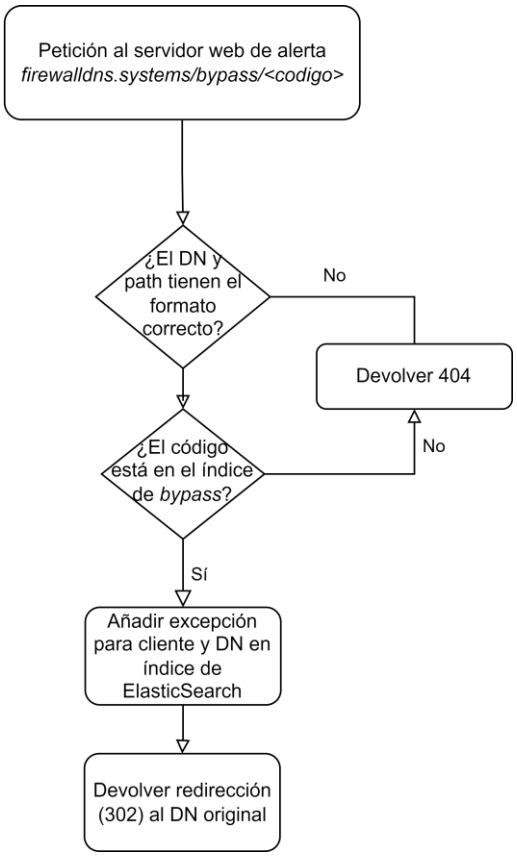

Figura 10. Diagrama flujo petición servidor web de alerta: Generación de excepción y redirección a DN malicioso.

3.8 Recogida de logs y métricas

Todos los desarrollos propios de la plataforma, Firewall DNS, servidor web de alerta y actualizador de datos de fuentes públicas, producen logs que recogen información útil sobre su ejecución, incluyendo datos útiles de información, avisos, errores e información de métricas. Estos logs se almacenan, todos haciendo uso de la librería *logging*, en un fichero en el directorio */var/log*. Estos logs y métricas se almacenan en ElasticSearch para su posterior posible visualización. Para la recogida y envío de logs desde los ficheros hasta ElasticSearch se hace uso de Metricbeat y Filebeat para la recogida y Logstash para el procesamiento y envío a ElasticSearch.

Filebeat se instala en cada uno de los servidores que dispone de ficheros de logs que se requieren recoger, recoge estos logs y los envía a Logstash para su filtrado y procesamiento antes de ser indexados en ElasticSearch.

Dentro de los grupos de plataformas y desarrollos, se recogen los siguientes logs y métricas:

- Firewall DNS: se recogen los siguientes logs del aplicativo del Firewall DNS:
  - Mensajes de inicio, conexiones y errores del servidor en escucha.
  - Mensajes de error conectando con ElasticSearch en las búsquedas.
  - Mensajes de error del uso general del aplicativo.
  - Mensajes de categoría CRITICAL haciendo uso de la API de ElasticSearch.

    - Mensajes de información sobre peticiones al servidor: peticiones en general, normales y anómalas y clientes y URLs que hacen uso del servicio de *bypass*.
    - Métricas de uso del servidor DNS con los tiempos de ejecución de cada una de las peticiones para su análisis de rendimiento.
    - Métricas de salud de los servicios Firewall DNS y Filebeat (recogidas con Metricbeat).
- Actualizador de fuentes públicas: se recogen los siguientes logs del actualizador de información:
    - Errores conectando o procesando fuentes públicas.
    - Actualización correcta de fuentes públicas.
    - Mensajes de categoría CRITICAL haciendo uso de la API de ElasticSearch.
- Servidor web de alerta: se recogen los siguientes logs del servidor web de alerta:
    - Conexiones correctas al servidor web.
    - Conexiones a rutas incorrectas del servidor web.
    - Mensajes de categoría CRITICAL haciendo uso de la API de ElasticSearch.
    - Logs de acceso y error del servidor web Apache utilizado para servir el servidor Flask.
    - Logs del aplicativo fail2ban utilizado para bloquear clientes clasificados como maliciosos.
    - Métricas de salud del servicio de Apache (recogidas con Metricbeat).

Una vez todas estas métricas se recogen, se envían a Logstash para su procesamiento y envío a los correspondientes índices de ElasticSearch tal y como se indica en la Tabla 9.

Logstash se instala en los mismos servidores que ElasticSearch y se accede a él a través del balanceador de carga. En Logstash se procesan los logs, y en función a su tipo y fuente, haciendo uso de un campo (*type*) enriquecido en Filebeat y Metricbeat, se envían a un índice u otro. Dentro del procesamiento, se extraen ciertos datos como el *timestamp*, el tipo de log, las IPs de los clientes, los tiempos de ejecución de las peticiones del servidor DNS o los mensajes en general para una mayor facilidad a la hora de realizar consultas, programar alertas y visualizar los datos.

De forma adicional, y para disponer de información más completa a la hora de visualizar las métricas y establecer las alertas, se hace uso de un filtro de Logstash para el enriquecimiento de información de geolocalización para todas las entradas de log que dispongan de IPs. Este enriquecimiento se hace a través del plugin *geoip*. Este filtro utiliza la base de datos GeoIP2 Lite [18] para enriquecer los logs que disponen de IP con su correspondiente geolocalización extraída de esta base de datos. Esta base de datos se actualiza automáticamente de forma diaria para una mayor precisión y dispone información sobre el país y estado (o provincia) específicos desde los que provienen los clientes.

Al estar Logstash desplegado en las mismas máquinas que ElasticSearch, todo el tráfico entre estos se realiza de forma local y no a través de un balanceador ya que los datos de logs ya están balanceados en su llegada a Logstash.

Adicionalmente, ElasticSearch divide estos logs y los replica en *shards* a lo largo de todo el clúster.

3.9 Diseño de la plataforma de métricas y alertas

Una vez seleccionada la solución de visualización, se despliega Grafana en un servidor único el cual hace uso de ElasticSearch como fuente de datos. Para el acceso a ElasticSearch, Grafana realiza peticiones contra el balanceador de carga interno, a través del cual se balancean las peticiones hacia todos los nodos maestro de ElasticSearch.

3.9.1 Métricas recolectadas

Se han estudiado diferentes métricas útiles para el desarrollo de este trabajo y para los operadores de seguridad que operen el producto final de este. Teniendo en cuenta que se trata de un producto de seguridad, se consideran útiles las métricas sobre intentos y accesos a IPs y dominios maliciosos, permitiendo así observar qué clientes tratan de conectar diferentes veces a páginas maliciosas.
Más en detalle, y haciendo uso de los logs y las métricas recogidas detalladas en el capítulo anterior, las visualizaciones creadas son las siguientes:

1. Rendimiento del servidor DNS.
2. Dominios maliciosos más accedidos.
3. Disponibilidad del Firewall DNS, del servidor web, de los propios servidores, de los Filebeat de todos los servidores, de ElasticSearch.
4. Clientes bloqueados por fail2ban junto con mapa.
5. Mapa de peticiones normales de clientes.
6. Lista de dominios *bypass*.
7. Lista de clientes que hacen *bypass*.
8. Mapa de IPs y localización a las que se hacen *bypass*.
9. Métricas de los servidores incluyendo uso de CPU, de memoria RAM y de disco.

3.9.2 Paneles de control y gráficos

Dentro de Grafana, se despliegan los siguientes paneles y gráficos para la visualización de datos:

- Panel general de control del producto, como ejemplo en la Figura 11 y Figura 12, que contiene las siguientes métricas:
    - Peticiones al servidor DNS.
    - Mapa de clientes que realizan peticiones al servidor DNS.
    - Mapa de clientes que realizan peticiones al servidor web de alertas.
    - Nombres de dominio consultados más comunes.
    - Tiempo medio de respuesta de las peticiones del servidor DNS.
    - Nombres de dominio maliciosos consultados más comunes.
    - Nombres de dominio con *bypass* más comunes por cliente.
    - Clientes bloqueados por fail2ban por servidor.

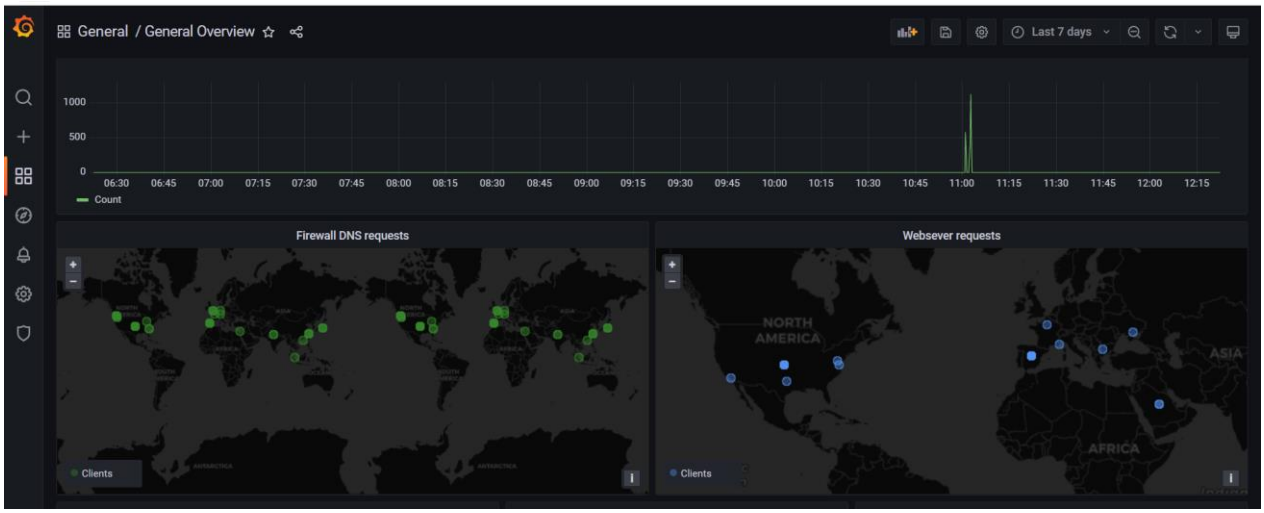

Figura 11. Panel de control general del producto 1.

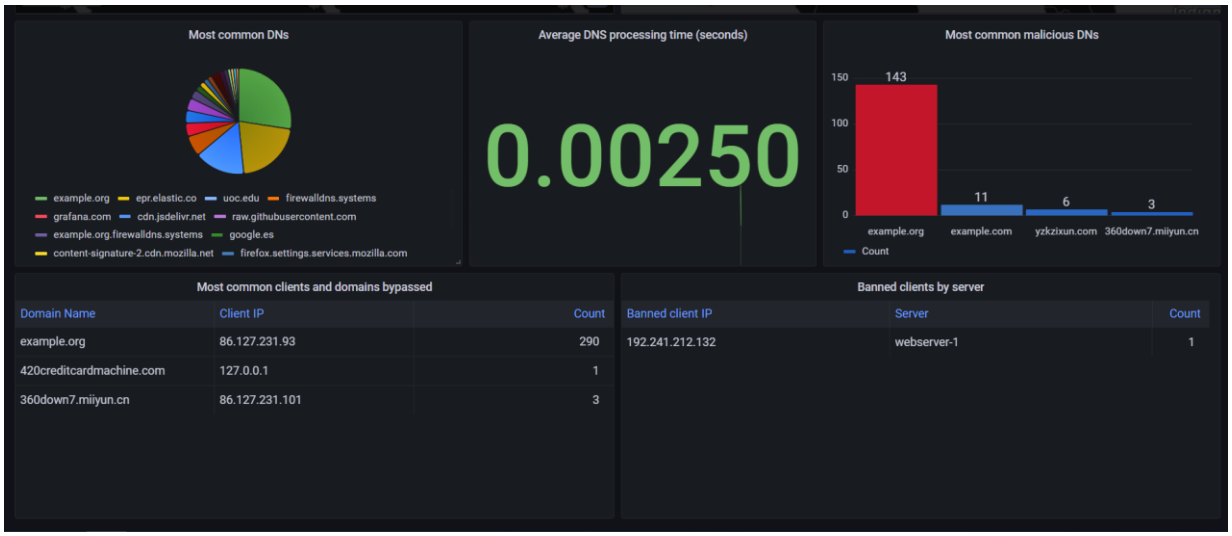

Figura 12. Panel de control general del producto 2.

