Universidad Zaragoza
1542

Trabajo Fin de Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Implementación de un DNS con capacidad de detección de dominios maliciosos generados algorítmicamente

Implementation of a DNS capable of detecting algorithmically generated malicious domains

Autor

Víctor Mateo Calvillo

Director

Director: Ricardo J. Rodríguez Fernández

Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza
Diciembre de 2022
Curso 2021/2022

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe remitirse a seceina@unizar.es dentro del plazo de depósito)

D./Dª. Víctor Mateo Calvillo ,

en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de Estudios de la titulación de Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación (Título del Trabajo)

Implementación de un DNS con capacidad de detección de dominios maliciosos generados algorítmicamente

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 25 DE NOVIEMBRE DE 2022

Fdo: Víctor Mateo Calvillo

RESUMEN

En los últimos años, ha habido un aumento drástico en la cantidad de ciberataques, lo que representa una seria amenaza para organizaciones de todo tipo. Las técnicas de detección y prevención de malware han avanzado para tratar de contener este crecimiento, pero los ciberdelincuentes también mejoran sus técnicas.

Este proyecto se centra en una técnica desarrollada por los atacantes llamada *Algoritmo de Generación de Dominios* (*Domain Generation Algorithm, DGA*). Esta técnica genera una gran cantidad de nombres de dominio que utilizan las muestras de malware para lograr ponerse en contacto con los servidores de los atacantes. Los métodos tradicionales para detectar DGAs, como las listas negras, no son efectivos y se ha tenido que desarrollar otros métodos de detección.

Este trabajo explica cómo el aprendizaje automático es un método que aporta buenos resultados en la detección de DGAs. Además, se centra en el desarrollo de un sistema software que implementa una capa de detección basada en aprendizaje automático mediante dos modelos de aprendizaje diferentes (aprendizaje no profundo y aprendizaje profundo). Se ha logrado montar un sistema funcional que permite detectar y bloquear los dominios generados por los DGAs. Además, debido a que el sistema está muy desacoplado, permite que la incorporación de otras técnicas de detección o nuevos modelos de aprendizaje automático sea fácilmente implementable.

ABSTRACT

In recent years, there has been a drastic increase in the number of cyber attacks, which represents a serious threat to organizations of all types. Malware detection and prevention techniques have advanced in an attempt to contain this growth, but cybercriminals are also improving their techniques.

This project focuses on a technique developed by attackers called *Domain Generation Algorithm* (*DGA*). This technique generates a large number of domain names that use malware samples to contact attackers' servers. Traditional methods for detecting DGAs, such as blacklisting, are not effective and other detection methods have had to be developed.

This work explains how machine learning is a method that provides good results in the detection of DGAs. In addition, it focuses on the development of a software system that implements a detection layer based on machine learning through two different learning models (no-deep learning and deep learning). A functional system has been set up that allows the detection and blocking of domains generated by DGAs. In addition, because the system is very decoupled, it allows the incorporation of other detection techniques or new machine learning models to be easily implemented.

Índice

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

Cada año el número de ciberataques aumenta de forma masiva, al igual que la adopción de Internet en la vida cotidiana. Así también, conlleva un gran coste a nivel mundial, hasta el punto en el que se espera que en 2025 alcance un coste anual de 10.5 billones de dólares estadounidenses [11].

Los objetivos finales de los atacantes pueden ser diversos, desde robo de credenciales hasta sustraer documentación confidencial. No obstante, se pueden definir las fases que debe cumplir cualquier ataque para lograr lo que se propone. Para ello, Lockheed Martin [12] desarrolló el framework Cyber Kill Chain® [13] en el cual se definen los siete pasos que un ataque debe completar para cumplir su objetivo (Figura 1.1).

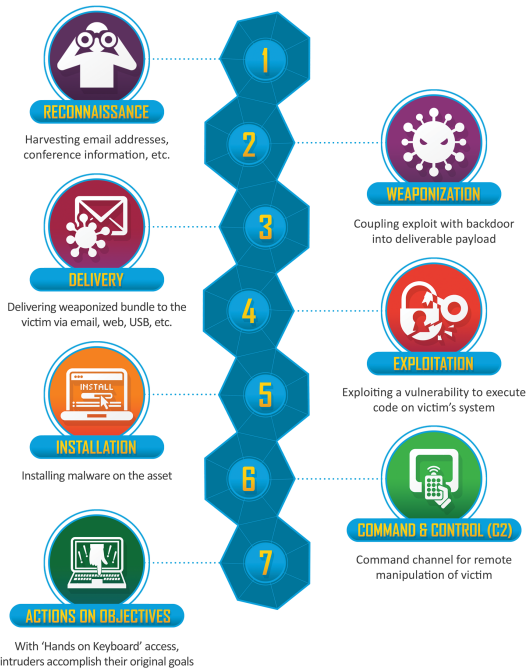

Figura 1.1: Los siete pasos de Cyber Kill Chain®
Fuente [14]

Tras las cinco primeras fases del ataque, se cuenta con el control del sistema de la

víctima, la instalación de software malicioso (*malware*) en este y se procede a ejecutar las acciones maliciosas dirigidas desde un servidor conocido como servidor de mando y control o, en inglés, *Command and Control server* (*C2*). Para recibir las instrucciones del C2 se debe establecer un canal de comunicación entre el malware y este servidor.

Existen diversas técnicas para lograr dicha comunicación y se pueden clasificar haciendo uso del framework *MITRE ATT&CK* [18]. En su matriz de ataques se pueden observar 14 tácticas diferentes y más de 200 técnicas utilizadas por los atacantes. Entre las tácticas se encuentra *Command and Control* [15]; en la cual se definen 16 técnicas para establecer la comunicación con el C2. Este trabajo se centrará en una de ellas, conocida como *Resolución Dinámica* (*Dynamic Resolution*) [16] y en concreto, en la sub-técnica llamada *Algoritmos de Generación de Dominios* (*Domain Generation Algorithms*) [17].

Para evadir las técnicas de detección y prevención comunes, los atacantes se aprovechan de la Resolución Dinámica [16], cuyo objetivo es ajustar dinámicamente parámetros como el nombre de dominio, la dirección IP o el número de puerto que se usan para establecer la conexión con el C2. Los atacantes pueden usar esta técnica como canal de respaldo en caso de perder el contacto con el servidor principal y así recuperar la conexión. Esto es posible mediante el uso de un algoritmo compartido por el malware y el atacante.

En el caso de los *algoritmos de generación de dominios* (*DGAs*) [17], el parámetro que se ajusta dinámicamente es el nombre de dominio. A través del algoritmo compartido, se genera de manera pseudoaleatoria el dominio C2 al que establecer la conexión, en vez de depender de una lista estática o de direcciones IP conocidas.

Habitualmente la detección y prevención de las conexiones con el C2 se han realizado a través de los llamados *sumideros DNS* [6]. Estos sumideros son servidores DNS intermedios que actúan como *proxy* de las peticiones DNS generadas dentro de una red. Los sumideros poseen una lista negra que incluye nombres de dominio que se han visto involucrados en ataques. Cuando se recibe una petición DNS, se comprueba si el dominio existe en su lista negra. Si no se encuentra, el servidor realiza la resolución DNS de la manera habitual. En caso contrario, el servidor responde con una resolución falsa (normalmente la dirección IP virtual del interfaz *loopback*) o con una resolución que incluye la dirección IP de un servidor falso para suplantar al C2 e investigar el comportamiento del malware. De esta forma, si el malware depende de una lista estática de dominios y todos ellos están incluidos en la lista negra, el malware pasa a estar inactivo y su reactivación supone un proceso demasiado costoso para el atacante. En el ámbito de esta detección es donde los DGAs toman ventaja, debido a su independencia con respecto a las listas estáticas.

Una de sus ventajas es que incluso si se ha bloqueado un nombre de dominio generado y se ha desmantelado la infraestructura del ataque, el atacante tiene la capacidad de registrar otro nombre de dominio que será generado por el algoritmo y así conseguir desplegar fácilmente una nueva infraestructura. Por otro lado, debido a que cada algoritmo puede generar una gran cantidad de dominios, imposibilita el uso de listas negras. Si se tratase de obtener una lista con todos los posibles dominios generados por uno de estos algoritmos, el tamaño sería tan grande que impediría su uso en la práctica.

Con el tiempo, los atacantes han desarrollado e implementado diferentes algoritmos para crear dinámicamente los dominios. Aún así, habitualmente estos algoritmos tienen un comportamiento similar al de los algoritmos generadores de números pseudoaleatorios (*PRNG*, de sus siglas en inglés) [20]. De esta forma, los valores tienen una apariencia aleatoria pero son generados de forma determinista a partir de ciertos valores iniciales. Con esto, el atacante puede predecir qué nombre de dominio debe registrar para que los dispositivos infectados se conecten a él, ya que es capaz de inicializar su generador de nombres del mismo modo que las muestras de malware. En muchas ocasiones, además de un parámetro llamado *semilla* (*seed*), para la inicialización se usa otro tipo de información que el atacante puede predecir como la fecha de ejecución. Con esto se logra no repetir la misma secuencia de dominios cada vez que se ejecuta el malware y, por tanto, dificultar la detección.

Debido al potencial de esta técnica y la gran variedad de posibles familias de algoritmos, resulta complicado bloquear, rastrear o lograr tomar el control del canal de comunicación que el malware trata de establecer con el C2. No obstante, se han realizado diversos estudios para aplicar métodos de aprendizaje automático para resolver este problema [4, 3, 1, 21]. Algunas investigaciones [4, 3] extraen datos del DNS como el TTL (*Time-to-Live*) para determinar si el dominio puede haber sido generado por un DGA. Otras se centran en el número de peticiones DNS fallidas [1], ya que los DGAs pueden generar y probar una gran cantidad de dominios hasta encontrar alguno que esté activo.

Para este trabajo se parte de la tesis doctoral de José F. Selvi Sabater [21], centrada en técnicas de detección basadas únicamente en características léxicas del nombre del dominio. Como principal ventaja de esta aproximación, el autor sostiene que "*los nombres de dominio, al contrario de lo que sucede en otros campos de los mensajes DNS, no cambian ya que son generados desde el lado cliente y no del servidor*" [21, p. 50] y, por tanto, las características empleadas no dependen de información volátil que puede cambiar a lo largo del tiempo.

1.1. Objetivos

El objetivo de este trabajo es implementar el sistema propuesto en [21, p. 90], que contempla el uso de un modelo de aprendizaje no profundo y otro de aprendizaje profundo para la detección y clasificación de nombres de dominio maliciosos generados por DGAs. Este sistema se integrará en un servidor DNS para añadirle una nueva capa de seguridad. La implementación del sistema se ha desacoplado totalmente del software del servidor DNS, lo que permite que pueda ser usado junto con cualquier servidor DNS. Esto aporta un mayor grado de flexibilidad e independencia a la hora de incorporar el clasificador en entornos reales ya que no está ligado a un DNS específico ni a una versión concreta.

