UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

INGENIERÍA EN INGENIERÍA INFORMÁTICA

Analizador de mensajes de correo

Subtítulo del proyecto

Autor
Pedro Luis Fuertes Moreno

Directores
Alberto Guillén Perales
Gabriel Maciá Fernández

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, julio del 2020

Analizador de mensajes de correo: Subtítulo del proyecto

Pedro Luis Fuertes Moreno

Palabras clave: Correo electrónico, ciberseguridad, phishing, malware, virus

Resumen

Este proyecto surge debido a la falta de herramientas, tanto para usuarios técnicos como domésticos, para analizar un correo sospechoso una vez que llega a la bandeja de entrada.

El objetivo de este proyecto es, por tanto, intentar proporcionar una ayuda extra en la identificación de correos maliciosos mediante la extracción y relación de características comunes. Por otro lado, se ofrecerá un servicio público y funcional para que los usuarios puedan aprovechar toda la información recopilada y analizar sus propios correos.

A lo largo del documento se hablará de los distintos tipos de correos maliciosos, de algunas técnicas usadas por los atacantes para engañar o manipular a sus víctimas, de los patrones que se van a extraer y cómo, de las relaciones que se van a hacer.

En la parte de diseño se analizarán y compararán tanto lenguajes de programación como bases de datos, teniendo la idea en mente de migrar todo el servicio a la nube para tener un SaaS, siendo especialmente relevante el servicio de Azure Functions para ejecutar el código. En la parte funcional, el servicio debe permitir tanto analizar como buscar resultados, así como mostrar información relevante derivada del análisis.

También se indicarán problemas encontrados, soluciones aplicadas y posibles mejoras.

Finalmente, y como objetivo último se intentarán obtener ingresos económicos por parte del servicio de cara a crear una posible startup o venta del servicio.

Analizador de mensajes de correo: Subtítulo del proyecto

Pedro Luis Fuertes Moreno

Keywords: e-mail, cybersecurity, phishing, malware, virus

Abstract

The aim of this project is to try to offer extra assistance in the identification of malicious emails through the extraction and correlation of common characteristics. Furthermore, a public and functional service will be offered so that users can benefit from the information gathered and analyze their own emails.

This project stems from the lack of tools —for technical as well as for home users— to analyze suspicious emails in their inbox.

This document exposes the different types of malicious emails, some of the techniques used by attackers to deceive or manipulate their victims, the patterns that will be extracted and how to extract them, and the connections between then.

Moreover, design-wise, programming languages and data bases will be analyzed and compared in order to be able to migrate the whole service to the Cloud and create a SaaS. Azure Functions' service will be especially relevant to execute the code.

Functionally, the service must allow analyzing results as well as searching for them, just as showing relevant information derived from the analysis.

Yo, **Pedro Luis Fuertes Moreno**, alumno de la titulación Grado en ingeniería informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI XXXXXXXXX, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Pedro Luis Fuertes Moreno

Granada a 7 de julio de 2020.

D. **Alberto Guillén Perales**, Profesor del Área de XXXX del Departamento Departamento de ... de la Universidad de Granada.

D. **Gabriel Maciá Fernández**, Profesor del Área de XXXX del Departamento Departamento de ... de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Analizador de mensajes de correo, Subtítulo del proyecto***, ha sido realizado bajo su supervisión por **Pedro Luis Fuertes Moreno**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 7 de julio de 2020.

Los directores:

Alberto Guillén Perales **Gabriel Maciá Fernández**

Agradecimientos

A mi familia, en especial a mi padre, a mi madre, a mi hermano y a mi hermana.

A mis amigos.

A mis profesores.

Índice general

Lista de códigos

Capítulo 1

Introducción

Aunque en la actualidad poseemos una gran cantidad de aplicaciones con las que comunicarnos con otras personas, el correo electrónico sigue siendo una de las herramientas de comunicación más usadas, especialmente en el mundo empresarial.

Esto es debido a que fue uno de los primeros servicios en permitirnos enviar y recibir tanto texto como otros tipos de archivos de manera rápida y sencilla.

Según un estudio de hace tres años de The Radicati Group el correo electrónico tenía tres mil setecientos millones de usuarios, que enviaban doscientos sesenta y nueve mil millones de correos cada día [**?**].

Viendo las cifras anteriores se puede entender la importancia de tener un servicio de este tipo lo más seguro posible, ya que un ataque bien diseñado puede afectar a millones de personas.

Sin embargo, debido a cómo y cuándo crea el correo electrónico, la seguridad nunca ha sido uno de sus puntos fuertes y eso ha llegado hasta nuestros días.

1.1. Breve historia del correo electrónico

El correo electrónico se remonta a 1962 en el Massachusetts Institute of Technology (MIT), cuando compraron a IBM un ordenador que permitía que distintos usuarios iniciaran sesión y guardaran archivos en él. Estos lo aprovecharon para intercambiar mensajes, lo que provocó que para 1965 se desarrollara un servicio que facilitase esa comunicación entre los distintos usuarios y lo llamaron MAIL [**?**].

Hay que tener en cuenta que, en ese servicio, los mensajes no salían de dicho ordenador. Habría que esperar hasta 1971 para ver lo que sería el primer "correo electrónico" enviado a través de una red, en concreto de ARPANET y fue gracias a Ray Tomlinson. Él adaptó un programa que permitía enviar mensajes a distintos terminales de distintos usuarios de un mismo ordenador,

para poder enviar mensajes entre distintos terminales, aunque no estuviesen en el mismo ordenador. Precisamente el "@" del correo electrónico viene de la necesidad de Tomlinson de tener que separar al usuario del equipo, ya que anteriormente esto no era necesario [**?**].

No es hasta 1977 cuando se crea el primer rfc del correo electrónico, concretamente el rfc733 [**?**], aunque, este protocolo no es usado en la actualidad. El primer rfc del primer protocolo que aún se usa es el rfc821 [**?**] de Simple Mail Transfer Protocol (SMTP) de 1982 y el rfc918 [**?**] de Post Office Protocol (POP) de 1984.

La situación por aquél entonces de lo que ahora conocemos como Internet, era muy distinta. Internet estaba reservado a universidades, centros de investigación e instituciones gubernamentales. Esto hizo que cuando se desarrollasen estos protocolos, no se pensara en la seguridad de ellos.

Y este es uno de los grandes problemas que tiene el correo en la actualidad, ya que, aunque tanto SMTP como POP (E Internet Message Access Protocol (IMAP) [**?**] aunque no se ha mencionado antes) han ido recibiendo actualizaciones, están basados en unos protocolos diseñados y pensado para un entorno radicalmente diferente en el que se siguen utilizando.

1.2. Origen del proyecto

Este proyecto surge debido a la gran cantidad de demandas que he recibido en los últimos años por parte de familiares y amigos para que, les ayudase a verificar si un correo sospechoso que les había llegado a su bandeja de entrada era o no malicioso.

Y esta tarea, que, para mí era trivial en la mayoría de los casos, no lo era para ellos. Aunque algunos estaban tan bien preparados que incluso a mí me costaba diferenciarlos.

Todo esto me llevó a pensar dos cosas. La primera, que una vez que un correo llega a la bandeja de entrada del usuario, sólo le queda su intuición para confiar o no en el mensaje, intuición que puede fallar incluso si se tienen conocimientos técnicos.

Por otro, que en caso de querer investigar dicho mensaje y obtener más información, no existe ninguna herramienta específica para ello, por lo que se tiene que hacer todo a mano y siendo imposible obtener algún tipo de relación con otro mensaje parecido, lo cual dificulta mucho el proceso.

1.3. El proyecto

El objetivo de este proyecto es tratar de paliar esta carencia de herramientas tanto para la identificación de correos maliciosos una vez que han pasado los filtros de spam, como para la investigación de un mensaje en los casos más complicados.

Por este motivo se va a crear por un lado, un servicio que permita analizar mensajes *on-line*, para que cuando se reciba un mensaje sospechoso, el usuario tenga una segunda validación con más información y por otro lado, se va a permitir al usuario poder "navegar" entre los datos encontrados en el correo para poder obtener más información, lo que puede facilitar en gran medida un análisis más profundo del correo por parte de una persona más técnica en caso de ser necesario.

Para llevar esto acabo se debe crear una gran base de datos con todos los correos analizados, así como los distintos datos que se han extraído de los mismos y sus relaciones.

También se deben identificar patrones maliciosos conocidos, para poder alertar a los usuarios menos especializados sobre los patrones encontrados sin necesidad de que ellos sepan en qué consisten.

Capítulo 2

Motivación

En la actualidad los ataques por correo electrónico siguen siendo un hecho y prácticamente todo el mundo ha recibido algún correo de este tipo alguna vez, lo que demuestra que, las herramientas actuales no son capaces de solucionar de manera efectiva este problema.

Esto se suma a que cada vez los ataques son más y más sofisticados, y, por tanto, complicados de detectar, ya no solo por estas herramientas, sino, por las propias personas, sean o no profesionales del sector. Y es que cuando un usuario recibe un mensaje de este tipo no tiene ninguna herramienta extra que le ayude a comprobar si es o no malicioso.

Esto es así hasta tal punto, que la solución para identificar estos correos de grandes empresas antivirus dedicadas a la ciberseguridad es que los usuarios sigan su intuición [**?**], otras ofrecen pequeños cursos para identificarlos [**?**].

Todo esto refleja como los usuarios están en una clara situación de vulnerabilidad, especialmente los usuarios menos técnicos que no conocen cómo funciona realmente la tecnología y que están usando ¿Y es que acaso deberían?

Bajo mi punto de vista, el enfoque de las grandes compañías sobre cómo atacar los problemas de seguridad que tienen el correo electrónico, está dirigido a personas con unos conocimientos que la mayoría de las personas no tienen y por tanto sólo es útil para una minoría de usuarios del servicio.

Pero además, es que dichas compañías tampoco están exentas de ataques de esta naturaleza, esto se pone de manifiesto en algunos ataques llevados a cabo con éxito a empresas tecnológicas de máximo nivel, como pueden ser Google o Facebook, a las que un atacante les consiguió robar 121 millones de euros [**?**]. Y aunque no han trascendido más detalles del ataque, se podría haber utilizado lo que se conoce como ingeniería social, haciéndose pasar por empresas comerciales conocidas y utilizando detalles reales de estas, como firmas o logotipos. De este tipo de ataques ya advertía Kaspersky Lab en 2018. [**?**]

Todo esto lleva a pensar que actualmente sigue habiendo un gran agujero de seguridad en el correo electrónico y que las herramientas existentes no son suficientes ni siquiera para los expertos del sector.

2.1. Aplicaciones relacionadas

A continuación, se van a analizar un conjunto de herramientas que, si bien no ofrecen soluciones completas a este problema, pueden ser buenos servicios en los que apoyarse para tratar de dar una solución más amplia y completa que las actuales.

2.1.1. Herramientas específicas del correo electrónico

Las herramientas descritas en esta sección únicamente son válidas en correos electrónicos, por lo que, si en el futuro también se desean analizar otro tipo de mensajes, o no se tienen todas las cabeceras asociadas al mensaje, no serán válidas.

Filtros de correo no deseado

Tal vez sea la mejor solución que se tiene actualmente para mitigar este tipo de problemas. Son filtros que analizan cada uno de los mensajes que se reciben y en base a distintos criterios informan al usuario si el correo es o no legítimo.

Aunque, este tipo de herramientas tienen varios problemas asociados:

- La imposibilidad de detectar todos los correos maliciosos: Conseguir una herramienta con una efectividad del 100 % es prácticamente imposible y esto es algo que se debe asumir. Sin embargo, y debido por un lado a la gran cantidad de correos que son enviados cada día [**?**] y por otro que en torno a la mitad (52.48 %) son correos no deseados [**?**] hace que sea prácticamente imposible detectarlos todos.
- La imposibilidad de poner un filtro de este tipo: hay muchos servicios de correo electrónico que no permiten al usuario poner un filtro personalizado de correo no deseado. Ejemplos de estos servicios son Gmail u Outlook, 1500 [**?**] y 400 [**?**] millones de usuarios respectivamente (¡Son muchos!)
- Una vez que el mensaje pasa el filtro, esta herramienta deja de ser efectiva, lo cual puede ser muy sencillo para un atacante, simplemente va probando con distintos correos hasta que consigue uno que lo pase.

Un servicio de este tipo puede ser muy útil para un primer análisis si el mensaje es compatible. Normalmente tienen una larga trayectoria, son rápidos y tienen una efectividad comprobada.

2.1.2. Analizador de cabeceras

Un analizador de cabeceras de correo electrónico da información relevante y normalmente invisible al usuario sobre dicho correo. Estas cabeceras pueden aportar información muy útil como por ejemplo la ruta seguida por el mensaje, la IP del servidor de correo desde donde se envió o si ha pasado o no los filtros antispam del proveedor de correo.

Actualmente hay varias páginas que ofrecen este servicio, como por ejemplo Google [**?**], Mircrosoft [**?**] o Mxtoolbox [**?**].