- Panel de control de monitorización general con datos de Metricbeat, como ejemplo en la Figura 13 con datos de servidores web. Basado en el panel propuesto en [19] y que contiene las siguientes métricas:
  - Carga de CPU del sistema normalizada (por el número de núcleos).
  - Estado actual del servidor.
  - Número de núcleos del servidor.
  - Memoria total del servidor.
  - Espacio en disco total del servidor.
  - Porcentaje de uso de CPU en base al tiempo de inactividad (*idle*) en formato serie de tiempo.
  - Porcentaje de uso de CPU en base al tiempo de inactividad (*idle*) actual.
  - Uso de CPU en base a las métricas *user*, *iowait* y *system* en serie de tiempo.
  - Porcentajes de uso de CPU y memoria en serie de tiempo.
  - Uso de memoria actual.
  - Media de uso de núcleos.

- Uso de disco total.
- Tráfico de red entrante y saliente.

Figura 13. Panel de control de métricas de servidores web.

- Panel de control de monitorización de disponibilidad general de servicios, como ejemplo en la Figura 14, y que contiene las siguientes métricas:
  - Gráfico general de disponibilidad de los servicios monitorizados.
  - Disponibilidad del servidor DNS.
  - Disponibilidad del servidor web de alertas.
  - Disponibilidad de Filebeat en todos los servidores.
  - Disponibilidad de Grafana.
  - Estado actual de los servicios: FirewallDNS, Filebeat, Servidor web de alertas y Grafana.

Figura 14. Panel de control de disponibilidad de servicios.

3.9.2 Alertas generadas

Mediante el uso de la herramienta de alertas nativa de Grafana, se crean las siguientes alertas:

- Servicio de Firewall DNS no activo por un periodo de 5 minutos. Ejecutada cada minuto.
- Servicio de Apache para el servidor web de alertas no activo por un periodo de 5 minutos. Ejecutada cada minuto.
- Servicio de Filebeat en cualquier servidor no disponible por un periodo de 5 minutos. Ejecutada cada minuto.
- Cliente baneado del servidor firewall DNS o del servidor web de alertas.

Para el envío externo de estas alertas se hace uso de un *bot* de Telegram creado con la utilidad nativa de Telegram, *BotFather*, y un grupo específico del que únicamente es miembro dicho *bot* y un usuario que recibe las alertas.

3.10 Alta disponibilidad y buenas prácticas de seguridad

Como se puede apreciar en el diseño general de la Figura 4, y en los diseños de esta capitulo, el diseño está planteado para garantizar en todo momento la alta disponibilidad y el correcto funcionamiento de toda la plataforma, a excepción de la plataforma de visualización de métricas Grafana.

Adicionalmente, para el despliegue y configuración de todos los elementos se utilizan buenas prácticas de seguridad de arquitectura tales como:

- Todas las máquinas virtuales disponen de su correspondiente firewall (mediante el uso de *Azure Network Security Groups*) con los puertos cerrados por defecto, y únicamente expuestos los puertos necesarios a los orígenes necesarios.
- Las máquinas disponen de acceso a Internet deshabilitado por defecto, a excepción de las máquinas del Firewall DNS y Grafana. Para el acceso al servidor web se hace uso de tráfico interno de Azure entre el balanceador y los servidores.
- El balanceador de carga interno, no se conecta a Internet y solo permite tráfico entre los grupos de servidores web y firewall DNS y Logstash o ElasticSearch.
- El acceso SSH a todas las máquinas se realiza a través de una de las máquinas del grupo de servidores DNS utilizada a modo máquina de salto.
- Todos los accesos a través de SSH se realizan haciendo uso de par de claves, nunca con usuario y contraseña y sin permitir inicio de sesión directo al usuario *root*.
- Las máquinas disponen de una copia de seguridad de sus discos realizada de forma periódica con el fin de prevenir posibles ataques de denegación de servicio.

4. Implementación y pruebas

La implementación de este trabajo se realiza mediante dos vías: mediante una implementación local para el desarrollo de los componentes y pruebas iniciales y mediante el despliegue en la nube de Microsoft, Azure, para las pruebas de un entorno simulado con alta disponibilidad.

4.1 Implementación local

Se realiza una primera implementación en local para las primeras fases de codificación y pruebas. Todo este despliegue se ha realizado sobre Windows mediante el uso de cuadernos de Jupyter y contenedores Docker con *Docker for Desktop*.

Para las primeras fases de codificación y pruebas, se hace uso de cuadernos de Jupyter con el motor de Python 3 para el desarrollo de los aplicativos: servidor DNS, servidor web, actualizador de fuentes y pruebas de aprendizaje automático para la detección de *phishing* debido a su gran versatilidad para ejecutar código y realizar pruebas intermedias.

Para la disposición de la **plataforma de almacenamiento**, se hace uso de un contenedor de ElasticSearch ejecutado en local en el cual se crean los índices, plantillas de índices, políticas de ciclo de vida y *api keys* necesarios. Para una mayor facilidad de gestión de este ElasticSearch, se hace uso de un contenedor de Kibana, conectado a este ElasticSearch, a través del cual se han podido visualizar y consultar datos de una forma más sencilla.

Tras disponer de ElasticSearch y todos sus componentes necesarios, se realizan las pruebas de los desarrollos que contactan con este a través del uso del api *key* creada para las pruebas.

Tras estas pruebas con cuadernos de Jupyter, y una vez validados todos los desarrollos, se procede a comentar todas las clases y métodos desarrollados para un mejor mantenimiento futuro y, además, se ha configurado la librería *argsparse* en cada uno de los desarrollos para permitir una parametrización completa de todos los desarrollos y sus parámetros de configuración. De forma adicional, en el desarrollo del actualizador de datos de fuentes públicas, se configura la lectura de un fichero de configuración a través del cual se procesan las fuentes a utilizar y todos los parámetros que estas necesitan como su identificador, categoría, URL de la cual se descargan los datos y expresión regular que se utiliza para procesar cada una de las entradas de estos. El uso de esta expresión regular permite la recogida de datos de cualquier fuente independientemente de la sintaxis que tengan siempre y cuando el formato sea texto plano separado en líneas.

En esta fase de pruebas en local se hace uso de un contenedor de Logstash para la configuración de la recogida de logs de los aplicativos y su procesamiento para la indexación de ElasticSearch. Debido a las limitaciones de Docker for Desktop con respecto al mapeo de volúmenes con carpetas locales de Windows, para las pruebas de esta recogida de logs se hace uso de los aplicativos para la generación de sus ficheros de logs y, posteriormente, se ha lanzado Logstash y se le han inyectado estos ficheros de forma artificial en una ruta local para simular el proceso.

De forma adicional, y haciendo uso del contenedor de ElasticSearch, se ha creado otro contenedor de Grafana, conectado a ElasticSearch, y mediante el uso de las métricas recolectadas, se han creado los paneles de control, gráficas y alertas que se han diseñado para el aplicativo completo. En el caso

de la implementación local, las alertas se configuran sin ningún punto de contacto para el envío de estas fuera de la plataforma.

4.2 Implementación en Azure

Tras la implementación y pruebas en local, se realiza una implementación en Azure para simular toda la infraestructura que dispondría el producto completo, en alta disponibilidad, tal y como se detalla en la Figura 4.

Para ello, se despliegan todos los servidores necesarios en una misma red y en dos subredes, una para el firewall DNS y los servidores Apache y otra para el clúster de ElasticSearch, Logstash y la plataforma de métricas.

Para el balanceo de las cargas se hace uso de un balanceador de carga público, que balancea el tráfico del servidor DNS y el servidor web de alertas y otro balanceador privado que balancea el tráfico entrante a Logstash y ElasticSearch.

Para la disposición del **firewall DNS**, se introduce el ejecutable de Python con el desarrollo de este y se ha creado un servicio de systemd, que lo ejecuta haciendo uso de los parámetros pertinentes y que dispone de inicio automático con la ejecución del servidor. De esta forma, el servidor DNS es consultable a través del balanceador de carga, por cualquiera de sus servidores de backend.

Dentro de uno de los servidores del firewall DNS se introduce también el ejecutable que actualiza los datos de fuentes públicas junto con el fichero de configuración pertinente que contiene todas las fuentes utilizadas en este proyecto. Para la actualización de la base de datos, junto con el uso de las políticas de ciclo de vida en el lado de ElasticSearch, se configura una entrada de *crontab* para que este proceso se ejecute diariamente y, de esta forma, disponer de las fuentes actualizadas sin saturar su actualización.

Para la plataforma de **servidores web de alerta**, se hace uso de Apache como proxy intermedio entre las peticiones y Flask. Este se utiliza debido a que el servidor implementado por Flask no está recomendado por su propia documentación para entornos productivos, solo para pruebas. Para ello, se hace uso del módulo *mod_wsgi* para la ejecución de fondo del ejecutable del servidor web de alertas.

Debido al diseño comentado tras la modificación del apartado 3.7 Problemática de certificados SSL, este servidor debe recoger peticiones con dos formatos diferentes:

- Peticiones con un nombre de host diferente de *firewalldns.systems*.
- Peticiones con el nombre de host *firewalldns.systems*.

En el caso de recibir una petición con un nombre de host diferente a *firewalldns.systems* estamos en el caso de una petición que está siendo redirigida por una solicitud de resolución de un dominio detectado como malicioso, por lo que se devuelve una respuesta a la URI de alertas mediante una redirección en el propio Apache.

En el caso de recibir una petición con nombre de host *firewalldns.systems*, la petición puede ser o bien a la URI de alertas, “/query/”, o bien una petición a la URI de bypass, “/bypass/”.

Para la separación de estas respuestas, se hace uso de dos *VirtualHosts* que separan lógicamente las respuestas recibidas. El primero, recibe las peticiones entrantes con el nombre de host del nombre de dominio del firewall DNS y el segundo utiliza un comodín para actuar de sumidero y recolectar todas las peticiones que entren con un nombre de host diferente.

Para la plataforma de **recogida y almacenamiento de logs**, se hace uso de tres servidores que disponen de Logstash y ElasticSearch formando un clúster de tres nodos con roles maestros y de datos.
En todos los servidores implementados, se instala Filebeat para la recogida de logs de los ficheros específicos de cada aplicativo y Metricbeat para la recogida de métricas de salud de los servidores en general y de los servicios de los aplicativos correspondientes. Estos binarios envían la información a Logstash para su procesamiento e ingesta en los índices correspondientes en ElasticSearch a través del uso del balanceador de carga privado.
De forma adicional, los desarrollos interactúan directamente con ElasticSearch para realizar las consultas sobre dominios e IPs maliciosas y excepciones, así como para la ingesta de datos de fuentes públicas actualizados diariamente.
Se instala también en los servidores web de alertas el aplicativo **fail2ban** con la configuración correspondiente para bloquear a los clientes detectados como maliciosos siguiendo las directivas para ello del diseño de estos servidores.
Para la plataforma de **visualización de métricas y alertas**, se hace uso de un servidor con Grafana. Este, se enlaza con ElasticSearch de forma directa para la consulta de datos y se introducen en él todas las visualizaciones y alertas detalladas en el apartado de diseño y probadas en la fase previa de pruebas locales.
En esta fase, se crea un *bot* de Telegram que se utiliza para el envío de las alertas generadas en Grafana. Todas las alertas que se generan se reenvían utilizando este bot a una conversación grupal de Telegram.