Todo el código desarrollado se ha liberado bajo licencia GNU/GPLv3 para que pueda ser usado y mejorado por la comunidad [7].

1.2. Estructura del documento

Este documento se encuentra dividido en 6 capítulos. En el capítulo 2 se explica de forma básica el protocolo DNS y los tipos y funcionamiento de los DGAs. A lo largo del capítulo 3 se explica el estado del arte presente en la literatura sobre detección de DGAs. El capítulo 4 detalla el sistema desarrollado. En el capítulo 5 se explican las pruebas que se han realizado y los resultados obtenidos. Por último, en el capítulo 6 se realiza una conclusión del trabajo realizado y el trabajo a futuro.

Capítulo 2

Conceptos previos

En este capítulo se introducen los conceptos y definiciones que son necesarios para comprender el trabajo desarrollado. En primer lugar, se habla sobre el protocolo DNS y los sus procesos de resolución. Seguidamente, se explican los algoritmos de generación de dominios y sus clasificaciones.

2.1. Protocolo DNS

Los humanos recuerdan y reconocen mejor nombres que direcciones IP (e.g., unizar.es frente a 155.210.11.37). Dos dominios muy parecidos pueden tener direcciones IP muy diferentes y podría resultar demasiado costoso para los usuarios recordarlas. Por otro lado, una dirección IP está ligada a una máquina concreta, y por tanto, si se quiere cambiar el dominio de máquina, también cambiaría su dirección pero no se querría cambiar su nombre. Por todo ello surge el protocolo DNS (*Domain Name System*) [8], que establece un sistema de nombrado de máquinas y una correspondencia entre dichos nombres y las direcciones IP de dichas máquinas.

El protocolo DNS describe los mecanismos necesarios para traducir los nombres de dominio en direcciones IP. En el escenario habitual se encuentra *el cliente DNS*, quien solicita la resolución de un dominio, y los servidores DNS, encargados de obtener la dirección (o direcciones) IP asociada a dicho dominio. Existen dos tipos de servidores DNS: servidores autorizados, que tienen la información necesaria para devolver la asociación nombre-IP que se está buscando; y servidores recursivos, que no poseen dicha asociación pero gestionan la petición DNS para obtenerla.

Una resolución DNS comienza con la petición del cliente a su servidor configurado. Si este servidor tiene autoridad sobre el dominio por el cual se está preguntando, le devuelve al cliente la asociación (véase la Figura 2.1). En caso contrario, si el servidor tiene caché, consulta la petición en su caché. Si tiene almacenada la respuesta y no ha expirado, esta será enviada al cliente (véase la Figura 2.2). Si la respuesta DNS no

se encuentra en el servidor, comienza el proceso de resolución entre servidores. En la Figura 2.3 se ha representado el proceso de consultas recursivas. En este proceso, el primer servidor que ha recibido la consulta del cliente reenvía la petición recursivamente a otros servidores DNS hasta obtener la asociación nombre-IP y devolvérsela al cliente.

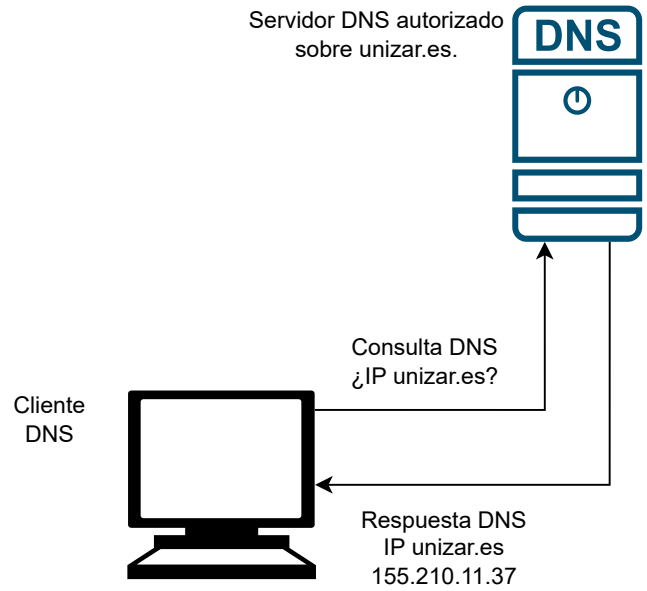

Figura 2.1: Resolución DNS con servidor DNS autorizado.

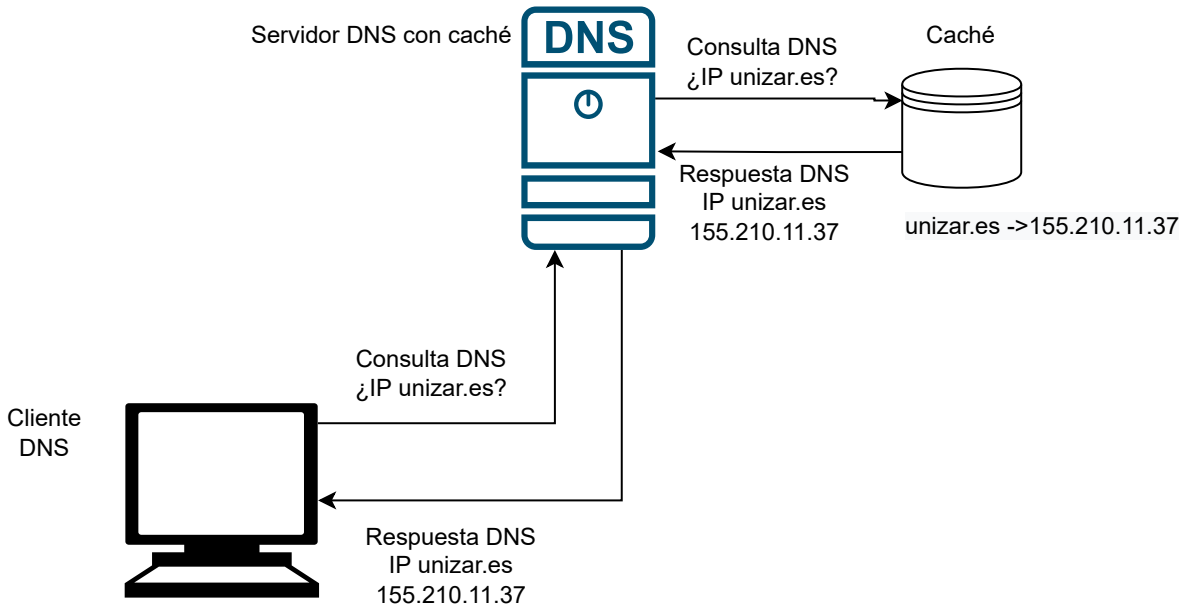

Figura 2.2: Resolución DNS con servidor DNS caché.

Otro proceso de resolución es el de consultas iterativas. En este proceso, el servidor que recibe la consulta contesta con lo máximo que sepa de esa búsqueda sin preguntar a ningún otro servidor. Si el servidor no tiene la respuesta de la consulta, suele devolver

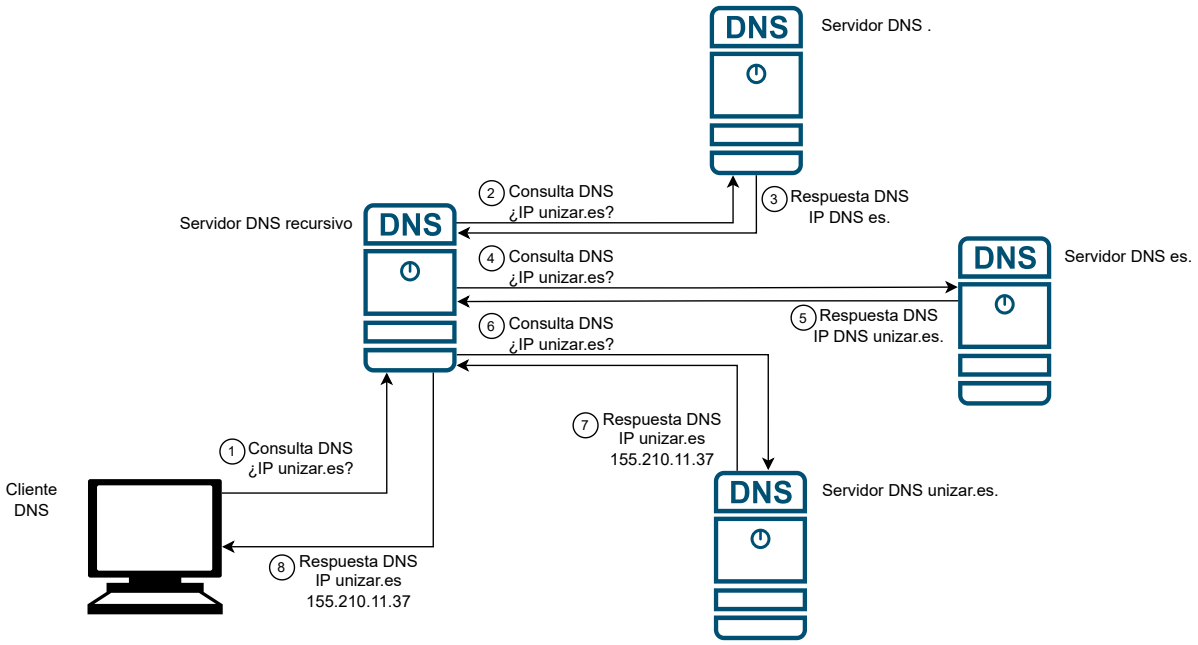

Figura 2.3: Resolución DNS con servidor DNS recursivo.

la IP del servidor del siguiente subdominio en la cadena de búsqueda. Este proceso de consulta es el que realiza aisladamente el servidor recursivo en la Figura 2.1 cuando está tratando de obtener la respuesta de la consulta de su cliente.

Como se puede observar, el cliente únicamente es consciente del envío de la consulta a su servidor DNS configurado, sin importar si este es el autorizado, si tiene la respuesta en caché o si ha realizado un consulta recursiva. Por último, cabe destacar que el protocolo DNS funciona por defecto sobre el protocolo UDP [24] en el puerto 53.

2.2. Algoritmos de generación de dominios

Tal y como se menciona anteriormente, la sub-técnica DGA es utilizada para ajustar dinámicamente el nombre de dominio a través de un algoritmo compartido por el atacante y el software. Plohmann et al. [19] dividen la clasificación de los DGAs según dos categorías: la fuente de la semilla y el esquema de generación. Combinando estas dos categorías, se generan 16 posibles familias de algoritmos, aunque solo 6 de ellas se han encontrado siendo implementadas por DGAs reales.