Ofrecer este tipo de análisis es muy interesante, ya que permite de manera sencilla hacer un análisis más profundo por parte de un profesional del sector.

2.2. Herramientas generales

Las herramientas que se describen a continuación, si bien no están destinadas al correo electrónico como tal, se pueden usar de manera efectiva para analizar los distintos mensajes que se reciban, sean o no correos electrónicos.

Virus Total

Virus Total [**?**] es una web propiedad de Google que permite analizar archivos, y direcciones web en busca de programas malignos y aunque no está directamente relacionada con el correo electrónico puede servir de ayuda para analizar posibles archivos adjuntos, así como posibles direcciones sospechosas.

Tener una herramienta que enlace directamente con el servicio puede ser de gran ayuda tanto para profesionales del sector como para usuarios menos especializados, que lo único que tendrán que hacer es clicar en un botón para analizar una dirección de su correo electrónico.

Virus total ofrece una api rest que permite analizar tanto archivos como urls, dominios e ips. [**?**]

Metadefender

Metadefender [**?**] es un analizador de archivos, url's, dominios e ips similar a Virus Total de la empresa de ciberseguridad Opswat.

Puede ser una buena alternativa a Virus Total y al igual que este tiene una api pública [**?**] en la que realizar consultas.

Have I been pwned

Have I been pwned [**?**] es una web que permite saber si una dirección de correo electrónico ha aparecido en alguna brecha de seguridad, y en caso de que haya aparecido te dice en qué brecha ha sido.

Saber si el mensaje proviene de una dirección comprometida puede ser de relevancia, siendo más probable que un mensaje malicioso provenga de una dirección comprometida que de una dirección que no lo sea.

Have I been pwned tiene una api [**?**] donde realizar consultas sobre direcciones de correo electrónico, además su base de datos se va actualizando con las últimas brechas de seguridad que van surgiendo y contando ya con más de 9.500 millones de cuentas de correo.

Capítulo 3

Objetivos y requisitos

3.1. Obtención y relación de patrones

Sacar mediante expresiones regulares datos de interés de correos electrónicos y relacionarlos entre sí. Esto permitirá a los expertos del sector contar con una herramienta para poder realizar análisis de mayor profundidad al poder relacionar datos comunes entre los distintos correos de la base de datos.

Un ejemplo sencillo de esto puede ser el análisis de un correo electrónico nuevo, pero con una url ya conocida y presente en otros correos electrónicos.

Un ejemplo más complejo podría ser, el análisis de un mensaje con una url nueva, pero con un dominio de cuya IP sí se tienen registros, lo que puede dar lugar a una nueva línea de investigación sobre si dicho dominio pertenece (O no) a un cibercriminal, aunque no se tengan registros previos ni del dominio ni de la url en cuestión.

Se debe extraer al menos los siguientes tipos de datos:

- Direcciones IP
- Dominios
- Enlaces
- Direcciones de correo electrónico

Aunque el servicio se debe pensar para que en un futuro se puedan extraer más tipos de datos como carteras de criptomonedas o números de teléfono.

3.2. Evaluar de manera relativa la maliciosidad

Se deben detectar ciertas técnicas comúnmente usadas por los ciberdelincuentes para atacar a sus víctimas y así asignar un valor de maliciosidad tanto a los datos extraídos como a los mensajes en sí.

Algunas de estar técnicas podrían ser:

- Mostrar un enlace distinto al que se redirige.
- Usar una gran cantidad de subdominios de subdominios
- Mostrar un correo electrónico distinto del real

3.3. Ofrecer un servicio de análisis público

La segunda parte del proyecto será ofrecer a todos los usuarios la posibilidad de analizar sus mensajes mediante una web donde podrán o bien copiar y pegar el mensaje, o bien subir un archivo eml para un análisis más completo.

La página devolverá varias listas con todos los tipos datos extraídos del análisis, así como enlaces a cada uno de ellos donde se muestre un informe más detallado.

Esto permitirá que a un usuario que le llegue un correo y no sepa si es o no malicioso, pueda analizarlo en la página y obtener más información sobre este.

3.4. Cumplir de manera efectiva con la ley de protección de datos

Es de vital importancia diseñar el servicio pensando en las leyes de protección de datos, especialmente en la europea por ser la más estricta hasta el momento.

Los usuarios deben poder eliminar, tanto los mensajes, como los datos obtenidos de ellos si así lo desean de manera automática y sin necesidad de que nadie intervenga en el proceso.

Aunque esta característica no se implemente en este proyecto, sí que se debe pensar el sistema para que se pueda implementar en el futuro de manera sencilla y modificando la mínima cantidad de código posible.

3.5. Monetización

Como parte complementaria se intentará pensar cómo obtener ingresos económicos del desarrollo. Estos ingresos nunca deben impedir que usuarios particulares puedan usar el servicio, aunque sí pueden impedir acceder a todas las funcionalidades de este.

3.6. Integración con terceros

Aunque esto está un poco al margen del TFG, sería muy interesante la integración directa desde la página con otros servicios como el de Virus Total o la de Have I been pwned.

Esto permitiría por un lado facilitar a los usuarios menos técnicos un análisis rápido desde la propia página y por otro puede dar información relevante a los expertos.

3.7. Crear un servicio funcional

Al final del proyecto se debe crear un servicio que dé un soporte real y online, no se debe limitar a tener un servicio local únicamente de prueba. Por lo que durante el apartado del diseño se deben elegir tecnologías factibles para su posterior despliegue en internet y por tanto no limitadas a un entorno de local pruebas.

Capítulo 4

Análisis y diseño

4.1. Análisis previo

Durante esta sección se analizarán cuestiones importantes que afectarán a múltiples secciones posteriores y que es necesario plantearse antes de comenzar, ya que pueden afectar de manera muy significativa a la elección y posterior diseño de futuros apartados.

De manera general, se deben tener en cuenta los siguientes factores:

- El tipo de datos que se van a guardar.
- El tipo y la cantidad relaciones entre los datos.
- La forma de identificar los datos.
- La forma de mostrar los datos.
- Cómo se va a buscar información y qué búsquedas se van a realizar.
- La cantidad de información que se va a almacenar.

4.1.1. Tipos de datos

Dentro de los tipos de datos se tienen que distinguir dos tipos, por un lado, están los mensajes que el usuario envía y por otro los datos que se extraen de ellos.

Tipos de datos a analizar

Dentro de los mensajes, este proyecto se centrará en mensajes de correo electrónico, por lo que hay fundamentalmente de tres tipos de datos, en texto plano, en HyperText Markup Language (HTML) [**?**] y en formato (EML).

A la hora de elegir un lenguaje, contar con alguna librería que sea capaz de analizar dichos tipos de archivos es algo crítico, puesto que hacer un

analizador de dichos tipos sería muy costoso y llevaría demasiado tiempo. Mientras la mayoría de los lenguajes modernos tienen librarías para leer archivos en HTML, no ocurre los mismo para el formato EML.

Sin embargo, el diseño no debe limitar la implementación de otros tipos de archivos tales como comma-separated values (CSV), Extensible Markup Language (XML) o JavaScript Object Notation (JSON).

Tipos de datos que se van a extraer

En un principio los datos que se van a extraer son enlaces, dominios, direcciones de correo, direcciones de Internet Protocol (IP) y en el caso de los archivos en formato EML, sus cabeceras.

4.1.2. Tipo y cantidad de relaciones

El tipo de relaciones serán normalmente de muchos a muchos, ya que de un único mensaje se pueden extraer varios datos de un mismo tipo, que a vez pueden estar múltiples mensajes.

Respecto a la cantidad de relaciones, mientras que de un correo no se deben extraer demasiados datos, un dato concreto puede aparecer en una gran cantidad de correos, por lo que en función del diseño que se haga, recuperar todos los correos en los que aparece un determinado dato podría llegar a ser una operación muy costosa.

4.1.3. Forma de identificar los datos

En este proyecto y debido a la naturaleza de los datos que se tratan, algo que pudiera ser trivial como es el hecho de identificar un dato concreto, se vuelve una tarea más compleja, pues estos pueden ser relativamente grandes y no se deben tratar de la misma manera que otros de menor tamaño, como tal vez puede ser un número o una fecha.

También es importante el hecho de poder "navegar" mediante enlaces, ya que no tendría sentido poner como dirección de un mensaje su propio contenido.

Este problema se puede solucionar de múltiples formas, una de ellas, consistiría en asignar a cada dato un valor numérico e incremental, de modo que se podría acceder al dato número 1, 2, ..., n. Este sistema permite tener un control del número de elementos que se han analizado, además ocupa muy poco (con 4 bytes por elemento se podrían numerar 4.294.967.296 elementos de dicho tipo), permitiría hacer búsquedas rápidas, al poder guardar en memoria una gran cantidad de elementos y no guarda relación alguna con el elemento al que identifica.

Sin embargo, también presenta los siguientes problemas, y es que, al no guardar relación con el elemento, no se puede obtener el identificador únicamente con el valor del dato, por lo que requiere de una consulta del

valor completo a la base de datos, que, en el caso de un mensaje, puede ser de gran tamaño.

Para evitar esta falta de relación entre el valor de un elemento y su identificación se puede utilizar una función hash, que sea rápida y que devuelva valores pequeños. Esto solucionaría la falta de correlación entre identificador y valor, facilitaría la navegación, permitiría búsquedas rápidas, (Constantes en teoría) y simplificaría las búsquedas de datos grandes como los mensajes.

Aunque también presenta problemas, el mayor de ellos es que ocupa mucho más espacio que una identificación numérica, lo que, en un sistema relacional, puede hacer que las uniones de tablas sean mucho más lentas. Además, también se pierde el orden de inserción, pero tampoco es importante en este caso.

Por tanto, una solución interesante puede ser adoptar ambos modelos, por un lado, tener un índice numérico e incremental y por otro un índice de tipo hash para buscar en caso de no tener el identificador numérico. Por ejemplo, para comprobar si un mensaje ya ha sido o no analizado se podría utilizar su hash, mientras que para las uniones de tablas de podría utilizar el identificador numérico.

Esta solución también presenta el problema de sobrecoste que conlleva guardar el hash y que puede ser mayor o menor en función de los bits que ocupe la función elegida.

Para ello, ha hecho un pequeño estudio del sobrecoste generado por cada millón de elementos insertados según algunas de las funciones hash más conocidas en este momento. No se va a tener en cuenta el coste computacional de dicha función al considerarlo relativamente bajo en todas ellas.

También se va a considerar la probabilidad de colisiones suponiendo que todos los valores son equiprobables. Para hacer este cálculo y teniendo en cuenta la paradoja del cumpleaños, se va a calcular la probabilidad de colisiones mediante la siguiente fórmula $k = \sqrt{2^n}$ siendo n el tamaño del hash en bits y k la probabilidad de que haya una colisión.

Función hash	Tamaño del hash (Bytes)	Probabilidad de colisión	Sobrecoste generado (Por cada millón) MB
MD5	16	1.84467×10^{19}	15.25879
SHA1	20	1.20893×10^{24}	19.07349
SHA256	32	3.40282×10^{38}	30.51758
SHA512	64	1.15792×10^{77}	61.03516

Tabla 4.1: Comparativa de las distintas funciones hash

En base a la tabla 4.1, puede verse que el sobrecoste por cada millón de documentos es completamente asumible en todos los casos, pues incluso con 10 millones de elementos, en el peor de los casos, es decir usando la función SHA512, los hashes tendrían un coste de tan solo 610MB, lo que no es algo exagerado teniendo en cuenta la cantidad de elementos que se tendrían y las ventajas que se obtienen.

Sin embargo, y dado que el hash se va a utilizar únicamente como identificador, es mucho más interesante usar una función que genere un hash de menor tamaño como podría ser MD5 o SHA1 ya que el tamaño es similar.

Finalmente se va a optar por SHA1 debido a que MD5 en la actualidad está roto, y aunque esto no debería afectar directamente al servicio, pues no se pretender obtener ningún tipo de seguridad, sí que se podría aprovechar esta vulnerabilidad por parte de algún ciberdelincuente añadiendo él mismo un mensaje falso que genere el mismo hash que un posible correo malicioso enviado por el mismo ciberdelincuente, evitando de esta manera que sea analizado en la plataforma.

A la hora de la elección de un lenguaje de programación, será necesario que este cuente con dicha función o en su defecto, que haya alguna implementación de esta ya desarrollada, pues no es objetivo de este trabajo desarrollarla.

4.1.4. Forma de mostrar los datos

La forma de mostrar los datos puede indicar o al menos sugerir de qué manera se deben almacenar para facilitar su posterior visualización.

Dependiendo del tipo de dato se puede mostrar una información u otra, aunque hay información común a todos los datos.

A continuación, se va a indicar la información común a todos los elementos y la asociada a cada uno de ellos.

Información común a todos los elementos

La información siguiente la tendrán todos los elementos visualizados en la plataforma, sean o no mensajes.