4.3 Pruebas

Tras disponer de toda la infraestructura desplegada en Azure, se realizan pruebas contra la plataforma para el estudio de las métricas y funcionalidades de esta.
Una de las principales observaciones realizadas en la fase de estudio es la baja latencia que disponen los servidores DNS. Para establecer una línea base a la que atenerse, se realiza una prueba contra algunos DNS públicos y comúnmente utilizados: Google, OpenDNS, Cloudflare y Telefónica. Mediante un script, y la herramienta *dig*, se extraen los tiempos de resolución del dominio “uoc.edu” y se promedian para la obtención de los datos de rendimiento tal y como se puede observar en la Tabla 10.

Servidor DNS	Tiempo de respuesta promedio (en milisegundos)
Google (8.8.8.8)	12.4
OpenDNS (208.67.222.222)	37.9
Cloudflare (1.1.1.1)	12.4
Quad9 (9.9.9.9)	16.2

Tabla 10. Rendimientos de servidores DNS comunes.

Debido a esta observación, durante el desarrollo del trabajo se ha tenido en cuenta el rendimiento del servidor DNS como punto importante. Tras el desarrollo del firewall DNS, se observa que, son necesarias dos búsquedas en ElasticSearch, una para comprobar si el nombre de dominio es malicioso y otra para comprobar si alguna de las IPs a las que apunta es maliciosa. De forma

adicional, haciendo uso de la funcionalidad de *bypass*, es necesario realizar una tercera búsqueda en ElasticSearch.
El motivo de interés en reducir el número de búsquedas a ElasticSearch es que estas se realizan a través de peticiones HTTP las cuales disponen de rendimientos menores que conexiones realizadas directamente por TCP.
Dado que la última búsqueda es opcional, solo en el caso de querer implementar la opción de hacer *bypass* de dominios detectados como maliciosos, se estudian los rendimientos del firewall DNS con y sin esta opción, siguiendo el mismo mecanismo que el estudio con los servidores genéricos, tal y como se puede observar en la Tabla 11.

Configuración del firewall DNS	Tiempo de respuesta promedio (en milisegundos)
Con opción de *bypass*	148.2
Sin opción de *bypass*	136.2

Tabla 11. Rendimientos del firewall DNS.

Tal y como se puede observar, los tiempos de respuesta no varían en gran medida en función de si se utiliza la funcionalidad de *bypass* o no.

Tras disponer de toda la plataforma funcionando correctamente, se hace uso de una máquina virtual desplegada en local mediante VirtualBox en Windows para configurar como servidor DNS el firewall DNS y realizar las pruebas. Para esta máquina virtual se hace uso de un Kali Linux con interfaz gráfica.
Para una mayor sencillez en estas pruebas y no interactuar directamente con servidores maliciosos, se inyecta dentro de la base de datos de dominios maliciosos la URL “example.org” y dentro de la base de datos de IPs maliciosas la IP a la cual redirige la URL “example.com” que es 93.184.216.34.
Para una primera prueba sobre la funcionalidad básica del firewall DNS, se hace uso de la herramienta dig en la máquina virtual Linux desplegada y configurada contra el firewall DNS. Tal y como se puede apreciar en la Figura 15, cuando se lanza una petición para un dominio normal, en este caso google.es, se devuelve una respuesta normal.
Tal y como se puede apreciar en la Figura 16 y en la Figura 17, cuando se lanzan peticiones a dominios maliciosos, example.org, y a dominios que apuntan a IPs maliciosas, example.com, se devuelve una respuesta que contiene el CNAME, a modo informativo, apuntando al servidor web de alertas y cuya IP resuelta apunta al balanceador de carga que balancea las peticiones del servidor web de alertas.

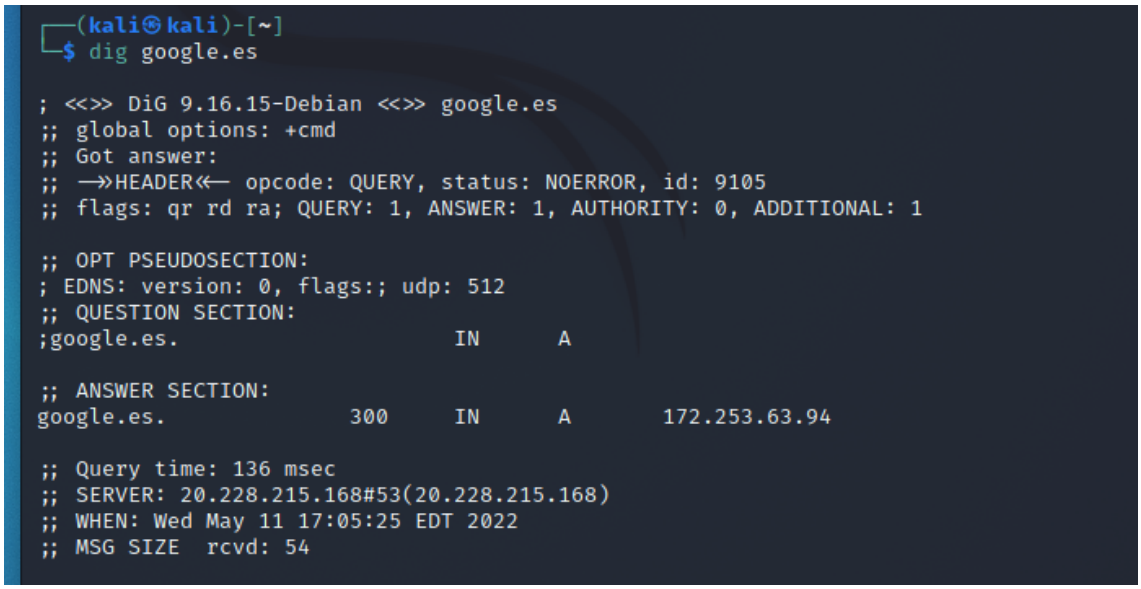

Figura 15. Prueba del servidor DNS con dig a un dominio normal.

Figura 16. Prueba del servidor DNS con dig a un dominio malicioso.

Figura 17. Prueba del servidor DNS con dig a un dominio con IP maliciosa.

De esta forma, cuando se hace uso de un navegador web para consultar a un dominio malicioso, en este caso con una petición a https://example.org, tal y como se puede observar en la Figura 18, obtenemos una redirección al servidor web de alertas mostrando la alerta de este dominio y dando la opción al cliente de aceptar los riesgos y continuar al dominio añadiendo una excepción.

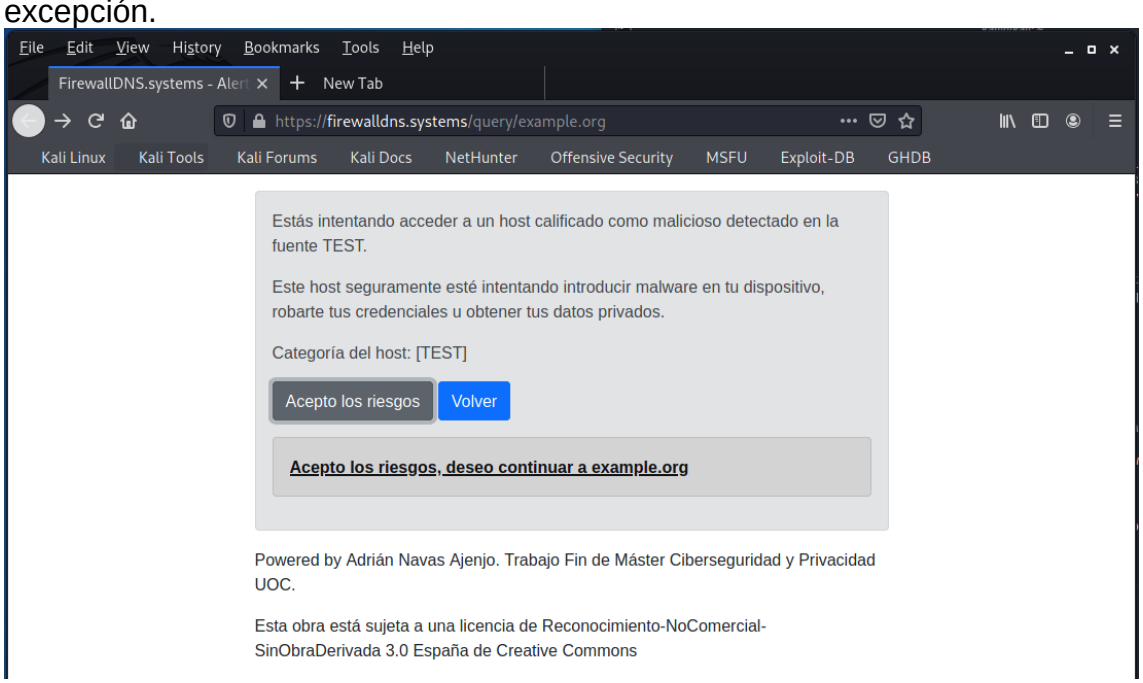

Figura 18. Alerta del servidor web al acceder a un dominio malicioso.

En este proceso de pruebas se observa un comportamiento que merma en gran medida la funcionalidad de añadir excepciones en el firewall DNS. Todos los sistemas operativos disponen de cachés de DNS que almacenan en su caché las resoluciones obtenidas. Por lo tanto, al realizar una primera resolución de un dominio detectado como malicioso, en estas cachés se mapeará el dominio correspondiente con la IP del servidor web de alertas y, tras crear una excepción, al tratar de volver a resolver el dominio, la petición no se redirige al firewall DNS y vuelve a apuntar al servidor de alertas, lo cual genera un bucle continuo de alertas.
Otra casuística problemática observada es la de las cachés de los navegadores web. En el caso de solventar el problema de la caché DNS del sistema operativo, o incluso del navegador web si dispone de ella, cuando se realiza una petición a un dominio malicioso y se devuelve la alerta, el contenido de esta es cacheado por el navegador web y, cuando el DNS devuelve la IP real del dominio tras crear la excepción, el navegador carga de igual manera la página de alerta. De igual modo, cuando la caché expira y se puede acceder a la web real, al expirar la excepción a los 10 minutos, el navegador web continúa mostrando el contenido de la web ya que está cacheado. En las pruebas realizadas, se observa que la caché por defecto de Mozilla Firefox es de 60 segundos.

Para el proceso de pruebas, se expone la plataforma de forma pública durante un periodo de 24 horas para la obtención de datos reales de uso con el fin de comprobar los paneles de control.

De forma adicional, se prueban las alertas mediante el apagado del servicio Apache del servidor web de alertas. Tras unos minutos, se comprueba que la alerta se genera efectivamente en Grafana tal y como se puede observar en la Figura 19. También se observa cómo en la Figura 14 el servicio se muestra caído y su disponibilidad ha disminuido en gran medida.