De esas dos categorías, el esquema de generación es el único que influye de forma directa en el aspecto de un nombre de dominio, y por tanto es la única categoría de interés para este trabajo. Dentro de esta categoría se distinguen cuatro tipos de esquemas [19]:

– Aritméticos: Calcula una secuencia de valores pseudoaleatorios que son convertidos a caracteres ASCII para formar el nombre de dominio. Este esquema

es el más común, encontrándose en 37 de los 43 DGAs analizados en [19]. Se muestran ejemplos de DGAs obtenidos mediante este esquema en el Listado 2.1.

- Basados en hash: Emplean una función criptográfica de resumen (hash) en su formato hexadecimal.
- Basados en diccionario: Concatenan una secuencia de palabras provenientes de uno o más diccionarios embebidos en la propia muestra u obtenidos de Internet. Es el segundo esquema más empleado (3 de los 43 DGAs analizados en [19]). Se muestran ejemplos de DGAs obtenidos mediante este esquema en el Listado 2.2.
- Basados en permutaciones: Derivan todos los dominios posibles a partir de permutar los caracteres de un dominio inicial.

```
4chij1xk8axsrite.ddns.net
alg81hc07rq4a6kdmnw.ddns.net
klkran5no83vwpotu8qxqro.ddns.net
u8kvadodg2evul5xsn3vgrk.ddns.net
54kjo6sxe4qtwn7.ddns.net
```

Listado 2.1: Ejemplo de nombres generados mediante generación aritmética.

```
unuinstrumentation.com
missiondeveloped.com
contentfortheweiler.com
numberforscience.com
numberworkshop.com
surfacethetheandmissions.com
februaryrecent.com
spacesciencehopeseeking.com
provideatmoscomp.com
provenmeritspacecraftco.com
launchlauheldnathe.com
distkuideproposals.com
```

Listado 2.2: Ejemplo de nombres generados por DGAs basados en diccionario.

En los DGAs, al igual que en los PRNGs, una de las propiedades que muestra su calidad es la cantidad de nombres pseudoaleatorios que puede generar sin producir repeticiones. Si el algoritmo de generación de dominios no está bien diseñado, puede producir pocos nombres únicos y así hacer que la estrategia para la que los DGAs fueron pensados falle. Por ejemplo, Bader [2] publicó una reimplementación en Python del DGA empleado por el malware Corebot para estudiar su repetibilidad. Este DGA se basa en generación aritmética y su aspecto es muy similar al esperado en un algoritmo PRNG. Las pruebas mostraron la capacidad de generar al menos 100 millones de dominios sin producir ninguna repetición.

Nótese que los nombres generados mediante DGAs basados en diccionario son visualmente muy diferentes a los generados por el método aritmético (véase la diferencia entre los dominios mostrados en el Listado 2.1 y Listado 2.2). Muchos de ellos podrían haber sido registrados por un ser humano con fines legítimos, y por ello resultan mucho más difíciles de detectar. Cabe destacar que su capacidad para generar una gran cantidad de valores sin producir ningún bucle es mucho menor. Por ejemplo, el malware Gozi [2] utiliza un DGA basado en diccionario y pruebas realizadas mostraron que se produjeron un 14,26

2.3. Estándar UML

UML (*Unified Modeling Language*) [9] es un lenguaje de modelado estandarizado desarrollado para ayudar a los desarrolladores de sistemas y programas informáticos a especificar, visualizar, construir y documentar los elementos de los sistemas de software, así como para el modelado comercial y otros sistemas no informáticos. Incluye aspectos como la funcionalidad del sistema y expresiones concretas del lenguaje. En UML se utilizan y asocian elementos de diferentes formas para desarrollar diagramas que representan ciertos aspectos estructurales y estáticos de un sistema en particular, como la colaboración de diferentes objetos dentro del sistema.

Se definen a continuación los diferentes tipos de diagramas usados en las fases de análisis y diseño del sistema a desarrollar.

Diagrama de Clases

El diagrama de clases se utiliza para modelar el comportamiento estático de un sistema. Además de incluir los atributos y las funciones de las clases a implementar, se indica gráficamente cómo se relacionan las diferentes clases entre sí. Las relaciones entre clases poseen una multiplicidad, que se indica gráficamente con un número en la línea de conexión al costado de cada clase. Cuando la relación es 1, significa que la clase solo está relacionado con una sola instancia, mientras que si es N, * o 1..*, significa que está relacionado un número N de veces que puede ser cualquier número de 0 a N, o en el caso 1..*, puede ser de 1 a N (es decir, debe estar relacionado al menos una vez).

Diagrama de Secuencia

Un diagrama de secuencia es un tipo de diagrama que muestra cómo los objetos interactúan entre sí en una secuencia determinada. Se utiliza para modelar el comportamiento dinámico de un sistema. Se pueden utilizar para comprender el comportamiento de un sistema, diseñar nuevos sistemas o documentar sistemas

existentes.

Las interacciones se representan como mensajes entre los objetos, que están representados en una línea de vida vertical. Los mensajes se organizan en una secuencia, de arriba a abajo, según el momento en que ocurren. Los mensajes entre los objetos están representados por flechas.

Capítulo 3

Estado del arte

El ámbito de la detección de dominios generados por DGAs ha recibido una gran atención en los últimos años. La gran mayoría de investigaciones y soluciones que se han realizado están basadas en técnicas de aprendizaje automático, el cual se puede dividir en dos grandes grupos en función de su naturaleza: técnicas de aprendizaje no profundo y técnicas de aprendizaje profundo. A continuación, se revisan algunas de estas investigaciones que están más relacionadas con el trabajo desarrollado en este TFG.

3.1. Técnicas de aprendizaje no profundo

Con respecto al estado del arte presente en la literatura sobre las técnicas de aprendizaje no profundo, en 2011 Bilge et al. publicaron EXPOSURE [4], mejorado más tarde en 2014 [3]. Este sistema de detección de dominios maliciosos emplea técnicas de análisis pasivo de DNS. Las técnicas se basan en 15 características organizadas en 4 grupos: relativas al tiempo, extraídas de las respuestas DNS, basadas en el campo TTL de los registros de recurso DNS y obtenidas del nombre de dominio en sí mismo. Muchas de estas características son altamente volátiles y pueden cambiar a lo largo del tiempo. Los autores se centraron principalmente en dos valores de las características léxicas: el porcentaje de caracteres numéricos en el nombre y el porcentaje de la subcadena de texto con significado de mayor tamaño con respecto a la longitud total del nombre. Sin embargo, el sistema no proporciona un análisis léxico completo del dominio. Empleando un árbol de decisión J48 [25], su sistema obtuvo un 98.4 % de acierto en la detección con alrededor de un 1

En 2021 Antonakakis et al. presentaron Pleiades [1] Su aproximación se basa en que una muestra de malware que utiliza un DGA para establecer conexión con el C2 genera una cantidad significativa de peticiones fallidas relacionadas con la inexistencia del dominio (respuestas *NXDomain* [5]). Esto es debido a que los DGAs pueden

generar una gran cantidad de nombres pero solo unos pocos de ellos están registrados. Además, también sostienen que una muestra diferente del mismo malware generaría un tráfico de respuestas NXDomain similar. El sistema utiliza algoritmos de agrupación y clasificación, empleando además de algunas características volátiles, estadísticas extraídas del nombre de dominio (como puede ser la longitud del nombre, nivel de aleatoriedad y frecuencia en la distribución de carácteres). Finalmente, usan un modelo oculto de Markov (*Hidden Markov Model* o HMM, del inglés) para agrupar los dominios sospechosos.

3.2. Técnicas de aprendizaje profundo

Otros autores han realizado propuestas basadas en técnicas de aprendizaje profundo. Estas técnicas no precisan de un proceso de ingeniería de características tan complejo como en los modelos de aprendizaje no profundo.

Woodbridge et al. propuso en 2016 una red neuronal que incluía una capa de codificación, una capa oculta LSTM con 128 unidades y una capa densa que combinaba las salidas de la capa anterior en un único resultado de clasificación [26]. Compararon los resultados de su modelo con los de otros tres modelos: un modelo HMM, una regresión logística a partir de los 2-Grams y un modelo Random Forest utilizando características extraídas manualmente similares a las propuestas por otros autores. Los resultados obtenidos mostraron que el modelo propuesto clasificaba correctamente el 98 % de los nombres generados con DGA, con una tasa de falsos positivos (*False Positive Rate o FPR*) de 0.001.

Unos años más tarde, Tran et al. propusieron en [22] una modificación con respecto al trabajo de Woodbridge que utilizaba una capa *LSTM with Multiclass Imbalance* en lugar de una capa LSTM. Esta red incluye la posibilidad de introducir un coste más granular en el proceso de aprendizaje, haciendo que el impacto de los errores de clasificación varíe en función del tipo de error. Esta aproximación se plantea en dos etapas. En la primera de ellas, se entrena una red para clasificar los dominios en "*generados por DGA*" o "*no generados por DGA*", con un coste mayor para el error de clasificar un dominio un nombre de dominio legítimo como generado por DGA que el error opuesto. Esto evita que los dominios legítimos se bloqueen por error. La segunda etapa solo se ejecuta si la primera ha clasificado el dominio como "*generado por DGA*". En ella, otra red clasifica el nombre de dominio para obtener el tipo de DGA específico que lo ha generado.

Estos trabajos han servido de punto de partida para la tesis doctoral de José F. Selvi Sabater [21] para proponer unos modelos que aprovechen las ventajas de ambos

y mitiguen sus desventajas como por ejemplo, basar la clasificación y detección de los dominios maliciosos en características muy volátiles provenientes del tráfico en la red en un momento puntual.

Capítulo 4

Sistema desarrollado

En este capítulo se procede a explicar el sistema desarrollado en el trabajo junto con todas las propuestas y decisiones de diseño, su arquitectura y su funcionamiento.

4.1. Análisis y propuestas de diseño

Inicialmente se parte del sistema propuesto en la Figura 4.1. Este sistema es un clasificador combinado que aprovecha las ventajas de los modelos Random Forest (aprendizaje no profundo) y LSTM (aprendizaje profundo) y mitiga sus desventajas.

Se comienza con una primera capa llamada *frontend* que recibe el dominio a clasificar. Este *frontend* busca en su memoria caché si el dominio ya ha sido clasificado con anterioridad, en cuyo caso responde con el resultado. Esto aumenta la eficiencia debido a que evita tener que volver a realizar la clasificación. Si la información no se encuentra en caché, el *frontend* solicita a Random Forest la clasificación. Este modelo proporciona una rápida respuesta y una fácil explicabilidad en el caso de bloquear un dominio legítimo. Con esto se puede justificar el motivo de bloqueo y permite ajustar el funcionamiento del modelo para posteriores peticiones. El resultado aportado se almacena en la memoria caché para futuras peticiones. Tras haber respondido con la clasificación, se ejecuta fuera de línea la misma petición al modelo LSTM. Este modelo tiene un tiempo de respuesta superior, pero no impacta en el tiempo de respuesta debido a que se ejecuta fuera de línea. En el caso de que LSTM clasifique un dominio como no malicioso, se actualiza el valor en la memoria caché para que en sucesivas peticiones no se clasifique el mismo dominio como malicioso. Esta combinación de modelos se aprovecha de la mejor capacidad de LSTM para no producir falsos positivos.