- Valor del elemento.
- Hash: Es el hash del valor del elemento.
- Fuente(s): Indica de dónde se ha obtenido el dato, puede ser la persona que publicó el mensaje o en caso de un enlace, la fuente podría ser un mensaje.
- Tipo: Será un array con distintos posibles valores donde los usuarios podrán votar el cuál de esos tipos es.

- Fecha de análisis: indica cuándo se analizó el mensaje.
- Score: indica la maliciosidad del elemento, pudiendo ser fiable o malicioso

También es importante indicar que en la visualización de cualquier dato extraído de un mensaje será importante que, además de la información relacionada con el mismo (Valor, fecha de análisis, score, …), se puedan listar todos los mensajes en donde ha aparecido dicho dato, así como un enlace a cada uno de ellos. El texto mostrado en este caso será el hash del mensaje.

El poder listar los mensajes donde aparece un dato concreto es importante para poder llevar a cabo investigaciones, o identificar nuevos mensajes perniciosos en base a elementos comunes con otros mensajes ya analizados.

Mensajes

La información siguiente es específica de los mensajes.

- Formato: texto plano, HTML o EML (En el futuro podría haber más como CSV, JSON, XML, …)
- Tipo: SPAM comercial, una estafa, una sextorsión o un ataque de phishing entre otros.
- Lenguaje: indica el lenguaje del mensaje (inglés, español, francés, …)

Direcciones IP

La información siguiente es específica de las direcciones IP.

- Versión: Indicara si las direcciones es IPv4 o IPv6.

Dominios

La información siguiente es específica de los dominios.

- Dominio: en caso de ser un subdominio, indicará el dominio al que pertenece.
- Subdominio: en caso de ser un dominio, se indicarán todos los subdominios analizados de dicho dominio.
- Tipo: Legítimo, phishing.

Enlaces

La información siguiente es específica de los enlaces.

- Domino: Se indicará el dominio o el subdominio al que pertenece.
- Tipo: Legítimo, phishing.

Direcciones de correo electrónico

La información siguiente es específica de las direcciones de correo electrónico.

- Domino: Se indicará el dominio o el subdominio al que pertenece.
- Tipo: Legítimo, phishing.

4.1.5. Cómo se va a buscar información y qué búsquedas se van a realizar

Saber qué búsquedas van a hacer y con qué datos se cuenta para llevarlas a cabo es necesario para determinar por un lado cómo se guarda la información y, por otro lado, las relaciones necesarias para que se puedan realizar las dichas consultas.

En este caso, dado un documento se debe poder obtener a cada uno de los datos extraídos de él y dado un dato cualquiera, tienen que poder obtenerse tanto los correos en los que aparece, como otros elementos relacionados con él. Por ejemplo, dado un domino, además de los mensajes donde está presente, también se debe obtener información sobre todos los enlaces analizados pertenecientes a dicho dominio.

Las búsquedas generalmente serán de elementos concretos, es decir, no será común realizar búsquedas por rango. Tampoco será común usar los operadores de mayor o menor.

Se debe poder buscar un elemento tanto por su valor, o por su hash en caso de ser un elemento de gran tamaño, así como por su identificador numérico, ya que su búsqueda puede ser mucho más rápida.

En un principio no será habitual buscar elementos por información relativa a ellos, por ejemplo, elementos analizados en una determinada fecha o con una determinada característica.

También es importante señalar que al principio las operaciones de inserción serán las más comunes y que a medida que se vaya haciendo uso del servicio, las operaciones de consulta serán las que prevalecerán. Además, es necesario que el servicio no sea demasiado lento, pues no sería práctico, por este motivo, debe prevalecer la velocidad de consulta sobre la inserción.

4.1.6. Cantidad de información que se va a almacenar

Respecto a la cantidad de información a almacenar, el sistema debe estar preparado para guardar un gran volumen de información, de decenas o cientos de millones de correos electrónicos, por lo que contar con un sistema escalable es crítico.

A la hora de realizar este informe, se cuentan con más de 8.5 millones de correos electrónicos perniciosos.

4.2. Información sobre los datos que se va a extraer

Como ya se ha comentado con anterioridad, uno de los objetivos del proyecto es extraer distintos tipos de datos y relacionarlos con otros correos electrónicos.

En concreto se quiere extraer:

- Direcciones IP
- Dominios
- Enlaces
- Direcciones de correo electrónico

Una forma sencilla de identificar y extraer este tipo de datos relativamente bien definidos es mediante expresiones regulares. El hacerlo de esta manera requiere que se sepa de manera muy precisa cómo están formados, qué símbolos tienen o no, qué tamaño,...

Por este motivo, se va a hacer un análisis exhaustivo de los distintos tipos de datos que se quieren obtener, especialmente en lo que a estructura, formato, tamaño y símbolos se refiere. Sin embargo, el análisis será eminentemente práctico, sin entrar detalles técnicos que no tengan relevancia a la hora de crear una expresión regular para evitar que la longitud de la memoria crezca en exceso.

De esta manera se podrán crear unas expresiones regulares muy precisas que generen el mínimo número de falsos negativos posibles.

4.2.1. Direcciones IP

Una dirección IP es un conjunto de números y/o letras que identifican de manera única a cada uno de los dispositivos de una red.

Existen dos versiones, la versión 4 y la versión 6, cada una tiene un formato distinto. [**?**]

IP versión 4 (IPv4)

Cuando se habla de IPv4 es importante mencionar que su formato no se ha especificado como tal en ningún RFC, quedando definido por el uso y por cómo fue descrito en otros RFCs. Con esto en mente se va a usar el RFC790 [**?**], donde se realiza la asignación de clases de redes, a distintos grupos de direcciones IP, para definir su formato.

Como se puede observar en dicho RFC, se podría decir que por convención una dirección IP en versión 4 está definida por un conjunto de cuatro números, separados entre sí por un punto y cuyo valor del 0 al 255. Pueden

ser escritos tanto con ceros a la izquierda o sin ellos, o lo que es lo mismo, tanto el 3 como el 03, como el 003 son válidos y tienen exactamente el mismo valor. Esto hace que una misma dirección pueda ser escrita de múltiples formas, por ejemplo 192.168.0.1 podría ser escrita como 192.168.000.001 ò 192.168.00.01 ò 192.168.000.01, etc.

IP versión 6 (IPv6)

La representación textual de una dirección IPv6 es muy flexible lo que hace que sea mucho más complicado crear una expresión regular para identificarlas. Su formato de definió en el RFC 4291, en la sección 2.2 [**?**], aunque debido a que ciertos operadores estaban teniendo problemas por su flexibilidad, la Internet Engineering Task Force (IEFT) publicó el RFC 5952 [**?**] con ciertas recomendaciones sobre su formato escrito para facilitar la implementación del protocolo.

En líneas generales una dirección IP en versión 6 está representada por 8 números hexadecimales de cuatro cifras, separados entre sí por dos puntos verticales (":") y escritos en normalmente en minúsculas, aunque también pueden estar escritos en mayúsculas.

A continuación, se van a comentar algunas de las posibles variaciones a la hora de representar textualmente una dirección de este tipo.

Omitir ceros a la izquierda Los 0 a la izquierda pueden (o no) ser omitidos y el valor de la dirección no se altera. Por ejemplo, las siguientes direcciones, pese a estar escritas de distinta forma, tienen el mismo valor.

2001:db8:aaaa:bbbb:cccc:dddd:eeee:0001
2001:db8:aaaa:bbbb:cccc:dddd:eeee:001
2001:db8:aaaa:bbbb:cccc:dddd:eeee:01
2001:db8:aaaa:bbbb:cccc:dddd:eeee:1

Contraer los ceros Si en una dirección uno o más números consecutivos son cero, se pueden eliminar. Esto puede hacerse una única vez.

Por ejemplo:
2001:db8:aaaa:bbbb:0:0:0:1
2001:db8:aaaa:bbbb::1

Precisamente esta característica es la que hace sea complicado analizar una dirección IP en su versión 6, ya que múltiples opciones son posibles y todas correctas.

Direcciones IPv4 embebidas Una dirección IPv6 puede tener embebida una dirección IPv4 al final de esta, que será escrita con el mismo formato de una dirección IPv4.

Por ejemplo
0:0:0:0:0:0:13.1.68.3

0:0:0:0:0:FFFF:129.144.52.38

Además, si se tiene en cuenta la propiedad anterior, podrían contraerse los ceros, quedado como sigue:

::13.1.68.3

::FFFF:129.144.52.38

4.2.2. Dominios

Un dominio es una cadena de caracteres que está asociada a una o varias direcciones IP. Está regulados por un organismo internacional llamado ICANN del inglés (Internet Corporation for Assigned Names and Numbers; ICANN) [**?**] y su especificación viene recogida en el RFC 1035 [**?**]

Formato

Según este RFC un dominio está formado por dos o más partes separadas entre sí por un punto «.».

Siendo el formato como sigue:

```
[<Subdominio>.]<nombre de dominio>.[<SLD>.]<TLD>
```

Donde:

- Subdominio: Los subdominios subpartes del dominio al que preceden. Puede haber tantos como se quiera.
- Nombre de dominio: Se registra en el ICANN o la autoridad competente dependiendo del top-level domain (TLD) o del second-level domain (SLD).
- SLD: Son conocidos como dominios de segundo nivel y permiten una especificación de un dominio de primer nivel. Por ejemplo, «co.es» podría estar destinado a empresas comerciales españolas. Todos los SLDs registrados pueden ser consultados en Publicsuffix [**?**].
- TLD: Son conocidos como dominios de primer nivel y son los gestionados por el ICANN. Todos los TLDs registrados se pueden encontrar en la web del Internet Assigned Numbers Authority (IANA) [**?**]

Tener en cuenta los TLDs y los SLDs es muy importante para asignar correctamente los dominios y los subdominios, ya que no tener en cuenta los SLDs podría hacer que se considerasen nombres de dominio que no lo son.

Además, es una buena forma de detectar falsos positivos cuando se están buscando dominios en un documento de texto.

Tamaño

En la sección "2.3.4. Size limits" del RFC 1035, se indica que el tamaño máximo para el dominio son 255 completo caracteres y el de casa sección de 63 caracteres, aunque en la actualidad no suelen ser tan largos.

Caracteres permitidos

Así mismo en la sección "2.3.3. Character Case" se indica que la codificación de un dominio debe ser ASCII del inglés (American Standard Code for Information Interchange) e insensible a mayúsculas y minúsculas, aunque no obliga a ello. En el RFC 3490 [**?**], se recoge la posibilidad de usar caracteres Unicode aunque estos sean representados internamente como caracteres ASCII, en la práctica, los caracteres Unicode no son muy usados.

Además, y según se especifica en el RFC 952 [**?**] y en el RFC 1123 [**?**], los únicos caracteres válidos dentro de un dominio son los caracteres alfanuméricos (A-Z) y (0-9), el guion medio (-) y el punto (.), aunque algunos servidores también permiten el guion bajo (_).

4.2.3. Enlaces (URLs)

Los enlaces o URLs (del inglés Uniform Resource Locator) vienen definidas por el RFC 1738 [**?**], de especial interés la sección "3. Specific Schemes". Una URL es a su vez un tipo de URI (del inglés Uniform Resource Identifier) definido en el RFC 3986 [**?**], de este último el apartado más interesante para el proyecto es la sección "3.3. Path" en el cual se especifica el formato que debe tener y también en el "Apéndice A: Collected ABNF for URI".

También es interesante el RFC 3696 en la sección "4.1. URI syntax definitions and issues" [**?**], ya que se centra en las URLs cuyo esquema sea Hypertext Transfer Protocol (HTTP) o HTTP seguro (HTTPS), y por tanto el más interesante para el proyecto .

Formato

El formato viene definido como sigue:

```
{http/https}://<dominio>[:<puerto>][/<ruta[<parámetros>]>]
```

Donde:

- http/https: Define el protocolo y aparecerá uno u otro.
- Dominio: su formato ya ha sido definido.
- Puerto: No es muy usado, es un valor numérico que va del 0 al 65535.
- Ruta: son distintas cadenas de texto que dan acceso a cada una de las páginas de un mismo dominio.

- Parámetros: Son posibles valores que envían información extra al servidor.

Caracteres permitidos

Los caracteres permitidos en una URL son:

- Cualquier carácter alfanumérico.
- Cualquiera de los símbolos siguientes: - . _ ~ ! $ & ' ( ) [ ] * + , ; = : @ # ?
- El carácter % precedido de un número hexadecimal de dos cifras.

Tamaño máximo

En principio una URL no tiene límite de tamaño tal y como se recoge en el RFC 7230 al final de la sección "3.1.1. Request Line" [**?**], sin embargo y como también se menciona en el mismo RFC, sí hay ciertas restricciones por parte de los navegadores. Por ejemplo, la longitud máxima de una URL si se usa Internet Explorer es de 2083 caracteres en total [**?**].