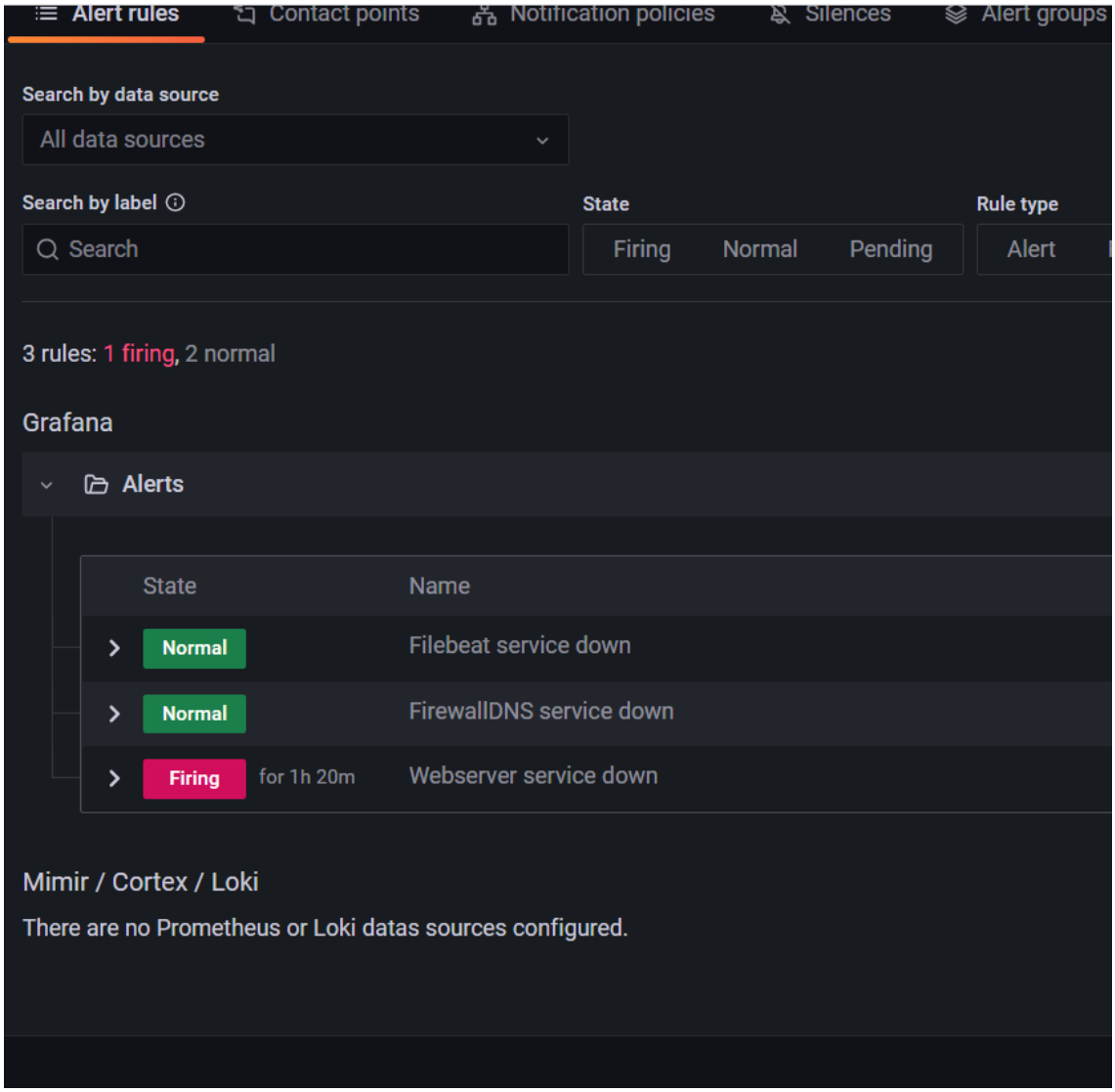

Figura 19. Prueba de alerta con servidor web.

Se comprueba también el grupo de Telegram creado para la recepción automática de alertas generadas en Grafana mediante el uso del *bot* de Telegram y, tal y como se puede apreciar en la Figura 20, se observa que la alerta se recibe correctamente. Tras reiniciar el servicio de nuevo, se puede apreciar en la Figura 21 cómo se recibe de nuevo una alerta indicando que el problema se ha resuelto.

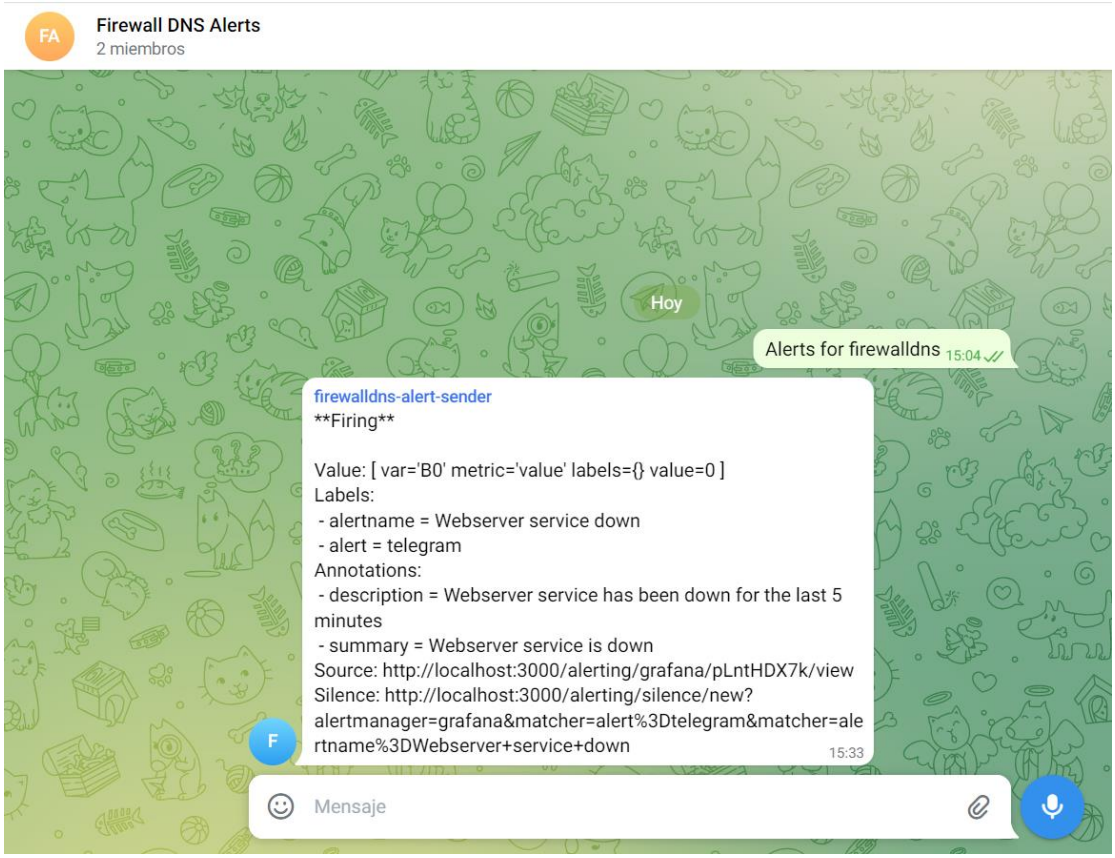

Figura 20. Prueba de alerta en Telegram a través de Grafana y *Bot* de Telegram.

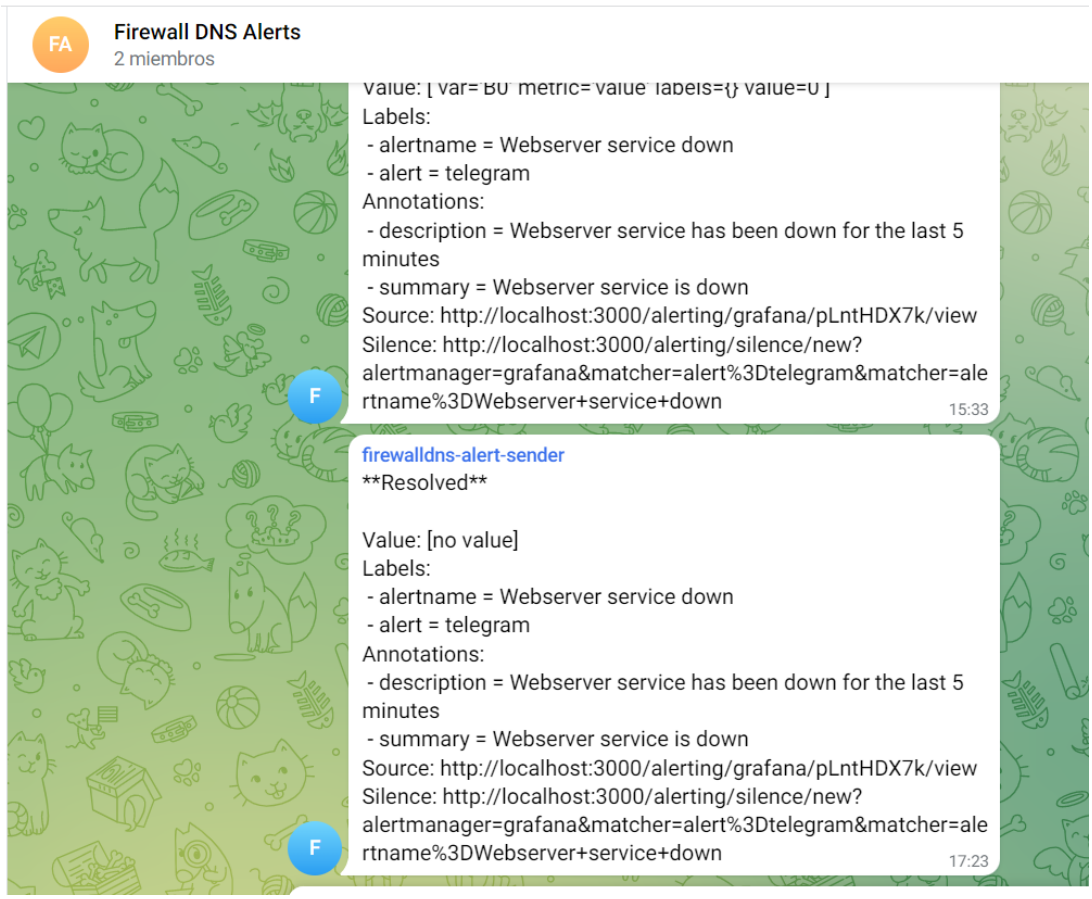

Figura 21. Prueba de alerta resuelta en Telegram a través de Grafana y Bot de Telegram.

5. Conclusiones

5.1 Conclusiones del trabajo

Tras la realización del trabajo, se puede concluir principalmente que la elaboración de un sistema de Firewall DNS es viable incluso mediante el uso de desarrollos propios. Existe un gran número de fuentes abiertas de información, gracias a la gran comunidad tanto empresarial como individual que existe, de las cuales se pueden extraer estos dominios e IPs maliciosos para bloquear accesos indebidos en cualquier ámbito.

Si bien es cierto, también se puede concluir que la implementación y uso de un servidor web de alertas para alertar de accesos a sitios maliciosos y permitir excepciones de los considerados falsos positivos no es algo trivial y que presenta diferentes problemáticas como las mencionadas a lo largo de este trabajo: problemática con los certificados SSL y problemática con las cachés DNS y de los navegadores web. Debido a estas problemáticas, se considera necesario un estudio más profundo y dedicado de este tema para tratar de encontrar una solución, quizás con cierta integración con algún navegador web, que elimine estas problemáticas. Se plantea como trabajo futuro debido a que se considera muy interesante el uso de esta funcionalidad para el correcto funcionamiento en ámbitos empresariales principalmente.

De este trabajo se han extraído conocimientos adicionales principalmente en cuanto al funcionamiento y diseño de servidores DNS, pero también de arquitectura en la nube e integración de datos y codificación en Python.

Otra conclusión que se extrae del trabajo es la importancia de la implementación de sistemas en alta disponibilidad para su uso principalmente en ámbito empresarial y la gran ayuda que prestan los servicios en la nube para la implementación de arquitecturas de este tipo. Gracias a sus servicios gestionados y configurados en alta disponibilidad como bases de datos o balanceadores de carga permiten delegar gran parte de la alta disponibilidad en ellos.