En esta propuesta teórica se tuvo en cuenta principalmente un aspecto para plantear su implementación real: el tiempo de clasificación. Si llegase a ser muy elevado, repercutiría en el tiempo de respuesta ante una petición DNS, haciendo que se descartara. En [21] se puede ver como el modelo Random Forest es capaz de clasificar

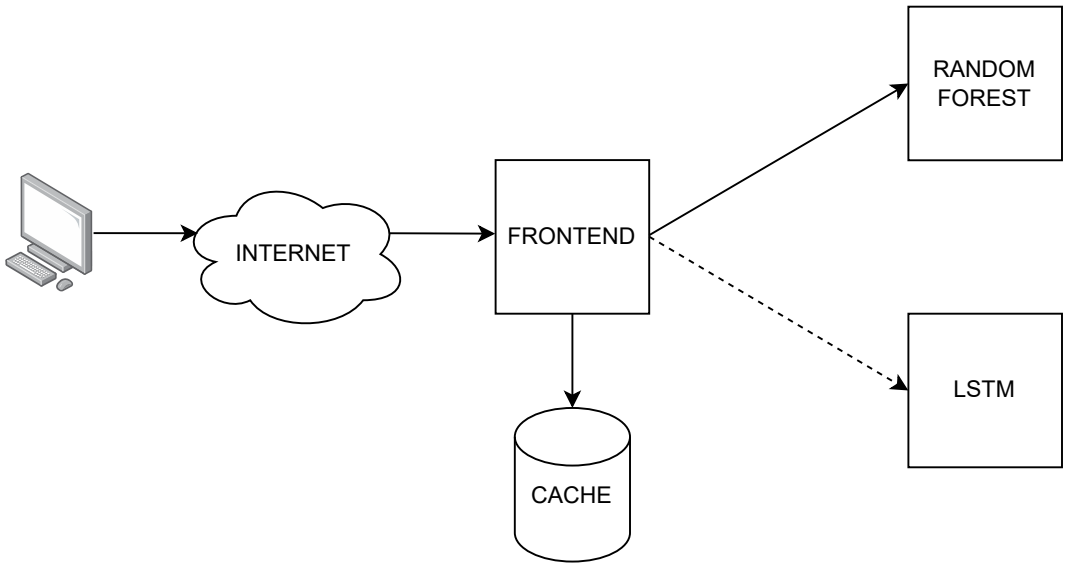

Figura 4.1: Clasificador Combinado.
Fuente [21, p. 90]

64.000 nombres de dominio en 2.569 segundos; por tanto, el tiempo de clasificación de un solo dominio no será significativo en el tiempo total de resolución DNS. Por otro lado, se observa en [21] como la velocidad de clasificación del modelo LSTM con el mismo conjunto de datos es de 59 segundos, un tiempo muy superior al modelo anterior pero que sigue sin tener un gran impacto en el tiempo total de resolución DNS.

Teniendo esto en cuenta se plantean esencialmente dos diseños; un módulo específico y un microservicio previo (Figura 4.2 y Figura 4.3, respectivamente). El módulo específico (Figura 4.2) es un diseño muy acoplado, ya que el clasificador combinado se integra con un software DNS. La principal ventaja de esta opción es que la velocidad de respuesta es mayor debido a la integración del clasificador en el propio servidor DNS. Además, a la hora de desplegar el servidor DNS, se instala todo en conjunto sin necesidad de incorporar el clasificador específicamente. Para el funcionamiento, el usuario simplemente ha de tener configurado el sistema como su servidor principal y al generar una petición DNS se envía directamente a este. El servidor clasifica la

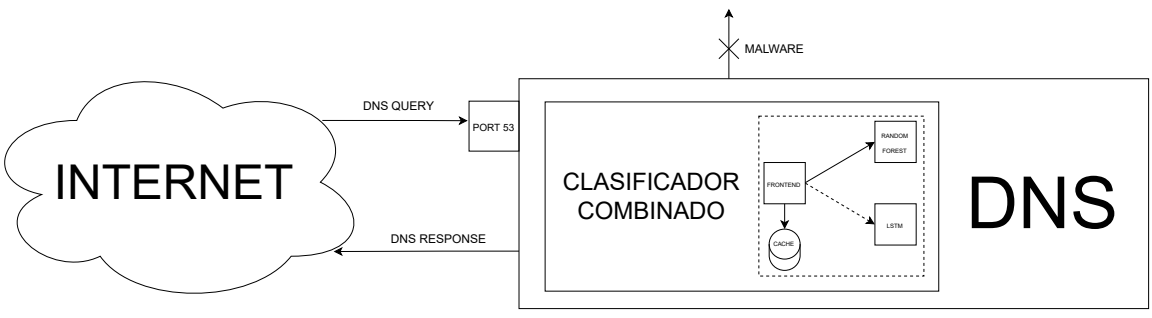

Figura 4.2: Módulo DNS específico.

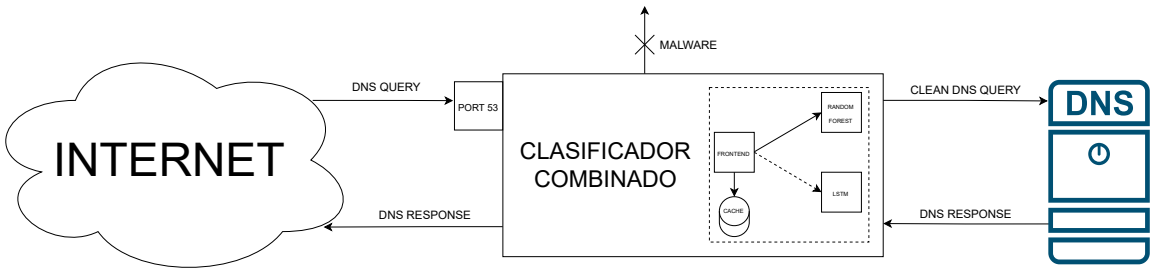

Figura 4.3: Microservicio previo a DNS.

petición y si no se detecta como malware, se procede a una resolución habitual. En caso contrario, el servidor responde con una resolución falsa (dirección IP virtual del interfaz loopback o con dirección IP de un servidor falso para suplantar al C2).

Por otra parte, la arquitectura de microservicio previo al DNS (Figura 4.3), tiene un tiempo de respuesta mayor ya que no está optimizado para un software DNS concreto. En cambio, tiene un diseño modular que lo hace independiente del servidor DNS y ofrece mayor versatilidad. Para el funcionamiento, el usuario tiene configurado el microservicio previo como su servidor DNS principal. Cuando el microservicio recibe una petición, se comprueba si se detecta el dominio como malicioso. Si es así, el microservicio responde de la misma forma que en el caso anterior (es decir, con una resolución falsa). Por el contrario, si no se detecta como malware, la petición DNS es reenviada a un servidor DNS para su resolución habitual. Al recibir la respuesta del servidor, el microservicio la reenvía al usuario. En este escenario, el microservicio actúa como un proxy para las peticiones que no son clasificadas como malware.

Teniendo en cuenta la velocidad global del clasificador que se ha comentado anteriormente, la ventaja en el tiempo de respuesta del módulo específico no es una razón suficiente para su elección. En cambio, el microservicio previo ofrece una flexibilidad e independencia sobre el software DNS utilizado que es muy significativa a la hora de una implementación real. Este diseño no solo permite incorporar una capa de seguridad adicional a cualquier servidor DNS local en una red interna, sino que puede implementarse sin necesidad de un servidor local y reenviar las peticiones a cualquier otro servidor público. Con todo ello, el diseño elegido para este proyecto ha sido el de un microservicio previo.

Además, se ha realizado un pequeño cambio con respecto al clasificador combinado. En este sistema la petición al modelo LSTM no se hace fuera de línea sino simultáneamente junto con la petición al modelo Random Forest. La primera clasificación que recibe el *frontEnd* es la que se usa para enviar al usuario. La principal ventaja de este cambio es que, en caso de que Random Forest tardase demasiado o dejase de funcionar por alguna razón, se puede usar la clasificación de LSTM para

devolver la resolución al usuario.

4.2. Arquitectura del sistema propuesto

Se han realizado diagramas UML [9] para comprender mejor la arquitectura del sistema. Para proporcionar una visión global de todo el sistema y sus componentes se diseña primero un diagrama de clases. Además, también se han realizado diagramas de secuencia para ilustrar las interacciones entre todos los componentes involucrados en el sistema.

En la Figura 4.4 se puede observar el diagrama de clases del sistema. Como clase principal se encuentra el *FrontEnd* que actúa de orquestador. Esta clase recibe las peticiones DNS y se encarga de su resolución. Otros elementos que ya se han visto en los esquemas anteriores son la caché y los clasificadores. La caché no solo busca y almacena las clasificaciones de forma simple, sino que tiene inteligencia para gestionar elementos duplicados, almacenar los dominios maliciosos para notificar al administrador y borrar clasificaciones antiguas. Por otro lado, cada clasificador tiene un algoritmo. La clase *Algorithm* representa los modelos de aprendizaje automático que clasifican los nombres de dominio. Para el diseño de esta clase se ha usado el Patrón Estrategia (*Strategy pattern*) [10]. De esta forma se ha desacoplado el modelo de aprendizaje usado para la clasificación, haciendo posible que en un futuro puedan añadirse nuevos modelos. La clase *IPAddress* se ha creado para representar algunos parámetros de configuración necesarios para el *FrontEnd*. Por último, la clase *SharedMemory* representa la memoria compartida entre los clasificadores y el *FrontEnd* para cada petición DNS que recibe el *FrontEnd*.

4.3. Funcionamiento

El funcionamiento del sistema de forma detallada se explica a continuación haciendo uso de los diagramas de secuencia. En la Figura 4.5 se observa la inicialización de todos los objetos y cómo comienza el sistema a funcionar. El objeto *FrontEnd* es el encargado de inicializar la caché y los clasificadores. También crea un socket en el que escuchará las peticiones DNS de los usuarios. Una vez que todos los objetos están inicializados, el *FrontEnd* queda a la espera de recibir una petición de resolución DNS por parte de un usuario. Cuando llega, lanza un hilo de ejecución con la función *handleQuery* encargada de resolver la petición a partir de los bytes del datagrama UDP recibido y el dirección IP del usuario. Tras lanzar este hilo de ejecución, el *FrontEnd* vuelve a quedar a la espera de recibir nuevos datagramas UDP.