4.2.4. Dirección de correo electrónico

La sintaxis de una dirección de correo electrónico viene definida en el rfc 5322 en la sección "3.4.1. AddrSpec Specification" [**?**], sin embargo, en la definición de este RFC se centra mucho en la parte del dominio, que ya ha sido definida, por lo que es más útil el RFC 3696, concretamente su sección "3. Restrictions on email addresses" [**?**]

Formato

El formato de una dirección de correo electrónico es de la siguiente forma:

```
<Parte local>@<dominio>
```

Siendo:

- Parte local: es la parte de la dirección que define al usuario.
- Dominio: indica el dominio al que pertenece la dirección.

Caracteres permitidos

Los caracteres permitidos en el dominio ya han sido definidos.

Los caracteres permitidos en la parte local pueden ser, cualquier carácter ASCII siempre que vaya entre comillas dobles o escapeado mediante el

símbolo "\". No es necesario ni escapar, ni entrecomillar los caracteres alfanuméricos, ni los siguientes caracteres especiales entre paréntesis (! # $ % & ' * + - / = ? ^ _ ` . ~{ } |).

El punto (".), pese a estar permitido, no puede estar ni en la primera, ni en la última posición, tampoco puede haber más de dos seguidos.

Tamaño máximo

El tamaño máximo de la parte local es de 64 caracteres, por lo que sumados a los 255 de la parte del dominio y el "@", da un máximo de 320 caracteres.

4.3. Información sobre los tipos de archivo que se van a analizar

Como ya se ha comentado en este proyecto se van a tratar tres tipos de archivos. Los primeros de ellos y los más simples son los que están escritos en texto plano, los segundos y tal vez los más usuales son los escritos en formato HTML y finalmente y los más complejos son los archivos con formato EML.

Durante esta sección es importante diferenciar entre codificación de caracteres (ASCII, UTF-8, Unicode) y formato de archivo (Texto plano, HTML o EML).

4.3.1. Archivos HTML

El formato HTML está definido por el W3C y cuya última versión es la 4.01 del 27 de marzo del 2018 y lo más interesante es lo referente a la sección "12.2 (The A element)" [**?**] ya que en ella se especifica cómo se definen los enlaces en el formato HTML.

Formato

Normalmente un enlace en formato HTML se define de la siguiente forma

```
<a href:"{URI}">{texto}</a>
```

Donde:

- URI: Será una URI tal y como ya ha sido definida, aunque generalmente será una URL. De todos los posibles tipos, tienen especial relevancia las URLs con esquema HTTP, HTTPS y MAILTO.
- Texto: Será el texto que se muestre al usuario.

4.3.2. Archivos EML

El formato EML es el más usado para guardar correos electrónicos. Aunque su especificación no está en ningún RFC concreto, en general cumple con lo definido en el RFC 5322 [**?**], del cual son especialmente interesantes los puntos 2 “Lexical Analysis of Messages” y 3 “Syntax”.

También es interesante el RFC 4021 [**?**] donde se detallan todos los tipos de cabeceras oficiales, su formato, y una pequeña descripción de estas. En el RFC 6854 [**?**], en concreto el punto 2 “Allowing Group Syntax in ‘From:’ and ‘Sender:’ ” se actualiza la sintaxis de los campos “From” y “Sender”, campos muy relevantes en este proyecto ya que pueden dar información sobre el emisor del correo.

Formato

En líneas generales y sin entrar en detalles concretos un archivo EML está formado por un conjunto de cabeceras y opcionalmente un cuerpo.

Cada cabecera está formada por dos campos, por el nombre y por el cuerpo, ambos están formados por una cadena de caracteres ASCII y separados por “:”.

El cuerpo está separado de las cabeceras mediante una línea vacía y está también compuesto por una secuencia de caracteres codificada ASCII.

Es importante señalar que un único archivo EML puede contener en el interior textos con formato HTML o en texto plano.

Un ejemplo sería (Para ver el documento mejor, ver apédice A.1):

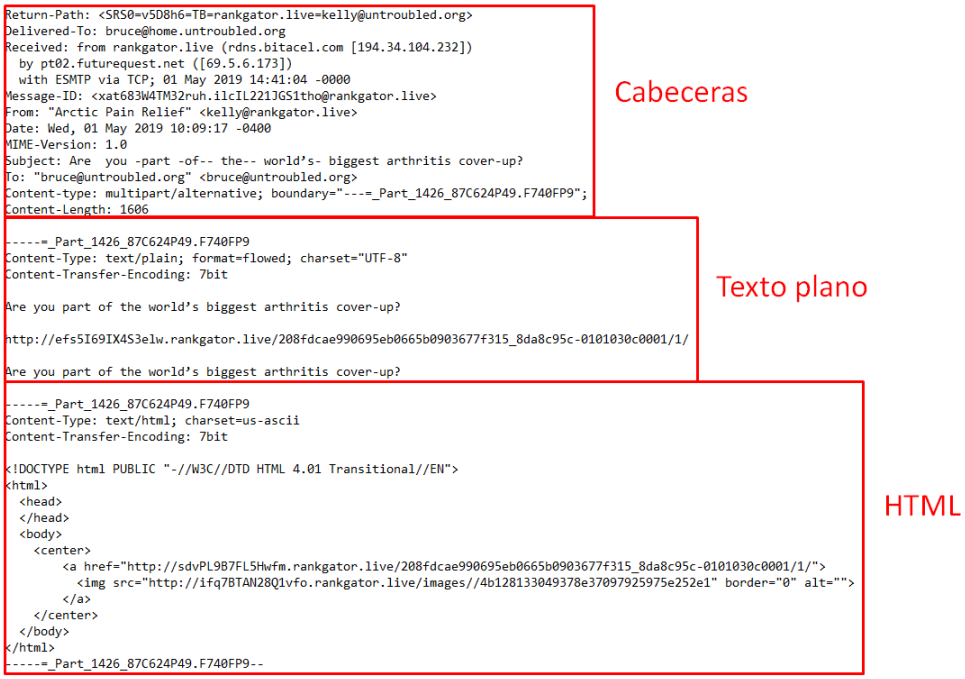


Figura 4.1: Ejemplo de EML

4.4. Patrones maliciosos comúnmente usados en el correo electrónico

Hasta el momento únicamente se ha analizado qué tipo de información se va a analizar, qué formato tiene y qué limitaciones de codificación y/o tamaño tienen.

Entender esto con claridad permite entender algunas de las técnicas usadas por los ciberdelincuentes para poder atacar a sus víctimas.

A continuación, se van a exponer una serie de patrones, que, si bien no permiten saber con seguridad de que se trate de un ataque, sí que permiten al menos tener cierta sospecha de que el correo sea malicioso.

Para el desarrollo de esta sección se han consultado entre otros los siguientes artículos

- A Framework for Detection and Measurement of Phishing Attacks [**?**]
- PhishDef: URL Names Say It All [**?**]
- Learning to detect phishing emails [**?**]

4.4.1. Patrones maliciosos en Dominios

Usar una gran cantidad de subdominios

Un patrón bien conocido es utilizar una gran cantidad de subdominios con el objetivo de camuflar en ellos el dominio principal.

En este tipo de ataques, el dominio a suplantar suele estar escrito al principio, seguido de más subdominios, hasta llegar el domino principal. De esta manera se dificulta su identificación.

Un ejemplo de este tipo podría ser:

```
http://www.ugr.es.custsupportref1007.dllconf.info/r1/vm/
```

Como se puede observar, la URL anterior se hace pasar por el dominio "www.ugr.es", sin embargo, su dominio principal es "dllconf.info".

Para detectar este tipo de ataque se podrían identificar el número de puntos "." que tiene un dominio, de modo que cuantos más tenga, mayor es la probabilidad de que sea malicioso.

4.4.2. Patrones maliciosos en Enlaces

Usar como dominio una IP

Este patrón consiste en despistar al usuario usando como dominio una dirección IP y escribiendo en la ruta el dominio a suplantar. De esta forma si el usuario lee la dirección, el único domino que verá como tal es el suplantado y por tanto le dará legitimidad a la página.

Un ejemplo de este patrón es:

```
http://211.233.39.145/www.ugr.es/login.html
```

Observando la dirección anterior puede verse como el único dominio como tal que aparece es el de "www.ugr.es", aunque realmente el host es la IP "211.233.39.145".

Detectar este tipo de patrón podría consistir en analizar si la máquina un dominio o una dirección IP. En caso de ser una IP, se podría decir que es posiblemente maliciosa.

Embeber una URL en la ruta de otra URL

Una variación del patrón anterior es, en vez de usar una dirección IP, usar un dominio, generalmente muy corto y añadir la dirección a suplantar en la ruta, al igual que en el caso anterior.

Un ejemplo de este caso sería:

```
http://fkDom.tl/www.ugr.es/login.html
```

Al hacer esto, un usuario que no tenga cuidado no se percatará de que el dominio realmente es “fkDom.tl” y no “www.ugr.es” como el hubiera esperado.

Para detectar este tipo de ataque, se podría comprobar que, en la ruta de un enlace, no haya ninguna otra ruta y en caso e haberla, tanto el dominio como la dirección completa serían sospechosos de ser perniciosos.

4.4.3. Patrones en archivos HTML

Ofuscación de la URL original una etiqueta tipo a

Uno de los métodos más comunes y simples consiste en hacer uso de la etiqueta `<a>` de HTML mostrando al usuario un valor distinto al recurso enlazado realmente mediante el atributo “href” de dicha etiqueta.

Este patrón puede darse con varios de los posibles recursos que pueden ser enlazados mediante esta etiqueta. La cantidad de recursos depende en gran medida del soporte que tenga navegador para cada uno de los distintos esquemas asociados a cada recurso.

Todos los esquemas disponibles y a qué tipo de recurso que enlazan, está definidos por el IANA [**?**]

De entre todos ellos destacan los siguientes, por tener soporte en todos los navegadores y ser los más utilizados.

HTTP/HTTPS Es el caso más habitual, en este caso en el campo de texto se muestra un enlace a una página que no se corresponde con la enlazada.

Un ejemplo sería `<a href="https://www.ujr.es/">www.ugr.es</a>`

Este método es especialmente útil cuando se trata de ataques dirigidos a usuarios de dispositivos móviles ya que, en ellos, o no se ve la dirección en la que están navegado o sólo se ve parcialmente, lo cual dificulta en gran medida percatarse de este ataque.

MAILTO Permite enviar abrir la aplicación de correo por defecto y añadir directamente al destinatario.En este caso se podría mostrar al usuario un email distinto al que se envía el mensaje.

Por ejemplo: `<a href="mailto:support@ujr.es"> support@ugr.es</a>`

Solución Una posible forma de detectar esta amenaza es, en primer lugar, comprobar si el texto mostrado al usuario es un enlace y en caso de serlo, compararlo con el enlace escrito en el atributo href de la etiqueta `<a>`. En caso de ser distinto, el enlace del atributo href sería podría ser un enlace malicioso.

Uso de JavaScript

Cuando un correo electrónico se abre en el navegador y está en formato HTML, puede contener código JavaScript que se ejecute automáticamente al abrirse. Si bien esto es hago normal en la web, no suele serlo en un mensaje de correo, ya que su contenido suele ser estático.

Este tipo de código siempre va asociado a una etiqueta "`<script> </script>`", por tanto, una forma sencilla de detectar este patrón es simplemente comprobar si esta etiqueta aparece o no en el mensaje.

4.5. Integraciones de terceros

A lo largo de esta sección se va a hablar de algunos servicios de terceros que pueden complementar los análisis de la plataforma.

Algunos de ellos están relacionados directamente con la ciberseguridad y ya han sido descritos anteriormente, otros son de propósito general, pero pueden aportar información interesante a la hora de analizar un correo sospechoso.

Para cada servicio se especificará en qué casos será relevante, cómo podría usarse y qué información adicional podría aportar.

4.5.1. Virus Total

Permite analizar archivos, enlaces, dominios y direcciones IP por varias decenas de servicios especializados dando así una visión más completa.

Podría utilizarse para analizar alguno de los tipos de datos anteriores, que salgan como sospechosos o de los que el usuario dude.

4.5.2. Metadefender

Es muy similar a Virus Total y por tanto de podría utilizar como alternativa a este.

4.5.3. Have I been pwned

Permite saber si un correo ha estado involucrado en una brecha de seguridad.

Podría utilizarse para saber si por ejemplo la dirección de la que proviene el correo ha estado expuesta en alguna brecha de seguridad, pudiendo sospechar de ella en caso de haber estado involucrada.

4.5.4. Whois

Da información sobre un dominio o una dirección IP, como cuándo se registró o a quién pertenece.

Se puede utilizar para obtener más información sobre dominios sospechosos. También se puede utilizar para intentar descubrir a un ciberdelincuente.

4.6. Elección de la base de datos

Debido a la gran cantidad de información que se va a manejar y al uso que se le va a dar a la base de datos, es el punto más importante que analizar, ya que una vez que se tenga una gran cantidad de datos almacenada, cambiar de sistema puede ser realmente complicado o simplemente imposible.