Con respecto a la prueba de concepto del uso de un motor de aprendizaje automático para la detección de dominios provenientes de *phishing*, los datos arrojados por esta prueba preliminar no lo hacen una utilidad a implementar dentro del producto final debido a que las tasas de falsos positivos y falsos negativos, aunque no demasiado altas, son lo suficientemente altas como para mermar el correcto funcionamiento de este. De forma adicional, tras el estudio del estado del arte actual de este tipo de motores de detección y extractores de características, se observa que en la gran mayoría de ellos se hace uso de características provenientes también de datos de las peticiones HTTP. Esto hace que en los servidores DNS, dónde solo se dispone de la información del nombre de dominio, cabecera *Host* en peticiones HTTP, exista una merma de información lo cual hace que no sean motores idóneos para aplicar en este tipo de servidores.

Tras el estudio e implementación de la plataforma de métricas, se puede concluir que la monitorización se trata de un proceso muy útil en infraestructuras con cierto nivel de criticidad y, además, no es un proceso complicado de implementar debido a la gran comunidad *open source* y empresarial que existe. Gracias a esta comunidad, existen muchos productos para la recopilación, procesamiento y análisis de los diferentes datos de métricas e integraciones nativas para su visualización y alertado.

5.2 Reflexión crítica

Tal y como se puede observar, todos los objetivos de estudio se han logrado correctamente extrayendo un estado del arte muy útil e informativo para la realización del trabajo.

En cuanto al diseño e implementación, se han cumplido todos los objetivos a excepción de los correspondientes con la plataforma de alertas. Estos objetivos solo se han podido cumplir de forma parcial debido a las problemáticas mencionadas previamente con respecto a los certificados SSL y a las cachés DNS y de navegadores web. Debido a estas limitaciones debidas al diseño externo de protocolos, HTTPS, y de productos, navegadores web como Google Chrome o Mozilla Firefox, la utilidad se ha podido implementar correctamente y es funcional, pero es necesaria la realización de dos tareas intermedias que complican su aplicación diaria: aceptación de certificados no coincidentes y dependencia de caducidad de cachés DNS y web de los navegadores.

Sin embargo, al considerarse una funcionalidad tan importante se propone para trabajo futuro su estudio en profundidad.

5.3 Análisis crítico del seguimiento de la planificación y metodología

Realizando el seguimiento general del trabajo, la planificación se ha ido siguiendo correctamente ya que todos los hitos y fechas se han cumplido en el plazo estipulado.

Con respecto a la metodología propuesta, ha sido adecuada y se ha seguido también durante todo el desarrollo del trabajo.

Sí que ha sido necesario introducir un cambio a lo largo de la elaboración del trabajo ya que, en la fase final de diseño e inicio de las pruebas se ha detectado un problema en la implementación que ha sido necesario corregir y modificar para que el producto continuase siendo funcional. Este cambio es el método de redirección de las peticiones al servidor web de alertas dada la problemática de los certificados SSL no coincidentes ya comentada. En un inicio se planteó hacer uso de CNAME, pero posteriormente se averiguó que los navegadores web no hacen uso de estos, si no que mantienen el parámetro *Host* de la petición HTTP original. Para solventar este punto, se modificó el enrutamiento en el servidor web para recoger correctamente los nombres de dominio mediante el uso de redirecciones permanentes del protocolo HTTP.

5.4 Trabajo futuro

A lo largo de este trabajo se han detectado ciertas líneas de trabajo que no entraban en la planificación y objetivos, pero que se considera muy interesante su estudio para la mejora del producto y su rendimiento. Estos puntos encontrados son los siguientes:

- Implementación con Terraform o Ansible para un despliegue completamente automático en entornos empresariales.
- Migración a Docker en un *stack* sencillo para el despliegue en casa (Docker Compose y Kubernetes).
- Creación de una plataforma colaborativa de fuentes.
- Estudio de un modelo de ML para añadir funcionalidad estudiada en capítulo de estudio de viabilidad.
- Estudio de una solución para evitar el problema del certificado no coincidente accediendo al servidor web de alertas.

- Estudio de una solución para evitar el problema de cachés tanto DNS como de navegadores web.

6. Glosario

DNS: Domain Name Server
RR: Resource Record
CTI: Cyber Threat Intelligence
TNR: True Negative Rate
TPR: True Positive Rate

5. Bibliografía

[1] Infoblox Firewall DNS Datasheet. (2017). Infoblox. Recuperado 24 de febrero de 2022, de https://www.infoblox.com/wp-content/uploads/infoblox-datasheet-dns-firewall.pdf

[2] EfficientIP. (2020, 19 mayo). DNS Firewall Protege a los usuarios y bloquea la actividad de malware basado en DNS. Recuperado 24 de febrero de 2022, de https://www.efficientip.com/es/productos/dns-firewall/

[3] NavyTitanium (Apodo). (2018). Setting up a DNS Firewall on steroids. DNSMasterChef. Recuperado 24 de febrero de 2022, de https://navytitanium.github.io/DNSMasterChef/

[4] Marques, C., Malta, S., & Magalhães, J. (2021). DNS firewall based on machine learning. Future Internet, 13(12), 309. doi:http://dx.doi.org/10.3390/fi13120309

[5] Mockapetris, P. (noviembre 1987). Domain Concepts and Facilities. IETF RFC-1034 https://datatracker.ietf.org/doc/html/rfc1034.

[6] Mockapetris, P. (noviembre 1987). Domain Names – Implementation and Specification. IETF RFC-1035 https://datatracker.ietf.org/doc/html/rfc1035.

[7] PaulC (Apodo). (2022). Dnslib - A Python library to encode/decode DNS wire-format packets. Recuperado 14 de marzo de 2022, de https://github.com/paulc/dnslib.

[8] Shackleford. D. (febrero 2015). Who's Using Cyberthreat Intelligence and How?, A SANS Survey. SANS Institute. https://cdn-cybersecurity.att.com/docs/SANS-Cyber-Threat-Intelligence-Survey-2015.pdf

[9] Conti M., Dargahi T., Dehghantanha A. (2018) Cyber Threat Intelligence: Challenges and Opportunities. In: Dehghantanha A., Conti M., Dargahi T. (eds) Cyber Threat Intelligence. Advances in Information Security, vol 70. Springer, Cham. https://doi.org/10.1007/978-3-319-73951-9_1

[10] Jatana, N., Puri, S., Ahuja, M., Kathuria, I., & Gosain, D. (2012). A survey and comparison of relational and non-relational database. International Journal of Engineering Research & Technology, 1(6), 1-5.

[11] Abunadi, A., Akanbi, O., Zainal, A. (2013) Feature extraction process: A phishing detection approach. 13th International Conference on Intellient Systems Design and Applications, 2013, pp. 331-335, doi: 10.1109/ISDA.2013.6920759.

[12] Xuan, C., Nguyen, H., Nikolaevich, T. (2020) Malicious URL detection based on machine learning. (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 11, No. 1, 2020, pp. 148-153.

[13] Marchal, S. (Creator) (2014). PhishStorm - phishing / legitimate URL dataset. Aalto University. urlset(v.zip). 10.24342/f49465b2-c68a-4182-9171-075f0ed797d5

[14] Jing, Y., Zhang, C., Wang X. (2009). An Empirical Study on Performance Comparison of Lucene and Relational Database. 2009 International Conference on Communication Software and Networks, 2009, pp. 336-340, doi: 10.1109/ICCSN.2009.96.

[15] Ribenzaft, D. (2021, 4 marzo). Open-Source Monitoring Tools: In-Depth Comparison. Recuperado 13 de marzo de 2022, de https://epsagon.com/tools/open-source-monitoring-tools-comparison/
[16] Cloud Native Cloud Foundation. CNCF Cloud Native Interactive Landscape. Recuperado 13 de marzo de 2022, de https://landscape.cncf.io/
[17] Elastic.co. “ElasticSearch 8.1 _id field”. Recuperado 11 de abril de 2022, de https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-id-field.html
[18] Max Mind, GeoIP2 Lite Free Geolocation Data. Recuperado 18 de abril de 2022 de https://dev.maxmind.com/geoip/geolite2-free-geolocation-data?lang=en.
[19] Grafana Labs, System Metrics Dashboard for Grafana. Recuperado 16 de mayo de 2022 de https://grafana.com/grafana/dashboards/2030.

6. Anexos

1. *Pruebas de extracción de características y clasificación de URLs de phishing*

En este código se describen las pruebas realizadas para el estudio de la implementación del motor de detección de phishing por aprendizaje automático. Las pruebas se han realizado en Jupyter Notebook y exportado a un fichero Python para su inserción en este documento.

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import re
import time
import random

class PhishingFeatureExtraction():
    def __init__(self):
        pass

    def domain_length(domain):
        return len(domain)

    def dot_number(domain):
        return len(domain.split(".")) - 1

    def subdomain_levels(domain):
        return len(domain.split(".")) - 3

    def domain(domain):
        return domain

    def dash_number(domain):
        return len(domain.split("-")) - 1

    def number_count(domain):
        return sum(char.isdigit() for char in domain)

    def sensitive_words_count(domain):
        s_words =
["secure","account","login","banking","confirm","bank","pass",
                  "bitcoin","coin"]
        counts = [1 for word in s_words if word in domain]
        return sum(counts)

    def brand_name(domain):
```

```
        brands = ["amazon", "santander", "bbva", "paypal", "ing",
"caixa", "adidas",
                  "nike", "facebook", "instagram", "tiktok",
"twitter", "whatsapp",
                  "youtube", "google", "gmail", "vodafone",
"movistar", "apple",
                  "samsung", "microsoft"]
        for brand in brands:
            if brand in domain:
                return True
        return False

    def extract_features(self,url,features):
        entry = {}
        for feature in features:
            entry[feature] = getattr(PhishFinder, feature)(url)
        return entry

    def create_dataset(self,domains):
        total = []
        features =
['domain','dot_number','subdomain_levels','domain_length','dash_number
',

'number_count','sensitive_words_count','brand_name']
        for idx,domain in enumerate(domains):
            total.append(self.extract_features(domain.lower(),
features))
        return pd.DataFrame(total,columns=features)

#--------------------------------------------------------------------
# In[2]:


import pandas as pd
import re
df = pd.read_csv('urlset.csv',error_bad_lines=False)

domains = []
labels = []
regex = re.compile(r"([\w\d\-\.]+)\/?.*")
for idx,row in df.iterrows():
    try:
        domains.append(re.search(regex,row["domain"]).group(1))
        labels.append(row["label"])
    except:
        pass

df = pd.DataFrame({"domain":domains,"label":labels})
```

```
df = df.drop_duplicates()

df_phishing = df[df['label']==1]
df_no_phishing = df[df['label']==0]
p = PhishingFeatureExtraction()
df_mal = p.create_dataset(list(df_phishing['domain']))
df_good = p.create_dataset(list(df_no_phishing['domain']))

#----------------------------------------------------------------------
# In[3]:


df_mal["label"] = 1
df_good["label"] = 0
df_original = pd.concat([df_mal,df_good])

#----------------------------------------------------------------------
# In[4]:


from sklearn import tree
from sklearn.tree import *
from sklearn.model_selection import train_test_split
from sklearn import metrics
import numpy as np

#----------------------------------------------------------------------
# In[5]:


def classifier_metrics(clf,clf_name,df):
    df = df_original.copy()
    y = df["label"]
    df = df.drop(["domain","label"],axis=1)
    X = df.values

    tns,fps,fns,tps,accuracys,recalls,rocs = [],[],[],[],[],[],[]
    for i in range(0,100):
        X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42)

        clf = clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)

        tn, fp, fn, tp = metrics.confusion_matrix(y_test,
y_pred).ravel()
        tns.append(tn)
        fps.append(fp)
        fns.append(fn)
```