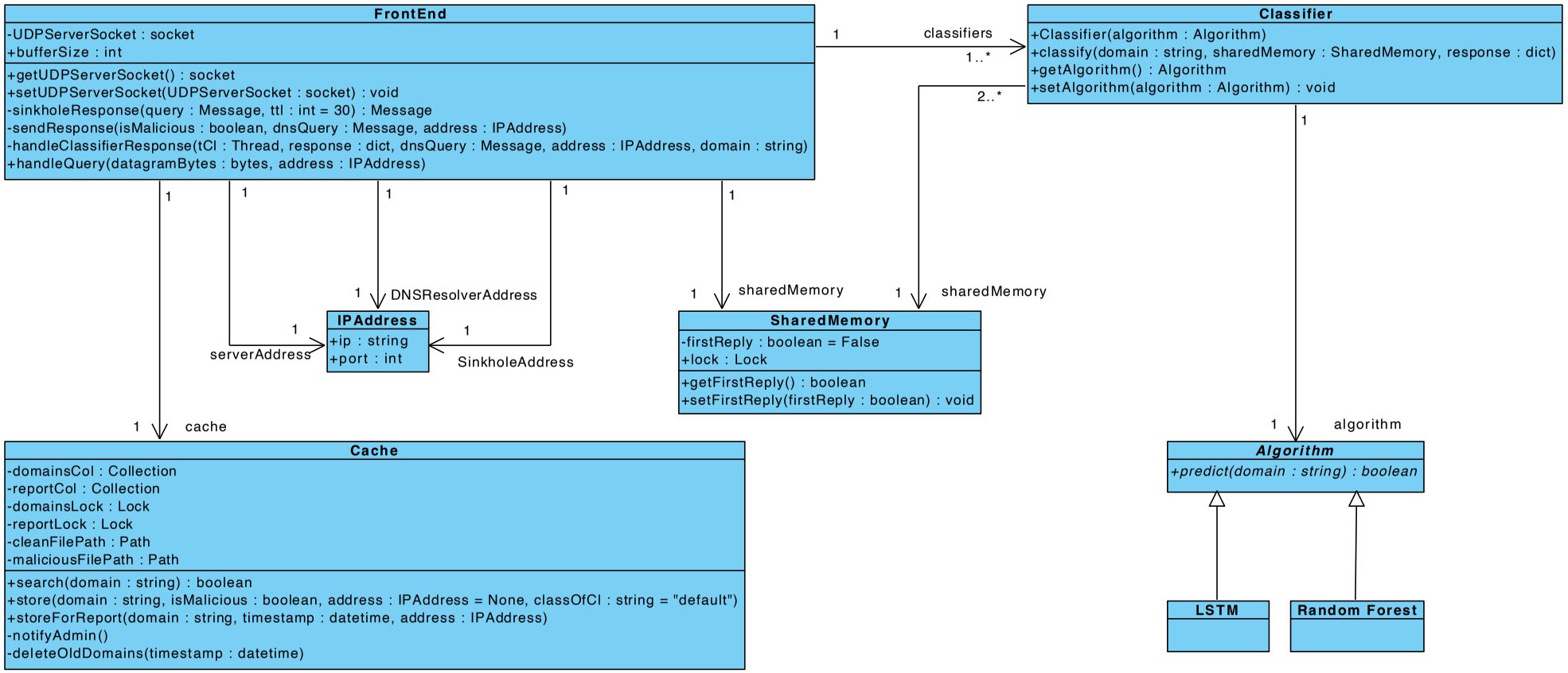

Figura 4.4: Diagrama de clases del sistema desarrollado.

La función *handleQuery* se detalla en la Figura 4.6. En primer lugar, se extrae la petición DNS de los bytes del datagrama UDP para obtener el dominio por el cual se está preguntando. Con esto se busca la clasificación en la caché y si la clasificación está almacenada, se ejecuta la función *sendResponse*. En caso contrario, se crea un objeto *SharedMemory* que pasará junto con otros elementos como parámetro de la función *classify* de cada clasificador. Esta función será ejecutada en un hilo de ejecución, creando así tantos hilos como clasificadores tenga el sistema. Una vez solicitada la clasificación a los modelos, el *FrontEnd* queda a la espera de la respuesta de esta función. En esta respuesta, además de la clasificación, se incluye información con respecto al algoritmo usado y acerca de si ha sido el primero de los clasificadores en acabar. Saber si la respuesta de un clasificador es la primera sirve para no enviar la resolución de manera repetida al usuario, ya que la clasificación del primer clasificador en acabar es la usada para enviar la respuesta al usuario. Por último, haya sido o no el primer clasificador en acabar, se lanza en un hilo de ejecución la función store del objeto *Cache* para que este gestione la información a almacenar para las siguientes peticiones.

La Figura 4.7 representa cómo el *FrontEnd* envía la resolución DNS mediante la función *sendResponse*. En esta lo primero que se comprueba es la clasificación del modelo. Si el dominio no se ha detectado como malicioso, la petición se reenvía al servidor DNS configurado para resolverla y enviar dicha resolución al usuario. En caso contrario, se envía una resolución falsa al usuario y se ejecuta en un hilo la función *storeForReport* del objeto *Cache* para almacenar esta petición como maliciosa y reportarla al administrador.

Por otro lado, en la Figura 4.8 se muestra la forma en la que los clasificadores realizan la clasificación. Primero, detectan si el dominio es malicioso. Una vez que han terminado, acceden a la *SharedMemory* en exclusión mutua para comprobar si han sido los primeros en realizar la clasificación y, en ese caso, así responderlo al *FrontEnd*. Antes de liberar la memoria compartida y en caso de haber sido los primeros, realizan las actualizaciones oportunas en ella para que los demás clasificadores sepan que otro modelo ha acabado antes.

Por último, en la Figura 4.9 se observa de manera detallada cómo el objeto *Cache* realiza la función store para almacenar las clasificaciones. Primero se obtiene la marca de tiempo actual (*timestamp*), útil para guardar junto con la clasificación y eliminar clasificaciones antiguas en un futuro. Seguidamente se comprueba qué clasificador ha aportado la información a almacenar. En este TFG, el resultado de clasificación del modelo LSTM tiene prioridad con respecto al resultado del resto de clasificadores. De esta forma, si el algoritmo usado es LSTM, se almacena o actualiza el registro en caché; mientras que si es Random Forest, únicamente trata de almacenarlo y en caso de que

ya exista un registro relacionado con ese dominio no se actualiza.

4.4. Implementación del sistema

El sistema ha sido desarrollado en Python y su código fuente se ha liberado bajo licencia GNU/GPLv3 para que pueda ser usado y mejorado por la comunidad [7].

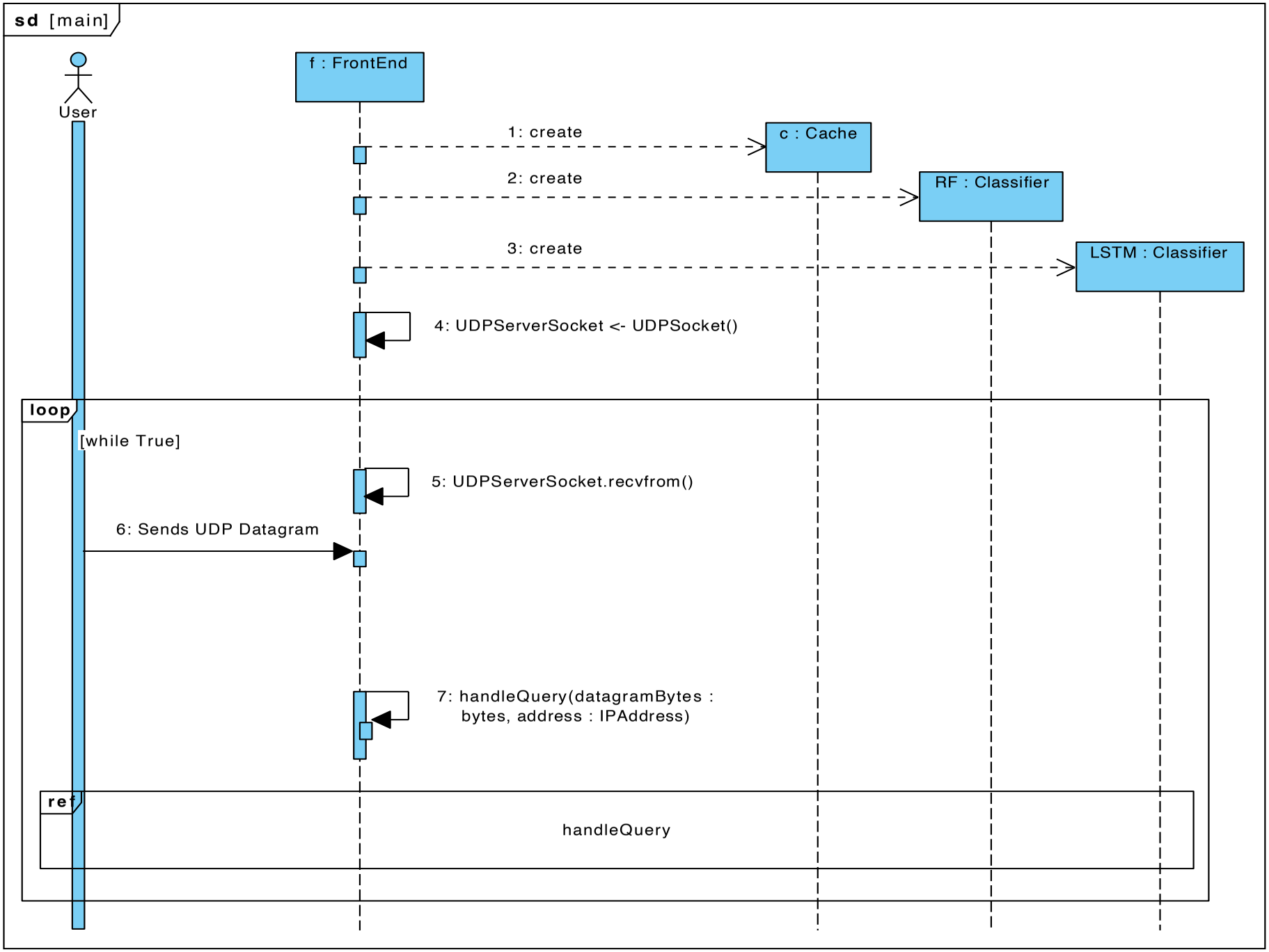

Figura 4.5: Diagrama de secuencia Main.