La elección de un sistema de gestión de información que permita tanto insertar como hacer búsquedas rápidas en algo fundamental y de lo que va a depender el éxito o el fracaso del proyecto.

Tradicionalmente la información se guardaba en bases de datos relacionales interactuando con ellas mediante SQL. Esto ha hecho que en la actualidad haya multitud de gestores de bases de datos relacionales como MySQL [**?**], MariaDB [**?**], SQLite [**?**], PostgreSQL [**?**] u Oracle [**?**] entre otros. Todos comparten el mismo lenguaje (SQL) para poder interactuar con ellos, aunque con pequeñas diferencias significativas, lo que hace que con algunos cambios se pueda cambiar de un sistema de gestión a otro y esto supone una gran ventaja.

Por otro lado, y aunque con menos uso, también existen muchos otros tipos de gestores de base de datos, que o bien no usan en absoluto el lenguaje SQL, como bien podría ser el caso de Mongo DB [**?**], o bien, además de usar SQL tienen ciertas peculiaridades que hacen que no sean únicamente una base de datos relacional.

A diferencia de los sistemas relacionales, migrar de uno a otro, o incluso a uno relacional, suele suponer un gran cambio, no sólo a nivel de código, también a nivel conceptual y por tanto de diseño, por lo que elegir un sistema NoSQL, debe ser una elección meditada y analizada en detalle, ya que todos estos sistemas de almacenamiento suelen estar diseñados para tareas muy concretas y bien definidas, lo que hace que tengan ventajas e inconvenientes que hay que tener muy en cuenta.

Independientemente del sistema que se elija, se deben tener en cuenta los siguientes puntos:

- Soportado o no por Azure
- Velocidad de las búsquedas
- Conocimientos previos
- Documentación
- Librerías disponibles
- Facilidad para hacer relaciones entre datos

- Escalabilidad

4.6.1. Sistemas relacionales SQL

Son los sistemas más utilizados en la actualidad. Todos utilizan un lenguaje común para interactuar con ellos llamado lenguaje de consulta estructurada (Del inglés Structured Query Language, o SQL), aunque puede haber pequeñas diferencias entre ellos.

Permite el uso del álgebra y el cálculo relacional para interactuar con el gestor de la base de datos, permitiendo de esta forma, buscar, modificar, añadir y eliminar información.

Son usadas en multitud de proyectos de distinta envergadura por lo que su eficacia está muy probada, además de tener una amplia comunidad, una gran documentación y multitud de librerías para los distintos lenguajes de programación, algunas de ellas integradas dentro del propio lenguaje.

Ventajas

- Facilidad para cambiar de un gestor a otro (MariaDB, MySQL, SqLite, Oragle, Postgre SQL)
- Mucha documentación
- Una gran comunidad
- Conocimientos previos
- Una gran cantidad de librerías para multitud de lenguajes de programación
- Hacer relaciones entre es sencillo
- Búsquedas relativamente rápidas
- Respeta ACID

Desventajas

- Inserciones lentas
- Escalado más complicado
- Esquema mucho más estricto
- Programación más lenta [1]

[1] El tener conocimientos previos unido a la gran comunidad, pueden mitigar este problema.

Conclusiones

Usar un sistema de gestión de contenido relacional aporta una gran cantidad de ventajas, la mayor de ellas puede ser la velocidad de sus búsquedas, así como la facilidad para crear relaciones entre elementos. También es muy interesante que se pueda cambiar de un sistema a otro con relativa facilidad, en caso de que en el futuro uno de ellos cuente con ventajas adicionales sobre los demás.

A todo esto, hay que sumarle que al ser un sistema con una larga trayectoria cuenta con una gran documentación, una comunidad amplia, multitud de librerías para la mayoría de los lenguajes de programación modernos y además se ha estudiado durante la carrera. Los dos mayores problemas son por un lado la escalabilid ad, que, aunque posible, es complicada. Esto puede suplirse al usar servicios en la nube que escalan de manera transparente los recursos de la base de datos para mantener las prestaciones, de ahí la importancia de usar un sistema soportado por Azure.

El segundo problema es la lentitud de las inserciones, pero dado que los correos únicamente se van a insertar una única vez y que van a prevalecer las búsquedas sobre las inserciones, hace que, aunque este problema se deba tener en cuenta, no sea crítico siempre que el tiempo de inserción no se vuelva dramáticamente lento, en cuyo caso habría que escalar el sistema con la dificultad y el coste que esto supondría.

4.6.2. Mongo DB

Es un sistema gestor de bases de datos orientado a documentos, en vez de a tablas, y por tanto NoSQL. Concretamente este sistema de almacenamiento utiliza documentos en formato JSON, que se almacenan en formato BSON del término JSON binario y como lenguaje de consulta utiliza JavaScript.

Al no ser un sistema estructurado, los documentos pueden ser distintos y no tener una estructura fija definida.

Su escalado horizontal es muy sencillo lo que asegura una gran disponibilidad y que no sea costoso.

Ventajas

- Escalado horizontal y vertical: En Mongo escalar tanto horizontal como verticalmente es algo simple que no requiere prácticamente de configuración.
- Alta disponibilidad: que el escalado horizontal sea simple permite a Mongo tener una disponibilidad muy alta, ya que esta estructura permite una mejor respuesta a fallos eventuales del sistema.
- Es muy rápido en las inserciones y en las actualizaciones de datos.

- Tiene muchas más opciones para tratar datos de manera nativa.

Desventajas

- Las búsquedas son muy lentas.
- No hay posibilidad nativa de hacer uniones entre colecciones.
- No respeta el modelo ACID.

Conclusiones

Que Mongo esté orientado a documentos hace que sea un gestor ideal para esta aplicación ya que lo que se va a guardar mayoritariamente son mensajes, es decir, documentos.

Además, su tiene una gran velocidad de inserción, un escalado horizontal prácticamente nativo, y que los documentos no tengan una estructura fija por lo que se pueden personalizar de modo que se minimicen las búsquedas, hacen de Mongo una opción a tener muy en cuenta.

También posee una gran cantidad de librerías para multitud de lenguajes de programación, una documentación muy completa y aunque no cuenta con la misma comunidad que los sistemas estructurados, sí que es bastante amplia por lo que no es complicado encontrar ayuda por parte de otros usuarios.

Sin embargo, Mongo también presenta problemas importantes como sus búsquedas lentas, el hecho de tener que mantener la integridad de los datos de manera manual o la imposibilidad de hacer uniones entre datos de manera sencilla y rápida. Y es que, como se ha comentado anteriormente las búsquedas prevalecen sobre las inserciones, además hay una gran cantidad de relaciones que serían complicadas de mantener. Otro punto débil es que, la posibilidad de migrar a otro sistema supondría el tener que comenzar desde cero, puesto que no hay ningún sistema compatible o similar.

4.6.3. Apache Cassandra

Es un sistema de almacenamiento de datos que destaca por su escalado horizontal al ser una base de datos distribuida de escalado lineal, lo que es muy interesante ya permite predecir el número de nodos que se van a necesitar en función del número de peticiones que se tengan y las peticiones que es capaz de atender un único nodo. Esto quiere decir, que, si con un único nodo se pueden atender 1000 consultas por segundo, con dos se podrán atender 2000 y así sucesivamente [**?**].

Su modelo de datos está basado en filas de columnas de clave-valor.

4.6.4. Ventajas

- Escalado horizontal y lineal.
- Velocidad de acceso.

Desventajas

- Las relaciones entre elementos son complicadas.
- Es un cambio de paradigma complicado.
- La documentación no está acabada [**?**].
- No se tienen conocimientos previos.

Conclusión

Lo más interesante de Cassandra es sin duda su escalado horizontal de manera lineal, sin embargo, cuenta con numerosos problemas como que la documentación no está totalmente acabada, que el modelo de estructura de datos no encaja con los modelos de datos de este proyecto, su comunidad de usuarios tampoco es muy amplia y tampoco se tienen conocimientos anteriores, por lo que la curva de aprendizaje puede suponer un gran problema.

4.6.5. Elasticsearch

Elasticsearch [**?**] no es tanto una base de datos como tal, es un motor de búsqueda y dado que lo que se van a almacenar son correos electrónicos, y por tanto, texto, puede ser muy buena opción, tal vez no como sistema de almacenamiento principal, pero sí como sistema complementario con el que realizar búsquedas avanzadas dentro de los propios mensajes.

Ventajas

- Escalado horizontal.
- Distribuido.
- Velocidad de búsqueda (incluso con búsquedas complejas).
- Se pueden aplicar algoritmos sobre las búsquedas.
- Se pueden usar analizadores de texto.
- Preparado para la inteligencia artificial [**?**].
- La documentación está muy bien detallada [**?**].

Desventajas

- Velocidad de indexación.
- Su función principal es buscar texto, no almacenar información.

Conclusión

Aunque ElasticSearch es una herramienta muy atractiva y puede resultar muy útil en lo referente a búsquedas avanzadas de mensajes, no es viable usarla como sistema principal de almacenamiento ya que no es para lo que está pensada.

De todas formas, podría ser muy interesante integrar esta herramienta en el servicio ya que ofrece funcionalidades muy interesantes y abre la posibilidad al uso de algoritmos de inteligencia artificial, lo que resulta especialmente interesante en mensajes de estafas, o de extorsión, en los que analizar detalles como enlaces, o dominios no es efectivo.

4.6.6. Conclusión

Después de analizar múltiples motores de bases de datos, los que más se ajustan a las necesidades de este proyecto son o un sistema SQL o Mongo, sin estar realmente claro cuál puede funcionar mejor.

Sin embargo, teniendo en cuenta que se van a realizar una gran cantidad de consultas y que éstas en Mongo son lentas, y que con la gran cantidad de relaciones que hay un sistema relacional ofrece un gran soporte, finalmente se va a optar por diseñar un sistema relacional.

Además, y aunque en un principio el desarrollo puede ser más lento, al tener experiencia previa en este tipo de sistemas, contar con una gran cantidad de documentación y una comunidad amplia en caso de tener algún problema eventual hará que el desarrollo no sea tan lento como lo pudiera ser si se tuvieran conocimientos previos de Mongo.

4.7. Diseño de la base de datos para un sistema relacional

A la hora de realizar el diseño de la base de datos, se va a optar por hacer un diagrama entidad relación y además se va a explicar cada una de las tablas, aunque hay tablas intermedias o con una finalidad común que se describirán de manera genérica para facilitar la comprensión del documento.

Aunque un esquema de este tipo está pensado para un sistema relacional, entender la estructura de este para pasarlo a otro tipo de diagrama y así adaptado a Mongo, la otra opción que se ha aproximado más a la solución no debería ser excesivamente complejo, más allá de adaptar las tablas a las

colecciones de Mongo, integrar parte de las tablas intermedias en una única estructura y crear ciertos índices que faciliten las búsquedas.

En cambio, en lo que respecta al contenido, habría que reanalizar todo pues no habría ninguna forma sencilla de pasar el contenido de un sistema a otro.

4.7.1. Tablas

Antes de analizar cada una de las tablas, se va a precisar algunos campos comunes entre tablas, así como algunas tablas comunes a algunos elementos.

Elementos comunes a varias tablas.

- `Id`: Es la clave primaria de la tabla. Siempre es un entero. Se ha elegido esta opción por rendimiento.
- `Ocurrences`: Es el número de veces que este elemento ha llegado para analizarse.
- `Score`: Es el valor de maliciosidad dado al elemento. Un número negativo indicará que es malicioso, un número positivo que es legítimo y cero que no se tiene información sobre dicho elemento.
- `Hash`: Es un hash del elemento. Esto evita analizar dos veces un mismo elemento ahorrando tiempo de computación.
- `Value`: Es el valor del propio elemento en sí.
- `Type`: Es una clave externa a un id de la tabla `<Nombre Elemento>Type`, que identifica el tipo de elemento que es.

  En algunos elementos se indican posibles tipos, esto se hace para facilitar la comprensión del documento, pero realmente se almacenan en la tabla `<Nombre Elemento>Type`.

Tablas genéricas

- `<Nombre Elemento>Delete`: Estas tablas están asociadas a cada uno de los elementos a almacenar susceptibles de ser borrados, como mensajes, emails,…

  Esto permite cumplir con el derecho al olvido de la ley de protección de datos europea.

  - `Id`
  - `Id<Nombre Elemento>`: es una clave externa al elemento en cuestión.
  - `Hash`: Este hash permite identificar a cada elemento insertado, de modo que se le da al usuario la posibilidad de borrarlo si tiene el hash. Está formado por el valor del elemento, una marca de

tiempo y un valor aleatorio. Si dos usuarios introducen el mismo elemento, y sólo uno de ellos quiere borrarlo, el elemento permanecerá hasta que todos los usuarios que lo han introducido quieran borrarlo.