```
        tps.append(tp)
        accuracys.append(metrics.accuracy_score(y_test,y_pred))
        recalls.append(metrics.recall_score(y_test,y_pred))
        fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred,
pos_label=1)
        rocs.append(metrics.auc(fpr, tpr))

    print("----------------------------------------")
    print(clf_name)
    print("----------------------------------------")
    print("Confusion Matrix")
    print(np.mean(tns),np.mean(fps),np.mean(fns),np.mean(tps))
    print("Accuracy",np.mean(accuracys))
    print("Recall",np.mean(recalls))
    print("ROC AUC",np.mean(rocs))

#---------------------------------------------------------------------
# In[6]:


clf = RandomForestClassifier(n_jobs=-1)
classifier_metrics(clf,str(clf),df_original)


'''Output
----------------------------------------
RandomForestClassifier(n_jobs=-1)
----------------------------------------
Confusion Matrix
5961.8 1349.2 2585.1 5461.9
Accuracy 0.7438273212657895
Recall 0.6787498446626072
ROC AUC 0.7471030033051785
'''
```

2. *Script de actualización de datos de IPs y dominios maliciosos de fuentes abiertas*

El script correspondiente es el siguiente. En él se aprecia una clase "*SourcesUpdater*" que es la encargada de toda la gestión. Para llamar a este script se recogen los argumentos con la librería "*argsparse*".
Al introducir sus datos en ElasticSearch, este script requiere que el servidor de ElasticSearch recibido esté correctamente configurado. También se requiere que las credenciales de la API que se utilicen tengan permisos de lectura y escritura sobre los índices utilizados.
Adicionalmente, para ello se hace uso de ciertas políticas de gestión del ciclo de vida de los índices para la rotación y eliminación automática de datos antiguos. La configuración de estas políticas a través de la API de ElasticSearch se puede observar después del código del script.

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import requests
import re
from elasticsearch import Elasticsearch
import time
import datetime
import configparser
import base64
import warnings
warnings.filterwarnings("ignore")
import argparse

import logging.config

logging.basicConfig(level=logging.INFO,format="%(asctime)s
[%(levelname)s] - %(message)s",
    handlers=[logging.FileHandler("sourcesUpdater.log")])

class SourcesUpdater():
    '''
    This class defines the sources updating process to create a new
index with the new data fetched in ElasticSearch.
    '''
    def __init__(self, api_key, sources_file="sources.conf",
es_host="https://localhost:9200", index_prefix="malicious"):
        '''
        SourcesUpdater Class constructor

        Parameters:
```

```
            api_key (Tuple(string,string)): the tuple containing the
api_key for the Elasticsearch
                connection base64-encoded.
            sources_file (string): the sources configuration file.
Each entry must have name, url, regex,
                type and category keys. Defaults: sources.conf
            es_host (string): the host of ElasticSearch in
protocol://host:port format. Default: "https://localhost:9200"
            index_prefix (string): the prefix for the sources data
indexing.

        '''
        self.date = datetime.datetime.today().strftime('%Y%m%d')
        self.index_prefix = index_prefix
        config = configparser.ConfigParser()
        config.read(sources_file,"UTF-8")
        # Elasticsearch configuration
        api_key_decoded = (base64.b64decode(api_key[0]).decode(),
base64.b64decode(api_key[1]).decode())
        self.__connection =
Elasticsearch(es_host,verify_certs=False,api_key=api_key_decoded)
        # Logger configuration
        self.logger = logging.getLogger(__name__)
        # Variable initialization
        self.documents = []
        self.document_idx = 0
        self.sources =  []
        # Sources dictionary creation
        for k,v in config.items():
            if k != "DEFAULT":
                # Only take configuration entries with all the
required parameters
                if set(v.keys()) ==
set(["type","url","category","regex"]):
                    source_dict = dict(v.items())
                    source_dict["name"] = k
                    self.sources.append(source_dict)
                else:
                    self.logger.error("Error adding source %s. It does
not have all the required parameters" % k)

    def __add_sources(self,value,type_key,name,category):
        '''
        This function adds the value received in the self.documents
list formatted for Elasticsearch bulk request.

        Parameters:
            value (string): the value to add.
            type_key (string): the type of the registry to add.
```

```
            name (string): the name of the source.
            category (string): the category of the source.

        '''
        document_id = int(len(self.documents)/2) + 1 +
self.document_idx
        index_name = self.index_prefix + "-" + type_key
        self.documents.append({"index": {"_index": index_name, "_id":
document_id}})
        self.documents.append({"@timestamp":int(time.time()*1000),
type_key:value,"source_name":name,"category":category})

    def update_sources(self):
        '''
        This function updates the sources content in Elasticsearch
        It updates all the data in the alias for each index
        '''
        for source in self.sources:
            url = source['url']
            regex = source['regex']
            type_key = source['type']
            name = source['name']
            category = source['category']
            try:
                data = requests.get(url)
                text = data.text.split("\n")
            except Exception as e:
                self.logger.error("Error downloading or processing %s
data: %s" % (name,e))
                continue

            for line in text:
                match = re.search(regex,line)
                if(match != None):

self.__add_sources(match.group(type_key),type_key,name,category)
            try:
                self.__connection.bulk(
self.documents,request_timeout=60)
                self.logger.info("Updated source: %s" % name)
            except Exception as e:
                self.logger.error("Error updating source %s: %s" %
(name,e))
            self.document_idx += len(self.documents)
            self.documents.clear()
        self.document_idx = 0


#-----------------------------------------------------------------------
```

```
# In[2]:

# Example of script arguments recover and sources updating


parser = argparse.ArgumentParser(description='Update malicious IP and
domain sources ')
parser.add_argument('-i', '--api_id', help="API Key ID base64
encoded",required=True)
parser.add_argument('-k', '--api_secret', help="API Key Secret Key
base64 encoded",required=True)
parser.add_argument('-f', '--file', help="Sources configuration
file",default="sources.conf")
parser.add_argument('-e', '--elasticsearch', help="ElasticSearch API
host in format: <protocol>://<host>:<port>",
                        default="https://localhost:9200")

args = parser.parse_args()
api_key_encoded = (args.api_id,args.api_secret)
m = SourcesUpdater(api_key=api_key_encoded, sources_file=args.file,
es_host=args.elasticsearch)
m.update_sources()
```

Las peticiones a realizar en ElasticSearch a través de consultas a su API para la configuración de los índices y políticas de ciclo de vida son los siguientes. Estos se deben realizar de forma secuencial en el orden provisto. Se hace uso del siguiente script para su creación:

```
# ILM policies
curl -X PUT -H "Content-Type: application/json" -u user:redacted -k
https://localhost:9200/_ilm/policy/malicious-data \
-d '
{
  "policy": {
    "phases": {
      "hot": {
        "min_age": "0ms",
        "actions": {
          "set_priority": {
            "priority": 100
          },
          "rollover": {
            "max_primary_shard_size": "10gb",
            "max_age": "1d"
          }
        }
      },
      "delete": {
        "min_age": "7d",
        "actions": {
```

```
          "delete": {
            "delete_searchable_snapshot": true
          }
        }
      }
    }
  }
}'

curl -X PUT -H "Content-Type: application/json" -u user:redacted -k
https://localhost:9200/_ilm/policy/webserver-bypass-data \
-d '
{
  "policy": {
    "phases": {
      "hot": {
        "min_age": "0ms",
        "actions": {
          "set_priority": {
            "priority": 100
          },
          "rollover": {
            "max_primary_shard_size": "1gb",
            "max_age": "10m"
          }
        }
      },
      "delete": {
        "min_age": "5m",
        "actions": {
          "delete": {
            "delete_searchable_snapshot": true
          }
        }
      }
    }
  }
}'

curl -X PUT -H "Content-Type: application/json" -u user:redacted -k
https://localhost:9200/_ilm/policy/exceptions-data \
-d '
{
  "policy": {
    "phases": {
      "hot": {
        "min_age": "0ms",
        "actions": {
          "set_priority": {
```

```
            "priority": 100
          },
          "rollover": {
            "max_primary_shard_size": "1gb",
            "max_age": "1m"
          }
        }
      },
      "delete": {
        "min_age": "9m",
        "actions": {
          "delete": {
            "delete_searchable_snapshot": true
          }
        }
      }
    }
  }
}'

# Index template for malicious domain
curl -X PUT -H "Content-Type: application/json" -u user:redacted -k
https://localhost:9200/_index_template/malicious-domain \
-d '
{
  "template": {
    "settings": {
      "index": {
        "lifecycle": {
          "name": "malicious-data",
          "rollover_alias": "malicious-domain"
        },
        "number_of_shards": "1",
        "number_of_replicas": "1"
      }
    },
    "mappings": {
      "properties": {
        "@timestamp": {
          "index": true,
          "ignore_malformed": false,
          "store": false,
          "type": "date",
          "doc_values": true
        }
      }
    }
  },
  "index_patterns": [
```

```
    "malicious-domain-*"
  ]
}'

curl -X PUT -H "Content-Type: application/json" -u user:redacted -k
https://localhost:9200/_ilm/policy/general-logs-data \
-d '
{
  "policy": {
    "phases": {
      "hot": {
        "min_age": "0ms",
        "actions": {
          "set_priority": {
            "priority": 100
          },
          "rollover": {
            "max_primary_shard_size": "10gb",
            "max_age": "1d"
          }
        }
      },
      "delete": {
        "min_age": "89d",
        "actions": {
          "delete": {
            "delete_searchable_snapshot": true
          }
        }
      }
    }
  }
}'

curl -X PUT -H "Content-Type: application/json" -u user:redacted -k
https://localhost:9200/_ilm/policy/general-metrics-data \
-d '
{
  "policy": {
    "phases": {
      "hot": {
        "min_age": "0ms",
        "actions": {
          "set_priority": {
            "priority": 100
          },
          "rollover": {
            "max_primary_shard_size": "10gb",
            "max_age": "1d"
```

```
          }
        }
      },
      "delete": {
        "min_age": "29d",
        "actions": {
          "delete": {
            "delete_searchable_snapshot": true
          }
        }
      }
    }
  }
}'

# Index template for malicious IP
curl -X PUT -H "Content-Type: application/json" -u user:redacted -k
https://localhost:9200/_index_template/malicious-ip \
-d '
{
  "template": {
    "settings": {
      "index": {
        "lifecycle": {
          "name": "malicious-data",
          "rollover_alias": "malicious-ip"
        },
        "number_of_shards": "1",
        "number_of_replicas": "1"
      }
    },
    "mappings": {
      "properties": {
        "@timestamp": {
          "index": true,
          "ignore_malformed": false,
          "store": false,
          "type": "date",
          "doc_values": true
        }
      }
    }
  },
  "index_patterns": [
    "malicious-ip-*"
  ]
}'

# Index template for webserver bypass request IDs
```

```
curl -X PUT -H "Content-Type: application/json" -u user:redacted -k
https://localhost:9200/_index_template/webserver-bypass \
-d '
{
  "template": {
    "settings": {
      "index": {
        "lifecycle": {
          "name": "webserver-bypass-data",
          "rollover_alias": "webserver-bypass"
        },
        "number_of_shards": "1",
        "number_of_replicas": "1"
      }
    },
    "mappings": {}
  },
  "index_patterns": [
    "webserver-bypass-*"
  ]
}'

# Index template for DNS exceptions
curl -X PUT -H "Content-Type: application/json" -u user:redacted -k
https://localhost:9200/_index_template/exceptions \
-d '
{
  "template": {
    "settings": {
      "index": {
        "lifecycle": {
          "name": "exceptions-data",
          "rollover_alias": "exceptions"
        },
        "number_of_shards": "1",
        "number_of_replicas": "1"
      }
    },
    "mappings": {}
  },
  "index_patterns": [
    "exceptions-*"
  ]
}'

curl -X PUT -H "Content-Type: application/json" -u user:redacted -k
https://localhost:9200/_index_template/general-logs \
-d '
{
```

```
  "template": {
    "settings": {
      "index": {
        "lifecycle": {
          "name": "general-logs-data",
          "rollover_alias": "general-logs"
        },
        "number_of_shards": "1",
        "number_of_replicas": "1"
      }
    },
    "mappings": {
      "properties": {
        "@timestamp": {
          "index": true,
          "ignore_malformed": false,
          "store": false,
          "type": "date",
          "doc_values": true
        }
      }
    }
  },
  "index_patterns": [
    "general-logs-*"
  ]
}'

curl -X PUT -H "Content-Type: application/json" -u user:redacted -k
https://localhost:9200/_index_template/general-metrics \
-d '
{
  "template": {
    "settings": {
      "index": {
        "lifecycle": {
          "name": "general-metrics-data",
          "rollover_alias": "general-metrics"
        },
        "number_of_shards": "1",
        "number_of_replicas": "1"
      }
    },
    "mappings": {
      "properties": {
        "@timestamp": {
          "index": true,
          "ignore_malformed": false,
          "store": false,
```

```
          "type": "date",
          "doc_values": true
        }
      }
    }
  },
  "index_patterns": [
    "general-metrics-*"
  ]
}'

# Initialize index
curl -X PUT -H "Content-Type: application/json" -u user:redacted -k
https://localhost:9200/malicious-domain-000001 \
-d '
{
  "aliases": {
    "malicious-domain": {
      "is_write_index": true
    }
  }
}'

curl -X PUT -H "Content-Type: application/json" -u user:redacted -k
https://localhost:9200/malicious-ip-000001 \
-d '
{
  "aliases": {
    "malicious-ip": {
      "is_write_index": true
    }
  }
}'

curl -X PUT -H "Content-Type: application/json" -u user:redacted -k
https://localhost:9200/webserver-bypass-000001 \
-d '
{
  "aliases": {
    "webserver-bypass": {
      "is_write_index": true
    }
  }
}'

curl -X PUT -H "Content-Type: application/json" -u user:redacted -k
https://localhost:9200/exceptions-000001 \
-d '
{
```

```
  "aliases": {
    "exceptions": {
      "is_write_index": true
    }
  }
}'

curl -X PUT -H "Content-Type: application/json" -u user:redacted -k
https://localhost:9200/general-logs-000001 \
-d '
{
  "aliases": {
    "general-logs": {
      "is_write_index": true
    }
  }
}'

curl -X PUT -H "Content-Type: application/json" -u user:redacted -k
https://localhost:9200/general-metrics-000001 \
-d '
{
  "aliases": {
    "general-metrics": {
      "is_write_index": true
    }
  }
}'
```