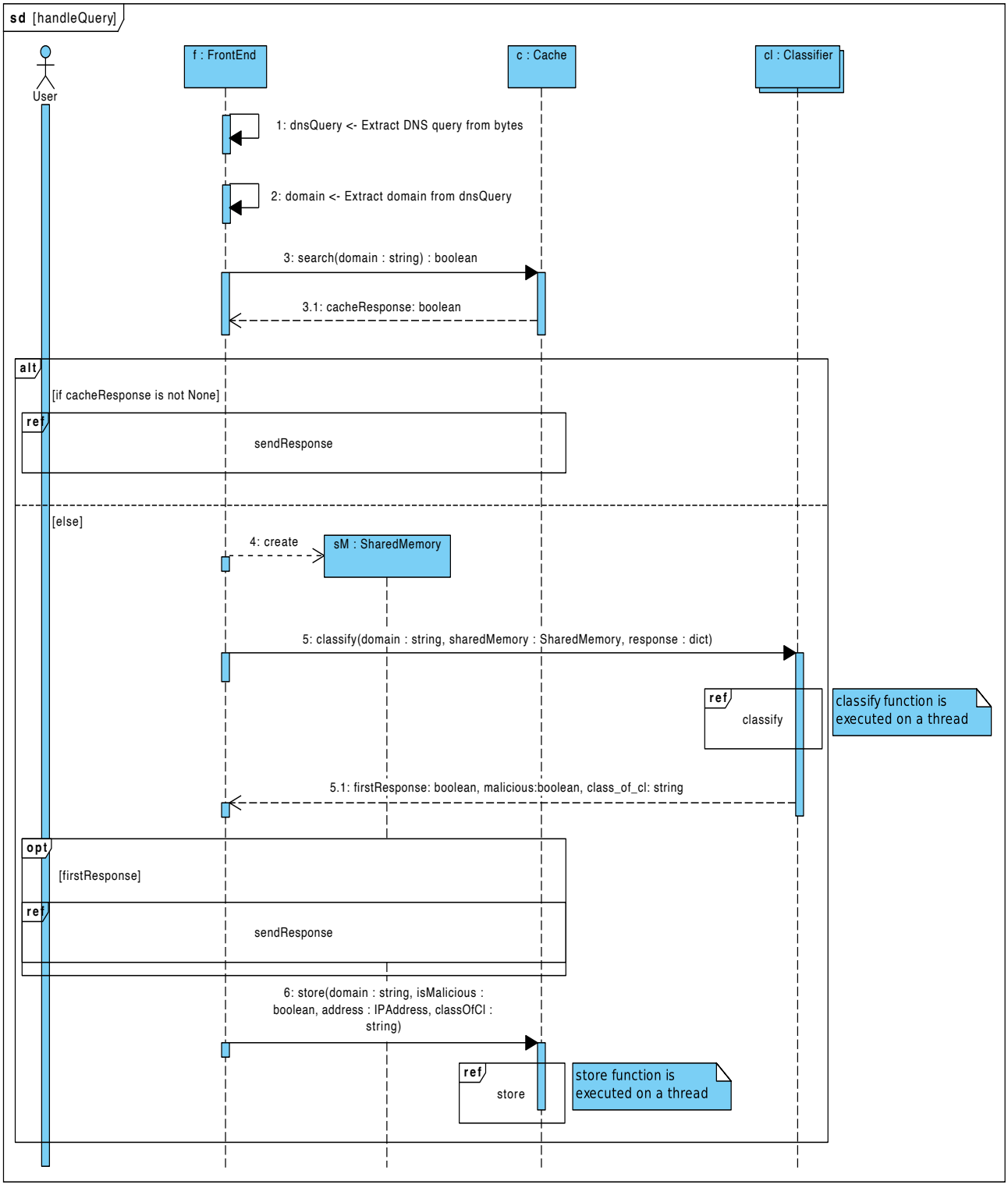

Figura 4.6: Diagrama de secuencia Handle Query.

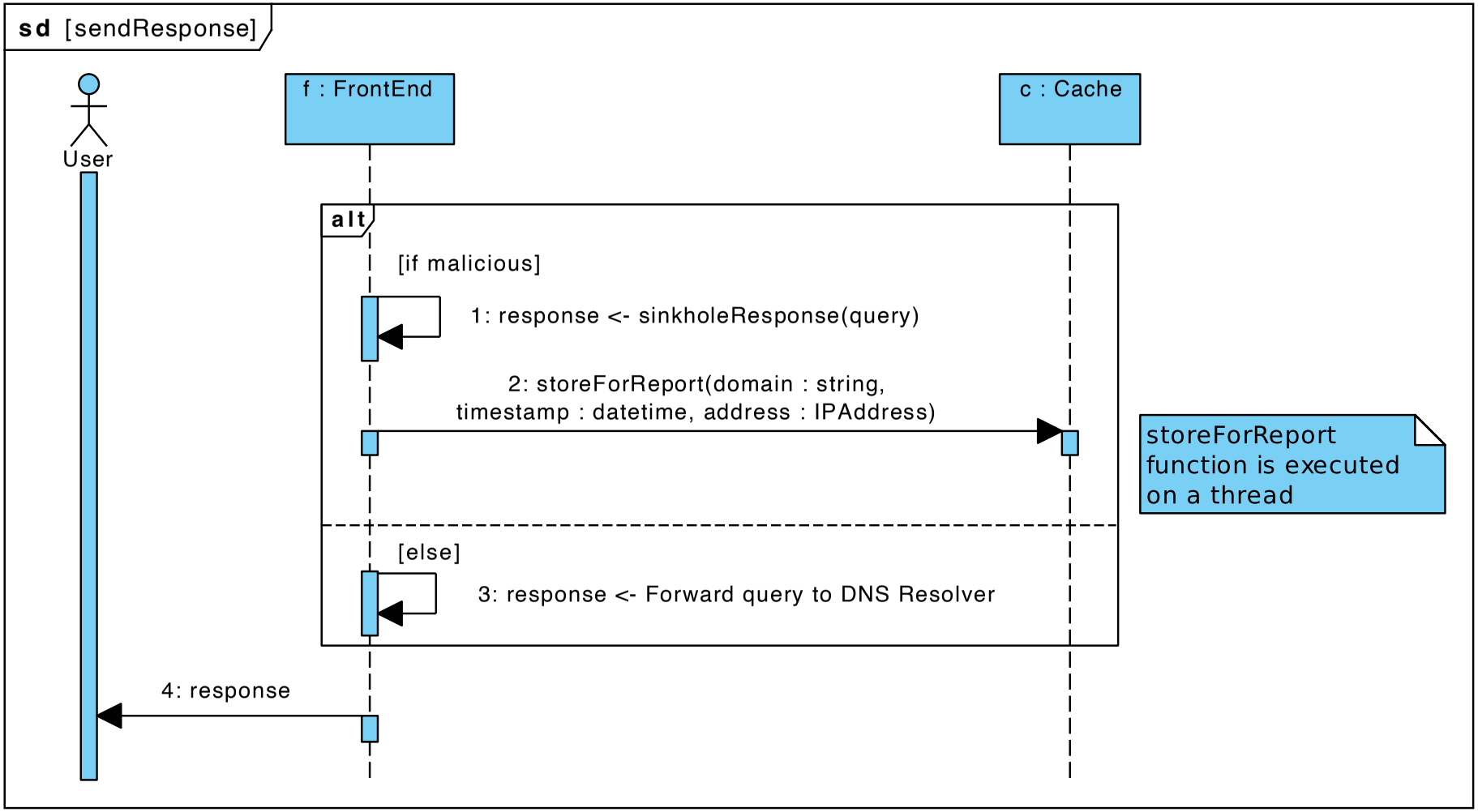

Figura 4.7: Diagrama de secuencia Send Response.

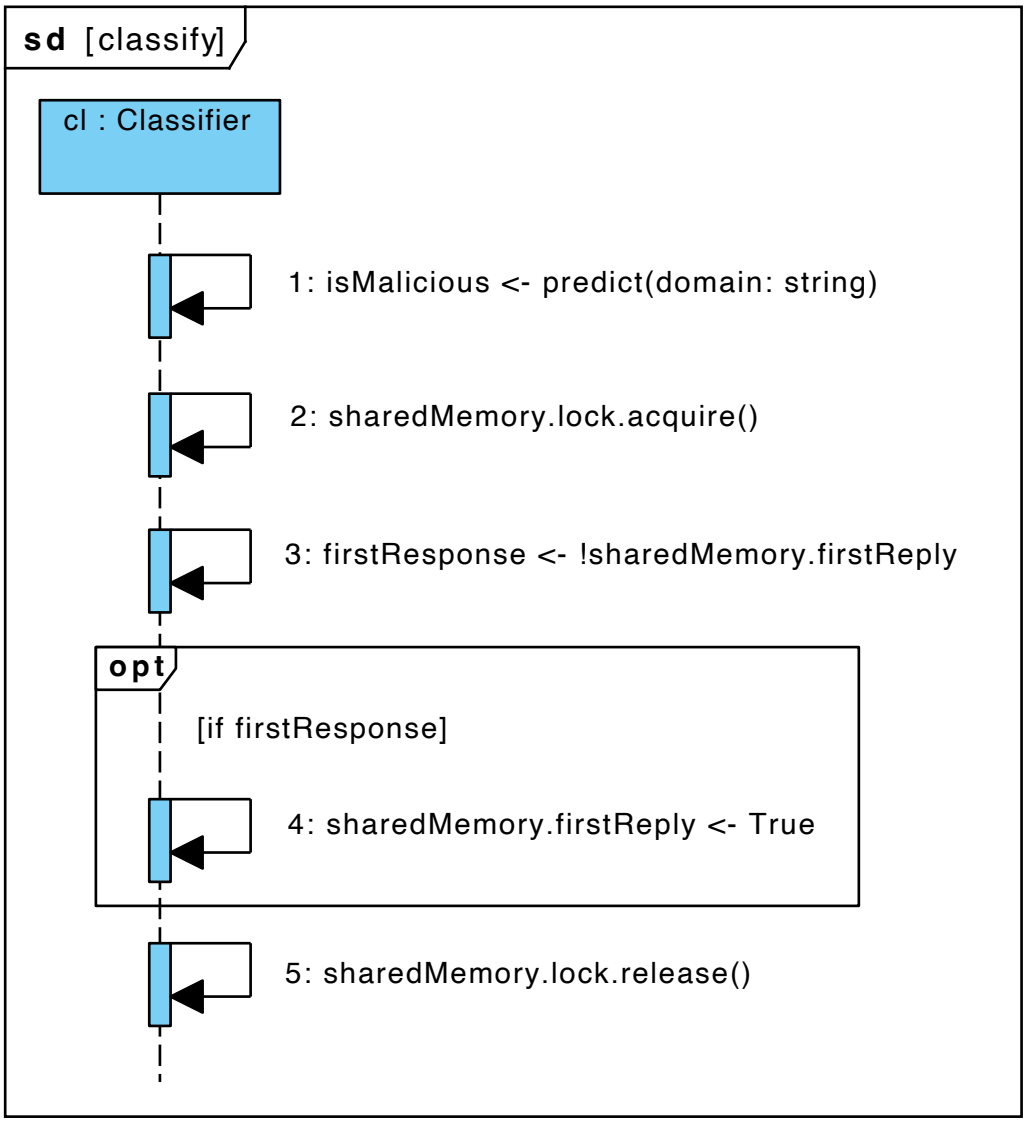

Figura 4.8: Diagrama de secuencia Classify.