- `<Nombre Elemento>Souces`: Identifica todas las fuentes de las que proviene el mensaje, pudiendo darle más o menos valor a la información que contenga en función de la fuente de la que provenga. Ej: emails identificados como maliciosos a mano tendrán más valor que los identificados por la IA (O al revés)
  - `Id`
  - `SourceName`: Nombre de la fuente de la que proviene.
  - `Score`
- `<Nombre Elemento>Souces`: Identifica todas las fuentes de las que proviene el mensaje, pudiendo darle más o menos valor a la información que contenga en función de la fuente de la que provenga. Ej: emails identificados como maliciosos a mano tendrán más valor que los identificados por la IA (O al revés)
  - `Id`
  - `Value`: Nombre de la fuente de la que proviene.
  - `Score`
- `<Nombre Elemento>Type`: Identifica los posibles tipos del elemento, por ejemplo, si el elemento en cuestión es una criptomoneda, identifica el tipo de criptomoneda que es.
  - `Id`
  - `Value`: Es el nombre del tipo.
- `<Nombre Elemento>MaliciousChecks`: Identifica cada uno de los posibles patrones maliciosos de dicho elemento, como por ejemplo en el caso de una URL, no coincida el valor del atributo “href” con lo mostrado al usuario.
  - `Id`
  - `Value`: es el nombre de patrón.
  - `Score`
- `<Nombre Elemento1>_<Nombre Elemento2>`: Esto son tablas intermedias para dejar la base de datos en Forma normal de Boyce-Codd cuando la multiplicidad es de muchos a muchos.
  - `Id`

- `Id<Nombre Elemento1>`: es una clave externa a un id de la tabla `<Nombre Elemento1>`
- `Id<Nombre Elemento2>`: es una clave externa a un id de la tabla `<Nombre Elemento2>`
- `Date` *(Opcional)*: Permite registrar el mismo elemento, mediante la misma fuente, más de una vez. Esto puede permitir detectar elementos muy analizados en un instante de tiempo, típico por ejemplo cuando se realiza una nueva campaña de phishing, dicho elemento tendrá un pico de análisis cuando se realice la campaña.

Tablas específicas de cada elemento

Son cada uno de los elementos que se van a guardar en la base de datos. Se pueden distinguir tres tipos de datos:

1. Los mensajes en sí. Es la tabla principal del proyecto y con la que están relacionada la mayoría de las tablas principales.
2. Los elementos que se detectan de cada mensaje. Por ejemplo: direcciones de emails, dominios, links, …
3. Los distintos tipos de mensajes que hay, por ejemplo, mensajes de correo electrónico (Los elementos principales), mensajes de Whatsapp o Telegram, SMS, mensajes de redes sociales como Instagram, Facebook o Twitter, … Estas tablas siempre tendrán una clave externa a la tabla Message, lo que permite identificar un mismo texto mediante distintos tipos de mensaje. Por ejemplo una campaña de phishing
4. Se podrían distinguir un cuanto tipo de elementos que son todas las tablas auxiliares específicas de elementos concretos y no detalladas anteriormente. Este tipo de tablas son, por ejemplo, la tabla de países de los números de teléfono, la de las cabeceras de los emails, … Son tablas que añaden detalles o facilitan la implementación.

Mensajes

- `Message`: es la tabla principal del proyecto. En esta tabla se guardarán los mensajes o la ubicación de estos y está formada por los siguientes campos:
  - `Id`
  - `Hash`
  - `Value`
  - `Occurrences`

- `Score`
- `Kind`: [eml, html, text]
- `AnalyzedBy`: en caso de tener varios lenguajes o varias formas de análisis, permite saber con qué lenguaje se ha hecho
- `Status`: indica si está analizado o pendiente de analizar.
- `AnalyzedAt`: indica la fecha del último análisis. (Esto permite que si en el futuro se añade nuevos análisis, sólo se analicen los archivos anteriores a la fecha de implementación)

Elementos a detectar en los mensajes

- `IpAddresss`: Son direcciones ip que pueden ser maliciosas o no, en función del score.:
  - `Id`
  - `Hash`
  - `Value`
  - `Occurrences`
  - `Score`
  - `ipV4`: indica si la ip es v4 o v6.
- `Domain`: en esta tabla se almacenan tanto dominios como subdominios. Los subdominios tendrán una clave externa al dominio.

  Puede haber dominios en los que todos sus subdominios sean maliciosos o en los que no tengan relación. (Véase el caso de los subdominios de wordpress.com o blogspot.com)

  Esta tabla está relacionada con la tabla ipAddress, anteriormente detallada permitiendo tanto a un dominio tener varias direcciones IP y como tener varios dominios apuntando a la misma IP, y estas ser tanto IPv4 como IPv6

  - `Id`
  - `Hash`
  - `Value`
  - `Occurrences`
  - `Score`
  - `DomainId`: permite identificar el domino en caso de ser un subdominio.

- `URL`: Son enlaces, es decir una ruta de un dominio y van a estar siempre asociadas a él. Este es un elemento clave a analizar pues la inmensa mayoría de los ataques de phishing se producen al ingresar en una url falsa.

  - `Id`
  - `Hash`
  - `Value`
  - `Occurrences`
  - `Score`
  - `Domain`: Es un clave externa a la tabla domains como ya se ha detallado con anterioridad.

- `EmailAddresses`: En esta tabla se almacenarán todas las direcciones de correo electrónico que se detecten.

  Esta tabla también tiene especial relevancia porque la mayoría de los ataques se realizan por correo electrónico. Esta tabla se relaciona con la tabla domain, anteriormente detallada. Además, también se tiene en cuenta el enmascaramiento de direcciones falsas mediante el tag "`<a>`" y el atributo "href" de html.

  - `Id`
  - `Hash`
  - `Value`
  - `Occurrences`
  - `Score`
  - `Domain`: Es un clave externa a la tabla domains como ya se ha detallado con anterioridad.

*La implementación permite añadir nuevos elementos a detectar, tales como cuentas bancarias, o principales palabras.

4.8. Elección del lenguaje de programación

El lenguaje de programación es el punto de unión entre la base de datos y la interacción del usuario. Se distinguen dos partes, por un lado, la parte de análisis del mensaje, extracción de datos e inserción de estos en la base de datos y, por otro lado, la visualización.

Lo ideal sería utilizar el mismo mensaje para ambos propósitos ya que daría homogeneidad al proyecto, aunque esto supone que la búsqueda de un lenguaje que satisfaga todos los requisitos sea más compleja.

Independiente a esto, se va a dar prioridad a lenguajes soportados por el servicio de Azure Functions ya que permite escalar el servicio de manera transparente y el costo va ligado a la utilización del servicio evitando tener costes iniciales o fijos.

De esta manera, los lenguajes candidatos serían C#, F#, Node.js, Python, Java o PHP [2].

Dentro de los lenguajes anteriores, se van a valorar:

Java: por el conocimiento previo, por su portabilidad, por tu soporte con multitud de tipos de bases de datos y por la cantidad de librerías existentes para este lenguaje.

Python: por ser uno de los lenguajes de programación más usados en Big Data y en inteligencia artificial, por la sencillez del lenguaje, y por tener soporte para web.

Node.js: por su uso en la web, por tu portabilidad y por su soporte completo de expresiones regulares.

PHP: por ser run estándar en la web, por su soporte completo a expresiones regulares, por su fácil integración con múltiples sistemas de bases de datos.

Se van a valorar los siguientes aspectos:

1. Librerías para bases de base de datos SQL[3]
2. Soporte y rendimiento con expresiones regulares.
3. Librerías para analizar archivos EML.
4. Librerías para archivos HTML.
5. Funciones hash, en concreto SHA1.
6. Soportado o no por Azure.
7. Facilidad para crear una web mediante dicho lenguaje.
8. Documentación.
9. Comunidad.

4.8.1. Java

Java es un lenguaje multidispositivo, con una gran comunidad, una documentación muy completa y tiene una gran cantidad de librerías para todo tipo de bases de datos.

[2]PHP ya no está disponible, pero al comienzo de este trabajo sí lo estaba.

[3]Adicionalmente y por si finalmente se migra a Mongo, se comprobará si existe o no algún tipo de soporte para este tipo de base de datos.

Soporte para bases de datos

Java cuenta con una librería nativa para conectarse a bases de datos relacionales conocido como Java Database Connectivity [JDBC] [?] que cuenta con controladores para multitud de bases de datos relacionales como Postgresql [?], MySQL [?] o MariaDB [?]

También cabe mencionar que hay ORM's que facilitan la tarea de acceder a la base de datos, el más famoso puede ser Hibernate [?], además, este mismo ORM ofrece soporte para Mongo.

Soporte para expresiones regulares

Java tiene la clase Pattern [?] a partir de su versión 7 la cual permite trabajar de manera nativa con expresiones regulares.

Librerías para analizar archivos EML

La mejor forma de leer archivos EML en Java es haciendo uso de la librería JavaMail [?]. Esta es una librería opcional en Java SE y está incluida en Java EE por lo que tiene soporte por parte de Oracle [?].

Librerías para archivos HTML

Para leer archivos con formato HTML se puede usar la librería Jsoup [?].

Otras librerías que también pueden ser interesantes para leer archivos de la familia XML puede ser Jerry [?] con Lagarto [?].

Funciones hash

Los algoritmos de hash en Java se encuentran en la clase MessageDigest [?] y cuentan con al menos las funciones MD5, SHA-1 y SHA-256.

Facilidad para crear una web mediante dicho lenguaje

Lenguajes debido a la máquina virtual donde se ejecuta código, por eso se utilizar un servidor Tomcat [?].

Encontrar hostings que soporten este tipo de servidores no es tan fácil como servidores que ejecuten código PHP.

Dentro de las librerías más famosas para crear una web con java están Spring [?] y Structs [?].

Documentación

Java posee una documentación muy detallada, además no es difícil encontrar una cantidad de ayuda y recursos en la web. [?]

Conclusión

Java es un lenguaje desarrollado por Oracle con una gran trayectoria, lo que hace que tenga una documentación muy buena, multitud de ejemplos en la web y una gran cantidad de librerías.

Además, es un lenguaje compilado lo que debería hacer que fuese rápido si no se ejecutase en una máquina virtual [**?**], lo que hace que consuma muchos recursos, especialmente RAM.

4.8.2. Python

Python es un lenguaje de muy alto nivel, que compila a código C.

Su rendimiento no es tan alto como otros lenguajes de bajo nivel, pero es sencillo de programar.

Soporte para bases de datos

Actualmente Python posee varios conectores para distintos gestores de bases de datos relacionales como los ya mencionados, algunos de estos son Mysql [**?**], MariaDB que utiliza el mismo conector que MySQL [**?**], o Postgresql [**?**].

En caso de migrar posteriormente a MongoDB, también posee el controlador Pymongo [**?**].

Soporte para expresiones regulares

Para el uso de expresiones regulares Python 3 cuenta con la librería re [**?**].

Librerías para analizar archivos EML

Para analizar archivos con formato EML, se puede usar el paquete eml-parser [**?**] o el paquete emaildata [**?**].

Librerías para archivos HTML

Python 3 cuenta con un módulo llamado HTMLParser [**?**] destinado a leer archivos con formato HTML.

Funciones hash

Las funciones hash están implementadas en el módulo hashlib [**?**] y tiene soporte para sha1, sha224, sha256, sha384, sha512, o md5 entre otras.

Facilidad para crear una web mediante dicho lenguaje

Actualmente existen varios frameworks que permiten crear una web con Python, los más famosos son Django [**?**] y Flask [**?**].

Documentación

La documentación de Python se puede encontrar en su página oficial [?] y aunque se echa en falta algo más de claridad, no es difícil encontrar buenos ejemplos en internet.

Conclusión

Python es un lenguaje que permite un desarrollo rápido, con una gran comunidad de usuarios y con gran cantidad de librerías. Sin embargo, su sintaxis es bastante distinta a la de los demás lenguajes, tampoco tiene un gran rendimiento y su documentación no es muy completa.

4.8.3. Node

Soporte para bases de datos

Node posee varios paquetes para conectarse a bases de datos SQL como el paquete mysql [?], o el paquete mysql2 [?].

Por otro lado, en caso de querer usar una base de datos Mongo, tiene el paquete mongodb [?], además cuenta con el ORM mongoosejs [?].

Soporte para expresiones regulares

Node tiene soporte nativo para expresiones regulares mediante la clase RegExp [?].

Librerías para analizar archivos EML

Para leer un archivo EML con node se puede usar el paquete mailparser [?] o el paquete eml-format [?].

Librerías para archivos HTML

Los archivos de la familia XML, entre ellos los que tienen formato HTML, puede ser analizados con el paquete htmlparser2 [?].