3. *Fichero de configuración de fuentes de IPs y dominios maliciosos.*

```
##########################################
#
# Sources configuration file
# Each entry must have [<source name>] and
# url, regex, type and category parameters
#
##########################################

[IPsum]
url =
https://raw.githubusercontent.com/stamparm/ipsum/master/ipsum.txt
regex=(?P<ip>[0-9]{0,3}\.[0-9]{0,3}\.[0-9]{0,3}\.[0-9]{0,3}).*
type = ip
category = malicious

[URLhaus]
url = https://urlhaus.abuse.ch/downloads/hostfile/
regex=[0-9]{0,3}\.[0-9]{0,3}\.[0-9]{0,3}\.[0-9]{0,3}\t(?P<domain>.+)
type = domain
category = malicious

[mitchellkrogza/Phishing.Database Phishing IPs]
url =
https://github.com/mitchellkrogza/Phishing.Database/raw/master/phishin
g-IPs-ACTIVE.txt
regex=(?P<ip>[0-9]{0,3}\.[0-9]{0,3}\.[0-9]{0,3}\.[0-9]{0,3})
type = ip
category = phishing

[mitchellkrogza/Phishing.Database Phishing Domains]
url =
https://raw.githubusercontent.com/mitchellkrogza/Phishing.Database/mas
ter/phishing-domains-ACTIVE.txt
regex=(?P<domain>.+)
type = domain
category = phishing

[OpenPhish]
url = https://openphish.com/feed.txt
regex = https?\://(?P<domain>[a-zA-Z0-9\-
\.ßàÁâãóôþüúðæåïçèõöÿýòäœêëìíøùîûñé]+).*
type = domain
category = phishing

[NoTracking/hosts-blocklists Domain]
url = https://raw.githubusercontent.com/notracking/hosts-
blocklists/master/domains.txt
```

```
regex = address=/(?P<domain>[a-zA-Z0-9\-
\.ßàÁâãóôþüúðæåïçèõöÿýòäœêëìíøùîûñé]+)/.+
type = domain
category = ads tracker

[NoTracking/hosts-blocklists Hostname]
url = https://raw.githubusercontent.com/notracking/hosts-
blocklists/master/hostnames.txt
regex=[0-9]{0,3}\.[0-9]{0,3}\.[0-9]{0,3}\.[0-9]{0,3} (?P<domain>[a-zA-
Z0-9\-\.ßàÁâãóôþüúðæåïçèõöÿýòäœêëìíøùîûñé]+)
type = domain
category = ads tracker

[Dan.me.uk Tor Nodes]
url = https://www.dan.me.uk/torlist/
regex=(?P<ip>[0-9]{0,3}\.[0-9]{0,3}\.[0-9]{0,3}\.[0-9]{0,3})
type = ip
category = tor nodes

[FireHOL Bitcoin]
url = https://github.com/firehol/blocklist-
ipsets/raw/master/bitcoin_nodes.ipset
regex=(?P<ip>[0-9]{0,3}\.[0-9]{0,3}\.[0-9]{0,3}\.[0-9]{0,3})
type = ip
category = cryptominer

[FireHOL Coinbl]
url = https://github.com/firehol/blocklist-
ipsets/raw/master/coinbl_ips.ipset
regex=(?P<ip>[0-9]{0,3}\.[0-9]{0,3}\.[0-9]{0,3}\.[0-9]{0,3})
type = ip
category = cryptominer
```