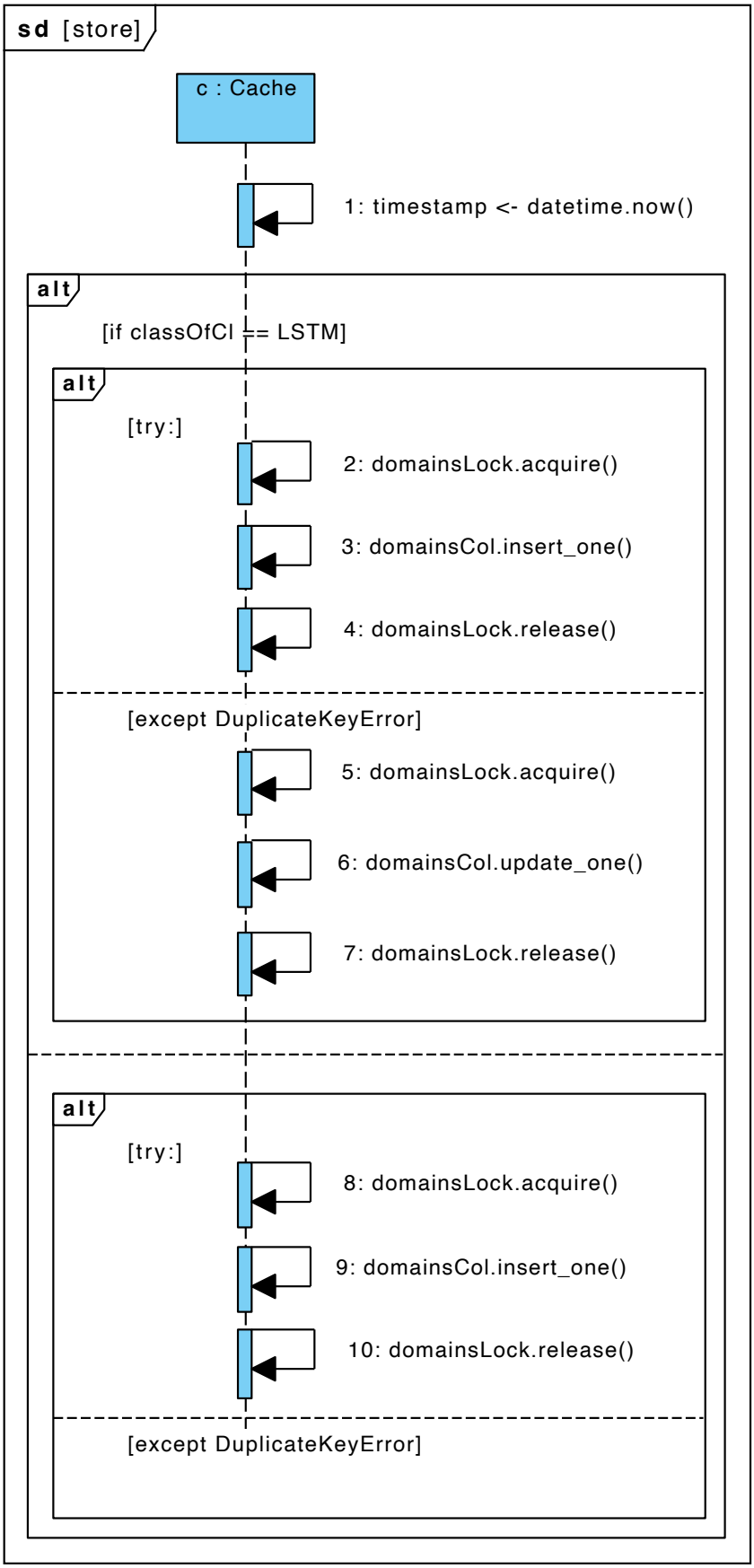

Figura 4.9: Diagrama de secuencia Store.

Capítulo 5

Experimentos y resultados

Para obtener las conclusiones del sistema desarrollado en este trabajo, se han realizado una serie de experimentos teniendo en cuenta varios aspectos. Lo primero que se ha estudiado ha sido el tiempo de resolución para una petición DNS. También se han obtenido las métricas necesarias para evaluar el rendimiento de los dos modelos del clasificador, una vez implementados en el sistema. Por último, se ha evaluado la mejora que supone tener el componente *Cache* en el sistema.

5.1. Entorno de experimentación

El estudio del tiempo de resolución es posible únicamente para los dominios legítimos y por tanto su entorno de experimentación ha sido diferente. Esto es debido a que para los dominios legítimos sí que es posible evaluar su resolución, ya que están correctamente registrados. Por el contrario, la resolución de la mayoría de los dominios maliciosos no es posible puesto que no están registrados o son bloqueados por los servidores DNS.

5.2. Tiempo de resolución de dominios legítimos

El dataset escogido para la experimentación es el que ofrece de forma gratuita Tranco [23]. Esta lista está orientada a la investigación y ofrece un ranking de las principales páginas web visitadas.

El sistema desarrollado en este trabajo tiene una alta dependencia de la velocidad de Internet, ya que para obtener la resolución del dominio necesita reenviar la petición a un servidor DNS. Esta dependencia influye en el tiempo de respuesta, que es una de las características a analizar. Para obtener unos resultados válidos a pesar de esta dependencia, la forma de realizar la experimentación ha sido la siguiente:

1. Se extrae el dominio de la lista [23] para realizar la petición.

2. Se realiza la consulta a un servidor DNS sin pasar por el microservicio.

3. Se realiza la consulta al microservicio y éste la reenvía al mismo servidor DNS en caso de no ser clasificado como malicioso.

Este procedimiento está enfocado a evaluar bajo la misma velocidad de Internet una petición directa al servidor frente a una petición a ese mismo servidor, pero pasando antes por el sistema desarrollado en este trabajo.

Además de los tiempos de resolución, se ha obtenido el resultado de la clasificación del microservicio. Realmente, teniendo en cuenta la implementación del sistema y la mayor velocidad de Random Forest frente a LSTM, la clasificación en una primera instancia es únicamente de Random Forest. Por ello, los pasos anteriores se han repetido para, no solo obtener la clasificación de LSTM, sino ver la mejora del tiempo de resolución gracias al elemento Cache.

La Figura 5.1 muestra el resultado de estos experimentos. Realizando la clasificación de 15.000 dominios legítimos de la manera mencionada se ha visto que el tiempo medio de una resolución DNS pasando por el microservicio previo es de 0.1768 segundos, mientras que enviando directamente la petición al servidor es de 0.1334 segundos.

Esto implica que el sistema hace que la resolución DNS sea un 32,55 % más lenta que si se enviase directamente al servidor. Tras la primera resolución de los 15.000 dominios, la clasificación ya se encuentra almacenada en el objeto *Cache* y por tanto se ha vuelto a estudiar la media de los tiempos. En este caso, el tiempo medio cuando se envía la petición al sistema desarrollado es de 0.1162 segundos, mientras que cuando es enviada directamente es de 0.1274 segundos. Cabe destacar que el microservicio ha sido un 8,78 % más rápido que preguntando directamente al servidor. Este resultado se discutirá a continuación.

5.3. Rendimiento de los modelos

Para la evaluación del rendimiento de los modelos se ha utilizado un dataset formado por 30.000 dominios, siendo la mitad de ellos dominios legítimos de la lista Tranco y la otra mitad dominios maliciosos obtenidos del dataset generado en [21]. Además, se han obtenido las métricas de precisión (*precision*), exhaustividad (*recall*), valor-F (*F1-score*) y la matriz de confusión (*confusion matrix*).

Los valores de la clasificación binaria utilizados son los siguientes:

- 0: dominio legítimo

- 1: dominio malicioso

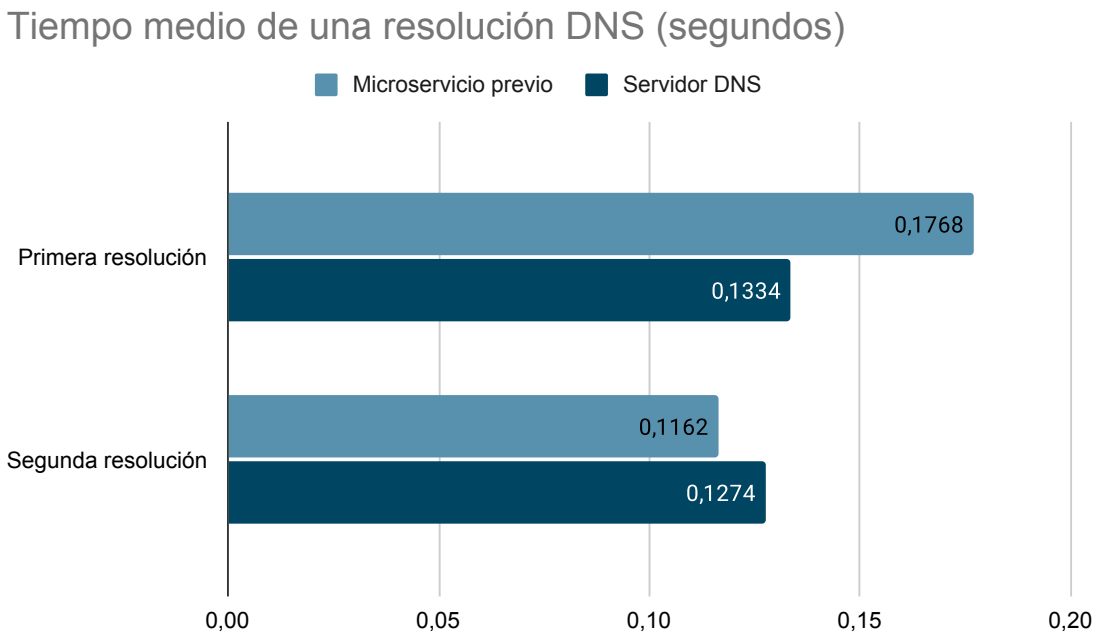

Figura 5.1: Gráfico del tiempo medio de una resolución DNS.