Funciones hash

Para utilizar funciones hash Node cuenta con el módulo Crypto [?]. La lista de algoritmos disponibles depende del módulo OpenSSL que tenga instalado el sistema operativo [?] y se puede obtener de la siguiente forma:

```
let crypto = require('crypto');
let listOfSupportedHashes = crypto.getHashes();
```

Facilidad para crear una web mediante dicho lenguaje

Uno de los principales usos de Node es para la creación de webs, por lo que ofrece un amplio soporte para ello. Uno de los principales frameworks para este propósito es Expressjs [**?**].

Documentación

La documentación de Node es muy completa y con varios ejemplos de uno dentro de la misma web [**?**].

Conclusión

Pese a ser un lenguaje interpretado tiene un rendimiento muy aceptable gracias al buen trabajo de Google con su motor de JavaScript V8 [**?**], además gracias a su programación mediante callbacks y promesas hacen que sea especialmente interesante ya que se pueden realizar tareas en paralelo de manera sencilla.

Su documentación es excelente, aunque no tiene tantas librerías como los otros lenguajes.

Por otro lado, permite tener un único lenguaje de programación para todo el proyecto, tanto para la parte del análisis como para la parte de la web.

4.8.4. PHP

Soporte para bases de datos SQL

PHP ofrece soporte nativo para una gran cantidad de bases de datos relacionales mediante la clase PDO [**?**], como MySQL o PostgreSQL [**?**].

Soporte para expresiones regulares

Las expresiones regulares pueden usarse de manera nativa en PHP como demuestra su documentación [**?**].

Librerías para analizar archivos EML

Para poder leer los archivos con formato EML, se puede usar de manera nativa la familia de funciones mailparse [**?**]. Para un soporte más completo se puede usar la librería php-mime-mail-parser [**?**].

Librerías para archivos HTML

PHP tiene de forma nativa un conjunto de clases para tartar con archivos de la familia XML como sería el caso del formato HTML [**?**].

Funciones hash

Este lenguaje soporta de manera nativa una gran cantidad de funciones hash de manera nativa [**?**]. Facilidad para crear una web mediante dicho lenguaje PHP es uno de los lenguajes más utilizados para web, de hecho, es prácticamente el uso principal de este lenguaje, por lo que se ofrece de manera nativa una gran cantidad de facilidades para crear webs. Aunque también posee varios frameworks como Synfony [**?**] o Laravel [**?**].

Documentación

La documentación es muy completa, teniendo soporte incluso en español [**?**]. Además, gracias a su larga trayectoria se pueden encontrar de manera sencilla multitud de ejemplos prácticos para una gran cantidad de problemas.

Conclusión

PHP es un lenguaje que está especialmente diseñado para web, tiene una gran trayectoria y aunque su rendimiento era bastante pobre, ha mejorado en gran medida en su versión 7. [**?**]

De todos los lenguajes es el único que permitiría hacer el proyecto de manera prácticamente nativa, especialmente interesante es el uso de la clase PDO, que te permite cambiar de un gestor de base de datos a otro de manera practica mente transparente.

Su documentación es aceptable, con ejemplos y la única disponible en español, también tiene una gran comunidad, lo que facilita encontrar ejemplos a problemas conocidos.

4.8.5. Rendimiento en expresiones regulares

Se ha encontrado un benchmark [**?**] totalmente independiente en el que se analiza el rendimiento con expresiones, además, con expresiones regulares similares a las usadas en este proyecto, de diferentes lenguajes de programación. De todos los analizados, se va a extraer la información de los que conciernen a este proyecto.

Language	Email (ms)	URI (ms)	IP (ms)	Total (ms)
PHP	29.20	28.05	9.12	66.37
Node.js	81.87	67.16	2.04	151.07
Python PyPy3	354.38	314.82	367.19	1036.39
Java	310.30	326.81	732.19	1369.30

Tabla 4.2: Comparativa del rendimiento en la evaluación de expresiones

Como puede verse en la tabla 4.2, el más rápido es PHP, a una distancia prudente está Node, muy por detrás quedan Python y Java.

4.8.6. Conclusión

De todos los lenguajes analizados, los más interesantes son PHP y Node.

PHP destaca por tener de manera nativa prácticamente todo lo necesario para el desarrollo, por la facilidad para cambiar a otro tipo de base de datos relacional de manera prácticamente transparente y por su alto rendimiento.

En el otro lado, Node también tiene un buen rendimiento, existen todas librerías necesarias para este proyecto, su documentación es muy buena y permitiría tener un único lenguaje de programación para todo el desarrollo.

Resultado final

Finalmente y comparando las ventajas e inconvenientes que nos ofrencen ambos lenguajes, se ha optado por elegir PHP ya que, por un lado prácticamente no es necesario instalar dependencias externas y por otro migrar de un gestor de base de datos de otro, siempre que sean relacionales como es el caso, es bastante sencillo.

Capítulo 5

Implementación

5.1. Expresiones regulares

Se ha elegido esta forma de extraer los datos por ser una forma rápida y eficiente de extraer datos que comparten un patrón más o menos definido.

5.1.1. Direcciones IP

IPv4

Como se ha comentado en el apartado 4.2.1 una dirección IP en su versión 4 está definida por 4 números. Su valor va del 0 al 255 y están separados por un punto.

Lo primero que se va a crear es una expresión regular que permita encontrar números del 0 al 255, tanto añadiendo ceros a la izquierda, como sin ellos.

Esto se puede hacer de la siguiente forma:

```
25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?
```

Tal y como se puede ver, hay tres posibles alternativas:

1. `25[0-5]`: Un número que empiece por 25 y termine con un número entre el 0 y el 5 ambos incluidos. Números incluidos den este rango [250-255].

2. `2[0-4][0-9]`: un número que empiece por dos, que esté sucedido de un número del 0 al 4 (El 5 se ha contemplado en la primera opción) y que finalmente esté sucedido de un tercer número del 0 al 9. Números incluidos den este rango [200-249].

3. `[01]?[0-9][0-9]?`: En este rango se contemplan todos los número del 0 al 199, ya sea empezando con ceros o sin ellos.

Esta expresión regular sólo detecta un número del 0 al 255, para detectar una dirección IPv4 completa sería necesario una expresión regular de la siguiente forma:

```
(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.)
{3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)
```

Código 5.1: Expresión regular IPv4

La expresión regular anterior se introduce en un grupo, al que se le agrega "\." para detectar los puntos que separan los números. Con "{3}" se indica que va a haber exactamente 3 grupos de este tipo y finalmente se añade de nuevo la primera expresión regular para detectar el último número del 4 grupo, esta vez sin punto.

IPv6

La expresión regular para detectar una dirección IP en versión 6 es mucho más compleja debido a la gran cantidad de posibles modificaciones que se pueden hacer. Ya que una dirección IPv6 está formada por 8 números en hexadecimal cuyo valor puede ir del 0 al FFFF y se pueden escribir tanto en mayúsculas como en minúsculas. Estos números se pueden detectar mediante la siguiente expresión regular.

```
[0-9a-fA-F]{1,4}
```

Código 5.2: Expresión regular para capturar un número exadecimal de 4 dígitos

También es necesario tener en cuenta que los dos últimos números pueden ser sustituidos por una dirección IP en versión 4, para lo que se utilizará la expresión regular del apartado anterior (Código 5.1), aunque se va a sustituir por "`${ipv4}`" con el objetivo de facilitar la comprensión ya que, de otra manera, quedan expresiones tan largas que es complicado leerlas.

La siguiente expresión regular puede capturar o bien dos números en hexadecimales separados por ":" o bien una dirección IPv4 y de aquí en adelante será sustituida por "`${2groups_or_ipv4}`":

```
(?:(?:[0-9a-fA-F]{1,4}:[0-9a-fA-F]{1,4})|${ipv4})
```

Código 5.3: Expresión regular para capturar los dos últimos números o una dirección IP en versión 4

Por último, recordar que los números cuyo valor valga 0, se pueden contraer mediante "::".

A continuación, se van a analizar nueve casos, puestos en orden de mayor a menor cantidad de posibles números contraídos, para así jugar con las posibles posiciones del valor "::".

Esto se hace de esta manera para evitar que se capturen subpartes de la dirección y no la dirección completa.

Dirección completa En este caso se va a contemplar que la dirección esté completa y por tanto no haya ninguna contracción. En este caso la expresión es la siguiente:

```
(?:(?:(?:[0-9a-fA-F]{1,4}:){6})${2groups_or_ipv4})
```

Código 5.4: Expresión regular para capturar dirección IPv6 completa

En ella se distinguen dos partes, la primera está compuesta simplemente por 6 números hexadecimales de 4 dígitos como ya se ha visto (Código 5.2), separados por ":" `(?:(?:[0-9a-fA-F]{1,4}:){6})` y la segunda parte de la expresión también ha sido analizada (Código 5.3) y, o bien captura dos números hexadecimales separados por ":" o bien una dirección IPv4.

Contracción del primer número El siguiente caso detecta una dirección cuyo primer número está contraído.

La expresión es muy similar a la anterior:

```
(?:(?::(?:[0-9a-fA-F]{1,4}:){5})${2groups_or_ipv4
    })
```

Código 5.5: Expresión regular para capturar dirección IPv6 con el primer número contraido

Simplemente se añaden los "::" de la contracción y se reduce de 6 a 5 los posibles siguientes números hexadecimales.

Contracción del segundo número (Y tal vez del primero) En la contracción del segundo número hay que tener en cuenta que el primer número también puede estar contraido, por ejemplo la dirección:

```
0000:0000:aaaa:bbbb:cccc:dddd:eeee:ffff
```

Podría contraerse como:

```
0000::aaaa:bbbb:cccc:dddd:eeee:ffff
```

O como:

```
::aaaa:bbbb:cccc:dddd:eeee:ffff
```

Y todas son correctas direcciones correctas.

Por este motivo, se debe tener en cuenta que ese primer número puede, o no, estar.

La expresión regular queda como sigue:

```
(?:(?:(?:[0-9a-fA-F]{1,4})?::(?:[0-9a-fA-F]{1,4}:){4})
    ${2groups_or_ipv4})
```

Código 5.6: Expresión regular para capturar dirección IPv6 con el segundo número contraido (Y posibles anteriores)

En primer lugar puede verse como se reduce de nuevo en uno los números tras la contracción, pasando de cinco a cuatro.

En segundo lugar, se contempla que el primer número puede, o no, estar `(?:[0-9a-fA-F]{1,4})?`, finalmente se contempla que los dos últimos números sean una dirección IPv4 (Código 5.3).

Contracción del tercer número (Y tal vez primer y/o segundo número) En este caso el número de posibilidades aumenta considerablemente, en cambio la lógica para detectar el patrón es muy similar.

La expresión regular es la siguiente:

```
(?:(?:(?:[0-9a-fA-F]{1,4})?::(?:[0-9a-fA-F]{1,4}:)
    {4})${2groups_or_ipv4})
```

Código 5.7: Expresión regular para capturar dirección IPv6 con el tercer número contraido (Y posibles anteriores)

En esta expresión regular hay que tener en cuenta que la contracción representa al tercer número.

También se sabe que los números anteriores pueden o no estar contraídos, tal y como sucedía en el caso anterior y que los siguientes van a estar.

Por tanto la expresión regular tras "::" queda exactamente igual salvo por que se tienen que reducir de cuatro a tres los números posteriores `::(?:[0-9a-fA-F]{1,4}:){3}`.

Delante de esto se tienen que poder detectar tanto el primer, como el segundo número contraídos y sin contraer. Y esto se hace de la siguiente forma:

```
(?:(?:[0-9a-fA-F]{1,4}:){0,1}[0-9a-fA-F]{1,4})?
```

En primer lugar, se tiene en cuenta que pueden estar los dos números sin contraerse.

En caso estar contraído únicamente el segundo número, se detectaría el primer número gracias patrón `[0-9a-fA-F]{1,4}`.

Finalmente, cabe la posibilidad de que no haya nada delante, de ahí que los patrones anteriores estén centro de un grupo de la expresión regular seguido del signo "?".