4. Código del Servidor DNS

```
import socket
from _thread import *
import threading
import sys
from dnslib import *
import random
import logging
import time
import re
from elasticsearch import Elasticsearch
import warnings
import argparse
import base64
import logging.config

# Ignore ElasticSearch TLS warnings
warnings.filterwarnings("ignore")

class FirewallDNSServer():
    '''
    This class defines a Firewall DNS server.
    It uses dnslib library to handle DNS queries and several sources to
detect malicious requests.
    '''

    def __init__(self,host='localhost',port=53,webserver="example.org",

webserver_dns="127.0.0.53",forward_servers=["127.0.0.53"],forward_timeout=
0.5,
                 api_key="api_key_b64",es_host="https://localhost:9200"):
        '''
        FirewallDNSServer class constructor

        Parameters:
            host (string): Hostname of the DNS server. Default:
'localhost'
            port (int): Port of the DNS server. Default: 53
            webserver (string): Sink web server domain name. Default:
"example.org"
            webserver_dns (string): Web server main DNS server. Default:
"127.0.0.53" (internal Linux DNS)
            forward_servers (list): Forward DNS servers for requests.
Default: ["127.0.0.53"]
            forward_timeout (float): Timeout in seconds for query
forwards. Default: 0.5
            api_key (tuple(string,string)): api_key values for
ElasticSearch connection
```

```
            es_host (string): ElasticSearch host in format
protocol://host:port. Default: "https://localhost:9200"
        '''
        # Logger configuration
        self.logger = logging.getLogger(__name__)
        # Server configuration
        self.host = host
        self.port = port
        # Webserver parameters configuration
        self.webserver = webserver
        self.webserver_dns = webserver_dns
        self.webserver_regex = re.compile(r"(?P<id>.+)\." + webserver)
        # DNS forward configuration
        self.forward_servers = forward_servers
        self.forward_timeout = forward_timeout
        # ElasticSearch connection configuration
        api_key_encoded = args.api_key
        api_key_decoded =
base64.b64decode(api_key_encoded).decode().split(":")
        api_key_decoded = (api_key_decoded[0],api_key_decoded[1])
        self.__es_connection =
Elasticsearch(es_host,verify_certs=False,api_key=api_key_decoded)
        logging.getLogger('elasticsearch').setLevel(logging.CRITICAL)

    def malicious_request(self,record,subdomain,client,ip=None):
        '''
        This function returns the CNAME answer to a request detected as
malicious (to a malicious domain or IP).
        The DNS answer contains the CNAME to
<malicious_request>.<webserver_url> (i.e.
maliciousdomain.com.sinkwebserver.org)
        which shows the client a message alerting him that he is trying to
access to a known malicious site.
        If the malicious request is found by a malicious IP, the CNAME
returned should be:
            <malicious_request>.<malicious_ip>.<webserver_url>
        The DNS answer contains also the A response to the web server IP.

        Parameters:
            record (DNSRecord): the DNSRecord object registered from the
client request.
            subdomain (string): the malicious domain name where the client
is trying to access.
            client (string): the client IP that has sent the DNS query.
            ip (string): if the malicious request is found for a malicious
IP, its related IP, None if not. Default: None
        Returns:
            reply (DNSRecord): the DNSRecord object that contains the
reply with CNAME and A RRs.
```

```
        '''
        webserver_name = subdomain
        if ip != None:
            webserver_name += "." + ip
        webserver_name += "." + self.webserver
        ws_req = DNSRecord.question(self.webserver)
        if
len(DNSRecord.parse(ws_req.send(dest=self.webserver_dns,port=53,timeout=se
lf.forward_timeout)).rr) == 0:
            self.logger.error("Error in malicious_request. Webserver DN
not found in webserver DNS")
            return None
        ws_record =
DNSRecord.parse(ws_req.send(dest=self.webserver_dns,port=53,timeout=self.f
orward_timeout)).rr[0]
        ws_record.rname = webserver_name
        ws_record.ttl = 1
        reply = record.reply()

reply.add_answer(RR(subdomain,QTYPE.CNAME,rdata=CNAME(webserver_name),ttl=
1))
        reply.add_answer(ws_record)

        self.logger.warning("Client %s has made a request to a know
malicious site (%s)" % (client,subdomain))
        return reply

    def malicious_resource(self,resources,type_name):
        '''
        This function checks if one of the IPs or domains received is
malicious.

        Parameters:
            resources (list): the list of resources to check.
            type_name (string): the type of the resources to check ("ip"
or "domain")

        Returns:
            (boolean,string): returns (True,<resource>) if a resource is
malicious or (False,None) if not.
        '''
        for resource in resources:
            try:
                search = self.__es_connection.search(index="malicious-
"+type_name+"-*",
                                query={"match":
{type_name+".keyword":resource}})
                if search['hits']['total']['value'] > 0:
                    return True,resource
```

```
            except ConnectionError as e:
                self.logger.error("Error connecting with ElasticSearch
(malicious_resource): %s" % e)
        return False,None

    def forward_dns_query(self,record):
        '''
        This function tries to resolve the record question from the class
forward servers.
        It fetches randomly each server until an answer is retrieved or a
NXDOMAIN is thrown.

        Parameters:
            record (DNSRecord): the record with the question.

        Returns:
            (DNSRecord): returns the response if found and NXDOMAIN if
not.
        '''
        # It uses randomly different DNS forward servers to find the IP of
the domain
        forward_servers = self.forward_servers.copy()
        random.shuffle(forward_servers)
        for dns_server in forward_servers:
            try:
                answer =
DNSRecord.parse(record.send(dest=dns_server,port=53,timeout=self.forward_t
imeout))
                # 10 minutes TTL for the case that the domain is bypassed
                answer.ttl = 600
                return answer
            except:
                self.logger.warning("Server %s timed out" % dns_server)
        # In case of NXDOMAIN in any seed server
        nx_reply = record.reply()
        nx_reply.header.rcode = getattr(RCODE,'NXDOMAIN')
        return nx_reply

    def resolve_host(self,record,client_ip):
        '''
        This function resolves a host. If the host is malicious it returns
a response pointing to a sink web server
        thar warns the client. If the host is not, it returns the record
with the real answer.

        Parameters:
            record (DNSRecord): the DNSRecord object with the DNS request.
            client_ip (string): the IP address of the client.
```

```
        Returns:
            (DNSRecord): returns DNSRecord with the reply of the query.
        '''
        domain_name = record.q.get_qname().idna()[:-1]
        # Check if the domain is escaped from sink webserver
        try:
            search = self.__es_connection.search(index="exceptions",
                        query={"bool":{"must":[
                            {"match":{"domain":domain_name}},
                            {"match":{"client":client_ip}}]}})
            if search['hits']['total']['value'] > 0:
                answer = self.forward_dns_query(record)
                self.logger.info("Client %s bypassed access to %s" %
(client_ip,domain_name))
                return answer
        except ConnectionError as e:
            self.logger.error("Error connecting with ElasticSearch
(resolve_host): %s" % e)

        # Check if the domain name is marked as malicious
        malicious_domain,domain =
self.malicious_resource([domain_name],"domain")
        if malicious_domain:
            return self.malicious_request(record,domain_name,client_ip)
        # Check if the resolved IP(s) is(are) marked as malicious
        answer = self.forward_dns_query(record)
        ips = [str(a.rdata) for a in answer.rr]
        if len(ips) > 0:
            malicious_ip,ip = self.malicious_resource(ips,"ip")
            if malicious_ip:
                return
self.malicious_request(record,domain_name,client_ip,ip=ip)
            else:
                # Good response
                return answer
        else:
            # NXDOMAIN response
            return answer

    def handle_request(self,data,address_port):
        '''
        Thread main function to handle client requests.
        It checks also the execution time to log it.
        It exits the thread after completion.

        Parameters:
            data (bytes): the data received from the client.
            address_port (tuple(string,string)): the address,port tuple
with the client information
```

```
        '''
        begin_time = time.time_ns()
        address,port = address_port

        self.logger.info("Received request from %s:%s" % (address,port))
        self.logger.debug(data)
        record = DNSRecord.parse(data)

        try:
            domain_name = record.q.get_qname().idna()[:-1]
        except:
            self.logger.error("Error processing the request from %s: %s" %
(address,data))
        response = self.resolve_host(record,address)
        # If response is none, the webserver resolution has problems.
        if response == None:
            self.logger.warning("Closing thread as webserver is not being
resolved")
            sys.exit()
        self.sock.sendto(response.pack(), (address,port))
        end_time = time.time_ns()
        execution_time = (end_time-begin_time)/10e9
        if(response.header.get_rcode() == getattr(RCODE,"NOERROR")):
            # Only log the time if the response was sucessful
            self.logger.info("[METRICS] Request for %s obtained in %.20f
seconds"
                             % (domain_name,execution_time))
        else:
            self.logger.error("Error resolving %s" % domain_name)

        # Close thread
        sys.exit()

    def main(self):
        # Create a UDP socket
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        # Bind the socket to the port
        server_address = (self.host, self.port)
        self.logger.info("Started server on %s:%s" % (self.host,
self.port))
        self.sock.bind(server_address)

        while True:
            data, address_port = self.sock.recvfrom(4096)
            t =
threading.Thread(target=self.handle_request,args=(data,address_port))
            t.daemon = True
            t.start()
```

```
########################
# Argument parsing
########################
parser = argparse.ArgumentParser(description='FirewallDNS Server')
parser.add_argument('-H', '--host',help="Server host where create the
listener",default="localhost")
parser.add_argument('-p', '--port',help="Server port where create the
listener (highly recommended to use 53)",default=53)
parser.add_argument('-w', '--webserver',help="Sink webserver
hostname",required=True)
parser.add_argument('-D', '--webserver_dns',help="Webserver resolution DNS
server",default="8.8.8.8")
parser.add_argument('-d', '--dns_forward_servers',
                    help="Comma separated list of forwarding servers. For
example, 8.8.8.8,8.8.4.4",
                    default="8.8.8.8,8.8.4.4")
parser.add_argument('-t', '--forward_timeout',help="Forward query timeout
in seconds",default=0.5)
parser.add_argument('-e', '--elasticsearch',help="ElasticSearch API host
in format: <protocol>://<host>:<port>",
                        default="http://localhost:9200")
parser.add_argument('-k', '--api_key',help="API Key Secret Key base64
encoded",required=True)
parser.add_argument('-l', '--log_path',help="Log filename (with full path
if needed)",default="/var/log/firewallDNS.log")

args = parser.parse_args()

########################

logging.basicConfig(level=logging.INFO,format="%(asctime)s [%(levelname)s]
- %(message)s",
    handlers=[logging.FileHandler(args.log_path)])

host=args.host
port=int(args.port)
webserver=args.webserver
webserver_dns=args.webserver_dns
forward_servers=args.dns_forward_servers.split(",")
forward_timeout=args.forward_timeout
api_key_encoded = args.api_key
api_key_decoded = base64.b64decode(api_key_encoded).decode().split(":")
api_key_decoded = (api_key_decoded[0],api_key_decoded[1])


es_host = args.elasticsearch

FirewallDNSServer(host,port,webserver,webserver_dns,forward_servers,
```

```
forward_timeout,es_host=es_host,api_key=api_key_encoded).main()
```

5. *Código del servidor web de alertas*

```
from flask import Flask
from flask import Blueprint
from flask import render_template,url_for
from flask import abort
from flask import request
from flask import redirect
from dnslib import *
import random
import re
import socket
from elasticsearch import Elasticsearch
import warnings
import argparse
import base64
import logging
import logging.config

# Ignore ElasticSearch TLS warnings
warnings.filterwarnings("ignore")



#########################
# Argument parsing
#########################
parser = argparse.ArgumentParser(description='FirewallDNS Sink WebServer')
parser.add_argument('-H', '--host',help="WebServer host where create the
listener",default="localhost")
parser.add_argument('-p', '--port',help="WebServer port where create the
listener",default=80)
parser.add_argument('-e', '--elasticsearch',help="ElasticSearch API host in
format: <protocol>://<host>:<port>",
                            default="https://localhost:9200")
parser.add_argument('-k', '--api_key',help="API Key Secret base64
encoded",required=True)
parser.add_argument('-l', '--log_path',help="Log filename (with full path
if needed)",default="/var/log/sinkWebServer.log")

args = parser.parse_args()
#########################
logging.basicConfig(level=logging.INFO,format="%(asctime)s [%(levelname)s]
- %(message)s",
    handlers=[logging.FileHandler(args.log_path)])

#########################
# Server configuration
#########################
es_host = args.elasticsearch
```

```
api_key_encoded = args.api_key
api_key_decoded = base64.b64decode(api_key_encoded).decode().split(":")
api_key_decoded = (api_key_decoded[0],api_key_decoded[1])
es_connection =
Elasticsearch(es_host,verify_certs=False,api_key=api_key_decoded)

logging.getLogger('elasticsearch').setLevel(logging.CRITICAL)

app = Flask(__name__)
app.config['SERVER_NAME'] = args.host
#########################

def return_alert(domain,category,source_name):
    #### Escape URL Generation ####
    escape_subdomains =
["bypass","firewalldns","insecure","risk","danger","alert"]
    escape_id = random.randint(10000000,99999999)
    name = random.choice(escape_subdomains)
    escape_url = app.config['SERVER_NAME'] + "/bypass/" + name + "-" +
str(escape_id)
    es_connection.index(index="webserver-bypass",id=escape_id,
            document={"id":escape_id,"domain":domain,"name":name})
    ########
    logging.info("Successfully served %s from client %s" %
(domain,request.remote_addr))
    return
render_template('warning.html',web=domain,url=escape_url,category=category.
upper(),source_name=source_name)

def alert(domain):
    # First, query domains index to see if the host is a malicious domain
name
    query = {"match": {"domain":domain}}
    try:
        search = es_connection.search(index="malicious-domain",
query=query)
    except Exception as e:
        logging.error("Error in alert: %s" % e)
        abort(404)
    if len(search['hits']['hits']) > 0:
        category = search['hits']['hits'][0]['_source']['category']
        source_name = search['hits']['hits'][0]['_source']['source_name']
        return return_alert(domain,category,source_name)
    else:
        # If domain is not found, look for pointed IPs
        ws_req = DNSRecord.question(domain)
        ws_record = DNSRecord.parse(ws_req.send(dest="8.8.8.8",timeout=3))
        ips = [rr.rdata for rr in ws_record.rr]
        if len(ips) == 0:
```

```
            logging.warning("Bad request to %s from client %s" %
(domain,request.remote_addr))
            abort(404)
        for ip in ips:
            stringIP = str(ip)
            query = {"match": {"ip":stringIP}}
            try:
                search = es_connection.search(index="malicious-ip",
query=query)
            except Exception as e:
                logging.error("Error in alert: %s" % e)
            if len(search['hits']['hits']) > 0:
                category = search['hits']['hits'][0]['_source']['category']
                source_name =
search['hits']['hits'][0]['_source']['source_name']
                return return_alert(domain,category,source_name)
    logging.warning("Bad request to %s from client %s" %
(domain,request.remote_addr))
    abort(404)

def exception(domain):
    splitted = domain.split('-')
    if len(splitted) == 2:
        subdomain_name = splitted[0]
        subdomain_id = splitted[1]
        try:
            # Searching for bypass in ElasticSearch
            search = es_connection.search(index="webserver-bypass",
                        query={"bool":{"must":[
                            {"match":{"id":subdomain_id }},
                            {"match":{"name":subdomain_name}}]}})
            if search['hits']['total']['value'] > 0:
                # If found, adding an exception
                real_domain =
search['hits']['hits'][0]['_source']['domain']
                logging.info(real_domain)
                doc = {"domain":real_domain,"client":request.remote_addr}

es_connection.index(index="exceptions",id=subdomain_id,document=doc)
                # And returning a redirection to real domain
                response = redirect("https://"+real_domain,code=302)
                response.headers["Cache-Control"] = "no-store, no-cache"
                return response

        except ConnectionError as e:
            logging.error("Error connecting with ElasticSearch (exception):
%s" % e)
            abort(404)
```

```
    # If the end is reached, a bypass request error has ocurred
    logging.warning("Request made to bypass server with wrong format from
%s to %s"
            % (request.remote_addr,domain))
    abort(404)

@app.route('/about')
def home():
    return render_template('home.html')


@app.route('/bypass/<domain>')
def handle_bypass(domain):
    logging.info("Bypassing")
    bypass = re.compile(r"(bypass|firewalldns|insecure|risk|danger|alert)-
[\d]{8}")
    if bypass.match(domain) != None:
        return exception(domain)
    else:
        logging.warning("Received a bad request to bypass from %s to %s" %
(request.remote_addr,domain))
        abort(404)

@app.route('/query/<domain>')
def handle_query(domain):
    url = re.compile(r"(www\.)?[\w\d\-]+\.[\w\d]+")
    if url.match(domain) != None:
        return alert(domain)
    else:
        logging.warning("Received a bad request to query from %s to %s" %
(request.remote_addr,domain))
        abort(404)


if __name__ == '__main__':
    app.run()
```