Las métricas precision, recall, y F1-score se calculan como se muestran en las ecuaciones 5.1, 5.2 y 5.3, respectivamente. True Negative (TN) y True Positive (TP) son, respectivamente, el número de predicciones correctas de dominios legítimos y maliciosos. Por el contrario, False Negative (FN) y False Positive (FP) son, respectivamente, el número de predicciones incorrectas de dominios legítimos y maliciosos.

$$precision = \frac{TP}{TP + FP} \tag{5.1}$$

$$recall = \frac{TP}{TP + FN} \tag{5.2}$$

$$F1 - score = 2 \times \frac{precision \times recall}{precision + recall} \tag{5.3}$$

La matriz de confusión indica qué tipos de errores se cometen. La métrica de precisión representa qué porcentaje de los dominios que se predicen como maliciosos, en realidad lo son. El recall indica qué porcentaje de dominios maliciosos ha sido capaz de identificar correctamente el modelo. El valor F1-score combina las medidas de precisión y recall en un único porcentaje para que sea más fácil comparar el rendimiento entre varios modelos.

La Tabla 5.1 muestra la matriz de confusión de los modelos Random Forest y LSTM, mientras que los resultados de las métricas para ambos modelos se muestran

en la Tabla 5.2.

		Random Forest		LSTM	
		0	1	0	1
Realidad	0	13560	1440	14345	655
	1	498	14502	337	14663

Tabla 5.1: Confusion matrix de los modelos Random Forest y LSTM.

Métricas	Random Forest	LSTM
Precision	0.9097	0.9572
Recall	0.9668	0.9775
F1-score	0.9374	0.9673

Tabla 5.2: Métricas de los modelos Random Forest y LSTM.

5.4. Discusión de resultados

Con respecto al tiempo, el sistema ofrece unos resultados bastante óptimos. En un entorno con una velocidad de Internet suficiente como para navegar de forma fluida, la ralentización de la resolución en un 32,55 % es una demora asumible para que no se descarte la petición DNS y el usuario la de como perdida o no resuelta. Por otra parte, la caché hace que esa demora sea inexistente e incluso se consigue que la resolución se haga en un menor tiempo que cuando no existe el sistema clasificador. Las razones de que este tiempo sea menor pueden ser variadas (por ejemplo, la velocidad de Internet puede haber sido mayor en un momento puntual o el servidor DNS ha almacenado en su propia caché la resolución). No obstante, una de las razones principales por las que esto sucede se debe a que cuando un dominio es clasificado como malicioso no se realiza su resolución y, por tanto, el tiempo que el usuario tarda en recibir la resolución del microservicio es menor que si la hubiera reenviado al servidor DNS.

Con esto se puede observar que ambos modelos tienen un desempeño aceptable, aunque claramente destaca LSTM sobre Random Forest. Los dos modelos identifican los dominios maliciosos con un alto porcentaje: Random Forest con un 96,68 % y LSTM con un 97,75 %. En cambio, la métrica de precisión de Random Forest y su matriz de confusión muestran que se produce una cantidad de falsos positivos muy relevante, lo que provocaría el entorpecimiento de un usuario o dispositivo cuyos fines no son maliciosos. Esta conclusión está en línea con la expresada en [21]. El número de falsos negativos de ambos modelos también es considerable, teniendo en cuenta que los dominios no identificados como maliciosos pueden servir para establecer la conexión con el C2, lo cual causaría la consecución de las acciones maliciosas del atacante. Por

último, la métrica F1-score muestra que el rendimiento del modelo LSTM frente al de Random Forest es aproximadamente un 3 % mejor.

Capítulo 6

Conclusiones y trabajo a futuro

Este capítulo contiene las conclusiones obtenidas de este trabajo y se sugieren una serie de trabajos futuros. Comienza con los objetivos logrados y un análisis de los resultados obtenidos en relación a estos objetivos, y termina proponiendo una serie de acciones como trabajo a futuro para mejorar el sistema y realizar un análisis más realista del mismo.

6.1. Conclusiones

Se puede concluir que el trabajo ha logrado los objetivos que se propusieron en un inicio. Se ha desarrollado un sistema que añade de forma funcional una capa de seguridad adicional a cualquier servidor DNS para la detección y prevención de malware que emplee la técnica DGA para establecer conexión con el C2. Además, el sistema es lo suficientemente desacoplado como para que la inserción de otras técnicas de detección o nuevos modelos de aprendizaje automático sean fácilmente incorporados al microservicio.

A la vista de los resultados, se puede decir que el tiempo que añade el interponer el microservicio entre el usuario y el servidor DNS no es tan alto como para hacer que el servicio de resolución falle. Incluso disponer de una caché en el microservicio hace que el retardo llegue a ser inexistente, debido a no tener que realizar de nuevo la clasificación del dominio. Por otra parte, los modelos usados en el clasificador combinado ofrecen un desempeño correcto y funcional para el propósito de este trabajo. Las métricas indican un alto porcentaje de acierto a la hora de identificar los dominios maliciosos generados por DGAs, pero también un alto número de falsos positivos que suponen un entorpecimiento de la experiencia del usuario en entornos reales al bloquear nombres de dominio legítimos. Además, a pesar de que el número de falsos negativos es muy bajo, idealmente este número tendría que ser lo más cercano a 0 ya que se evitaría a toda costa la conexión del malware con el C2.

6.2. Trabajo a futuro

Uno de los principales aspectos a mejorar son las métricas de clasificación. Se pueden ajustar los parámetros de los modelos para obtener unos mejores resultados o incluso proponer nuevos modelos para incorporarse al microservicio.

Por otra parte, y para obtener unos resultados más realistas, se debería de evaluar el sistema con tráfico real de una red. Para ello, puede redirigirse una copia del tráfico DNS de una red real para que se clasificara y resolviera. Con esto se obtendrían resultados con una carga de red y una implementación más realista, lo que aportaría una visión más detallada de los factores a mejorar.

Por último, se propone estudiar la incorporación de otros tipos de detección que trabajasen junto con el clasificador para obtener unos resultados más óptimos. Estas detecciones pueden apoyarse en características más volátiles como el número de peticiones fallidas o el valor del TTL, tal y como se ha realizado en otros estudios. De esta forma, se puede desarrollar un sistema de detección combinada que aprovecha las ventajas del análisis basado en características volátiles (tráfico de red) y persistentes (nombres de dominio).

Bibliografía

[1] Manos Antonakakis et al. "From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware". En: *21st USENIX Security Symposium (USENIX Security 12)*. Bellevue, WA: USENIX Association, ago. de 2012, págs. 491-506. ISBN: 978-931971-95-9. URL: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/antonakakis.

[2] Johannes Bader. *Some results of my DGA reversing efforts*. [Online; https://github.com/baderj/domain_generation_algorithms]. Accessed on November 23, 2022. Nov. de 2022.

[3] Leyla Bilge et al. "Exposure: A Passive DNS Analysis Service to Detect and Report Malicious Domains". En: *ACM Trans. Inf. Syst. Secur.* 16.4 (abr. de 2014). ISSN: 1094-9224. DOI: 10.1145/2584679. URL: https://doi.org/10.1145/2584679.

[4] Leyla Bilge et al. "EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis." En: ene. de 2011.

[5] Stéphane Bortzmeyer y Shumon Huque. *NXDOMAIN: There Really Is Nothing Underneath*. RFC 8020. Nov. de 2016. DOI: 10.17487/RFC8020. URL: https://www.rfc-editor.org/info/rfc8020.

[6] Guy Bruneau. "DNS Sinkhole". En: *SANS Institute* (2010).

[7] Víctor Mateo Calvillo. *DGA DNS Detection*. [Online; https://github.com/victormatcal/DNS-DGA-detection]. Accessed on November 23, 2022. Nov. de 2022.

[8] *Domain names - implementation and specification*. RFC 1035. Nov. de 1987. DOI: 10.17487/RFC1035. URL: https://www.rfc-editor.org/info/rfc1035.

[9] Hans-Erik Eriksson y Magnus Penker. "Business modeling with UML". En: *New York* 12 (2000).

[10] Refactoring Guru. *Strategy pattern*. [Online; https://refactoring.guru/design-patterns/strategy]. Accessed on November 23, 2022. Nov. de 2022.

[11] Cybercrime Magazine. *2022 Cybersecurity Almanac: 100 Facts, Figures, Predictions And Statistics*. [Online; https://cybersecurityventures.com/cybersecurity-almanac-2022/]. Accessed on November 23, 2022. Nov. de 2022.

[12] Lockheed Martin. *Lockheed Martin*. [Online; https://www.lockheedmartin.com/]. Accessed on November 23, 2022. Nov. de 2022.

[13] Lockheed Martin. *The Cyber Kill Chain*. [Online; https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html]. Accessed on November 23, 2022. Nov. de 2022.

[14] Lockheed Martin. *The Cyber Kill Chain Image*. [Online; https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/photo/cyber/THE-CYBER-KILL-CHAIN-body.png.pc-adaptive.1920.medium.png]. Accessed on November 23, 2022. Nov. de 2022.

[15] MITRE ATT&CK. *Command and Control*. [Online; https://attack.mitre.org/tactics/TA0011/]. Accessed on November 23, 2022. Nov. de 2022.

[16] MITRE ATT&CK. *Dynamic Resolution*. [Online; https://attack.mitre.org/techniques/T1568/]. Accessed on November 23, 2022. Nov. de 2022.

[17] MITRE ATT&CK. *Dynamic Resolution: Domain Generation Algorithms*. [Online; https://attack.mitre.org/techniques/T1568/002/]. Accessed on November 23, 2022. Nov. de 2022.

[18] MITRE ATT&CK. *MITRE ATT&CK*. [Online; https://attack.mitre.org/]. Accessed on November 23, 2022. Nov. de 2022.

[19] Daniel Plohmann et al. "A Comprehensive Measurement Study of Domain Generating Malware". En: *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, ago. de 2016, págs. 263-278. ISBN: 978-1-931971-32-4. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/plohmann.

[20] "Proceedings. 19th Annual Computer Security Applications Conference". En: *19th Annual Computer Security Applications Conference, 2003. Proceedings.* 2003. DOI: 10.1109/CSAC.2003.1254303.

[21] Jose F. Selvi Sabater. "Técnicas de aprendizaje automático parala detección de dominios maliciososgenerados algorítmicamente". En: (2021).

[22] Duc Tran et al. "A LSTM based framework for handling multiclass imbalance in DGA botnet detection". En: *Neurocomputing* 275 (2018), págs. 2401-2413.

[23] Tranco. *Tranco List*. [Online; https://tranco-list.eu/]. Accessed on November 23, 2022. Nov. de 2022.

[24] *User Datagram Protocol*. RFC 768. Ago. de 1980. DOI: 10.17487/RFC0768. URL: https://www.rfc-editor.org/info/rfc768.

[25] Ian H. Witten y Eibe Frank. "Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations". En: *SIGMOD Rec.* 31.1 (mar. de 2002), págs. 76-77. ISSN: 0163-5808. DOI: 10.1145/507338.507355. URL: https://doi.org/10.1145/507338.507355.

[26] Woodbridge, Jonathan and Anderson, Hyrum S and Ahuja, Anjum and Grant, Daniel. "Predicting domain generation algorithms with long short-term memory networks". En: *arXiv preprint arXiv:1611.00791* (2016).