Contracción del cuarto número (Y posibles anteriores) La lógica seguida en este caso y posteriores es muy similar, simplemente se aumenta en uno los posibles números delante de la contracción y se disminuyen los que van detrás, quedando como sigue:

```
(?:(?:(?:(?:[0-9a-fA-F]{1,4}:){0,2}[0-9a-fA-F]{1,4})
    ?::(?:[0-9a-fA-F]{1,4}:){2})${2groups_or_ipv4})
```

Código 5.8: Expresión regular para capturar dirección IPv6 con el cuarto número contraido (Y posibles anteriores)

Contracción del quinto número (Y posibles anteriores) En la contracción del quinto número se evitan las llaves de los números que van a ir tras la contracción, porque al salir sólo uno, no es necesario indicarlo.

```
(?:(?:(?:(?:[0-9a-fA-F]{1,4}:){0,3}[0-9a-fA-F]{1,4})
    ?::[0-9a-fA-F]{1,4}:)${2groups_or_ipv4})
```

Código 5.9: Expresión regular para capturar dirección IPv6 con el quinto número contraido (Y posibles anteriores)

Contracción del sexto número: (Y posibles anteriores) Tras la contracción, sólo puede haber dos números en hexadecimal o una dirección IPv4.

```
(?:(?:(?:(?:[0-9a-fA-F]{1,4}:){0,4}[0-9a-fA-F]{1,4})
    ?::)${2groups_or_ipv4})
```

Código 5.10: Expresión regular para capturar dirección IPv6 con el sexto número contraido (Y posibles anteriores)

Contracción del séptimo número: (Y posibles anteriores) Al estar la contracción en el séptimo número, tras esta sólo puede ir el octavo número.

```
(?:(?:(?:(?:[0-9a-fA-F]{1,4}:){0,5}[0-9a-fA-F]{1,4})
    ?::)[0-9a-fA-F]{1,4})
```

Código 5.11: Expresión regular para capturar dirección IPv6 con el sétimo número contraido (Y posibles anteriores)

Contracción del octavo número: (Y posibles anteriores) Al contraer el octavo número, ya no es posible que haya más números. Lo que sí es posible, es que todos los anteriores estén contraídos, en ese caso se tiene en cuenta la dirección "::", la cual por supuesto, es válida.

```
(?:(?:(?:[0-9a-fA-F]{1,4}:){0,6}[0-9a-fA-F]{1,4})?::)
```

Código 5.12: Expresión regular para capturar dirección IPv6 con el octavo número contraido (Y posibles anteriores)

Expresión final Al juntar todos los casos las expresiones regulares quedan de la siguiente manera:

```
(?:(?:(?:[0-9a-fA-F]{1,4}:){6})${2groups_or_ipv4})|

(?:(?:::(?:[0-9a-fA-F]{1,4}:){5})${2groups_or_ipv4})|

(?:(?:(?:[0-9a-fA-F]{1,4})?::(?:[0-9a-fA-F]{1,4}:){4})
    ${2groups_or_ipv4})|

(?:(?:(?:(?:[0-9a-fA-F]{1,4}:){0,1}[0-9a-fA-F]{1,4})
    ?::(?:[0-9a-fA-F]{1,4}:){3})${2groups_or_ipv4})|

(?:(?:(?:(?:[0-9a-fA-F]{1,4}:){0,2}[0-9a-fA-F]{1,4})
    ?::(?:[0-9a-fA-F]{1,4}:){2})${2groups_or_ipv4})|

(?:(?:(?:(?:[0-9a-fA-F]{1,4}:){0,3}[0-9a-fA-F]{1,4})
    ?::[0-9a-fA-F]{1,4}:)${2groups_or_ipv4})|

(?:(?:(?:(?:[0-9a-fA-F]{1,4}:){0,4}[0-9a-fA-F]{1,4})
    ?::)${2groups_or_ipv4})|

(?:(?:(?:(?:[0-9a-fA-F]{1,4}:){0,5}[0-9a-fA-F]{1,4})
    ?::)[0-9a-fA-F]{1,4})|

(?:(?:(?:[0-9a-fA-F]{1,4}:){0,6}[0-9a-fA-F]{1,4})?::)
```

Código 5.13: Expresiones regulares IPv6 con sustituciones

Y al deshacer todas sustituciones, queda como sigue:

```
(?:(?:(?:[0-9a-fA-F]{1,4}:){6})(?:(?:[0-9a-fA-F
    ]{1,4}:[0-9a-fA-F]{1,4})
    |(?:(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.)
    {3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?))))|

(?:(?:::(?:[0-9a-fA-F]{1,4}:){5})(?:(?:[0-9a-fA-F
    ]{1,4}:[0-9a-fA-F]{1,4})
    |(?:(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.)
    {3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?))))|

(?:(?:(?:[0-9a-fA-F]{1,4})?::(?:[0-9a-fA-F]{1,4}:){4})
    (?:(?:[0-9a-fA-F]{1,4}:[0-9a-fA-F]{1,4})
    |(?:(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.)
    {3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?))))|
```

```
(?:(?:(?:(?:[0-9a-fA-F]{1,4}:){0,1}[0-9a-fA-F]{1,4})
    ?::(?:[0-9a-fA-F]{1,4}:){3})(?:(?:[0-9a-fA-F
    ]{1,4}:[0-9a-fA-F]{1,4})
    |(?:(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.)
    {3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?))))|

(?:(?:(?:(?:[0-9a-fA-F]{1,4}:){0,2}[0-9a-fA-F]{1,4})
    ?::(?:[0-9a-fA-F]{1,4}:){2})(?:(?:[0-9a-fA-F
    ]{1,4}:[0-9a-fA-F]{1,4})
    |(?:(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.)
    {3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?))))|

(?:(?:(?:(?:[0-9a-fA-F]{1,4}:){0,3}[0-9a-fA-F]{1,4})
    ?::[0-9a-fA-F]{1,4}:)(?:(?:[0-9a-fA-F]{1,4}:[0-9a-
    fA-F]{1,4})
    |(?:(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.)
    {3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?))))|

(?:(?:(?:(?:[0-9a-fA-F]{1,4}:){0,4}[0-9a-fA-F]{1,4})
    ?::)(?:(?:[0-9a-fA-F]{1,4}:[0-9a-fA-F]{1,4})
    |(?:(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.)
    {3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?))))|

(?:(?:(?:(?:[0-9a-fA-F]{1,4}:){0,5}[0-9a-fA-F]{1,4})
    ?::)[0-9a-fA-F]{1,4})|

(?:(?:(?:[0-9a-fA-F]{1,4}:){0,6}[0-9a-fA-F]{1,4})?::)
```

Código 5.14: Expresiones regulares IPv6 sin sustituciones

Quedando finalmente:

```
(?:(?:(?:[0-9a-fA-F]{1,4}:){6})(?:(?:[0-9a-fA-F
    ]{1,4}:[0-9a-fA-F]{1,4})
    |(?:(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.)
    {3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?))))
    |(?:(?:::(?:[0-9a-fA-F]{1,4}:){5})(?:(?:[0-9a-fA-F
    ]{1,4}:[0-9a-fA-F]{1,4})
    |(?:(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.)
    {3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?))))
    |(?:(?:(?:[0-9a-fA-F]{1,4})?::(?:[0-9a-fA-F]{1,4}:)
    {4})(?:(?:[0-9a-fA-F]{1,4}:[0-9a-fA-F]{1,4})
    |(?:(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.)
    {3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?))))
    |(?:(?:(?:(?:[0-9a-fA-F]{1,4}:){0,1}[0-9a-fA-F
```

```
]{1,4})?::(?:[0-9a-fA-F]{1,4}:){3})(?:(?:[0-9a-fA-F
]{1,4}:[0-9a-fA-F]{1,4})
|(?:(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.)
{3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?))))
|(?:(?:(?:(?:[0-9a-fA-F]{1,4}:){0,2}[0-9a-fA-F
]{1,4})?::(?:[0-9a-fA-F]{1,4}:){2})(?:(?:[0-9a-fA-F
]{1,4}:[0-9a-fA-F]{1,4})
|(?:(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.)
{3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?))))
|(?:(?:(?:(?:[0-9a-fA-F]{1,4}:){0,3}[0-9a-fA-F
]{1,4})?::[0-9a-fA-F]{1,4}:)(?:(?:[0-9a-fA-F
]{1,4}:[0-9a-fA-F]{1,4})
|(?:(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.)
{3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?))))
|(?:(?:(?:(?:[0-9a-fA-F]{1,4}:){0,4}[0-9a-fA-F
]{1,4})?::)(?:(?:[0-9a-fA-F]{1,4}:[0-9a-fA-F]{1,4})
|(?:(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.)
{3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?))))
|(?:(?:(?:(?:[0-9a-fA-F]{1,4}:){0,5}[0-9a-fA-F
]{1,4})?::)[0-9a-fA-F]{1,4})|(?:(?:(?:[0-9a-fA-F
]{1,4}:){0,6}[0-9a-fA-F]{1,4})?::)
```

Código 5.15: Expresión regular para capturar direcciones IPv6

5.1.2. Dominios

Como se ha comentado en el apartado 4.2.2, un dominio está formado por dos o más partes separadas por ".", siendo la última parte leyendo el dominio de izquierda a derecha el TDL. Obtener esta parte por separado es muy interesante para determinar si un dominio es válido o no.

Además, sólo hay ciertos caracteres permitidos, como se vio en el punto 4.2.2, y un número máximo de caracteres.

Por tanto la expresión regular diseñada para detectar los dominios es la siguiente

```
[a-zA-z](?:[a-zA-Z0-9]|(?:(?:[\.\-_])\w))
    {1,252}\.(?<tdl>[a-zA-Z]{2,6})\.?
```

Código 5.16: Expresión regular para capturar dominios

Lo primero que se exige es que el dominio comience con una letra, a continuación se permiten un total de 252 caracteres válidos, es decir, alfanuméricos o uno de los siguientes símbolos (. - _), para acabar con un ".", seguido de dos a seis letras y con el último punto opcional.

Además, se ha creado un grupo para obtener el TDL, lo que será útil para hacer posibles validaciones.

5.1.3. Enlaces

Los enlaces que se van a analizar van a estar compuestos de un dominio seguido de una ruta, por tanto se aprovechará el código obtenido en el apartado anterior (Código 5.16).

El formato, los caracteres permitidos y el tamaño máximo de un enlace se estudió en el apartado 4.2.3.

La expresión regular para capturar enlaces es la siguiente.

```
(?<domain>[a-zA-z](?:[a-zA-Z0-9]|(?:(?:[\.\-_])\w)
    ){1,252}\.(?<tdl>[a-zA-Z]{2,6})\.?)(?:\:\d
    {2,5})?(?:\/[\#\%\~\$\-\_\.\+\!\*\(\)
    \,\;\/\?\:\@\=\&a-zA-z\d\n]*)
```

Código 5.17: Expresión regular para capturar enlaces

Como se puede ver, se ha creado un grupo para obtener el dominio y se da la posibilidad de obtener enlaces donde se especifica un puerto concreto «`(?:\:\d{2,5})?`», aunque esto no será lo habitual.

5.1.4. Direcciones de correo electrónico

Las direcciones de correo electrónico se estudiaron en el punto y la expresión regular es la siguiente:

```
[a-zA-Z0-9-_.]{1,64}@(?<domain>[a-zA-z](?:[a-zA-Z0
    -9]|(?:(?:[\.\-_])\w)){1,252}\.(?<tdl>[a-zA-Z
    ]{2,6})\.?)
```

Código 5.18: Expresión regular para capturar direcciones de correo electrónico

Como se puede ver, la expresión captura de uno a sesenta y cuatro de caracteres alfanuméricos además de los símbolos «`. - _`», seguido de un "@" y a continuación la expresión obtenida en el punto5.1.2 (Código 5.16).

5.2. Base de datos

En esta sección se van a comentar únicamente las tablas más relevantes o en las que sean más interesantes.

5.2.1. Datos comunes

Identificadores

Para los identificadores se han elegido datos enteros y sin signo de distintos tamaños, desde a.

Se han definido con "`auto_increment`" para aumenten de valor automáticamente.

Hash

Los hashes se van a guardar como datos binarios de 20 bytes, tal y como lo devuelve la función de hash sha1.

Fechas

Algunas fechas se han definido con "`DEFAULT CURRENT_TIMESTAMP(6)`" para que al insertarse se inserte la fecha actual directamente desde el gestor de base de datos.

5.2.2. Mensajes

La información de los mensajes está distribuida en varias tablas, a continuación se van a nombrar las más relevantes.

Tabla MessageStatus

La tabla MessageStatus B.3 permite saber si un mensaje ha sigo ya o no analizado. Esto permite poder separar por un lado el almacenamiento de los mensajes de su análisis, pudiendo de esta manera almacenar de manera rápida una gran cantidad de correos y posponer el análisis para más adelante.

Tabla MessageFormat

En la tabla MessageFormat B.1 se almacena el formato del correo para saber cómo debe ser analizado usando la librería correspondiente.

Tabla AnalyzedBy

La tabla AnalyzedBy B.2 permite saber con qué se ha analizado un mensaje, esta tabla permite realizar el análisis con varias versiones de software o con distintos tipos de lenguajes y en caso de haber algún error con alguno de ellos, sería fácil encontrar los mensajes que tienen el error.

Tabla MessageData

En la tabla MessageData B.4 se van a guardar todos los datos relativos a un mensaje, como su hash, cuándo fue analizado, cuándo fue añadido entre otros datos, pero no se guardará su contenido.

Se ha hecho esto para ganar rendimiento tal y como recomienda el manual de MySQL cunado se sabe que un campo de una tabla puede contener datos de gran tamaño [**?**].

Además, se ha creado un índice único con el campo hash, lo que por un lado va a facilitar las búsquedas con este dato y segundo, va a evitar datos duplicados [**?**].

Tabla MessageText

La tabla MessageText B.5 únicamente está destinada a almacenar el contenido de los mensajes. Además, se ha creado un índice FULLTEXT [**?**] para poder hacer consultas más complejas del texto contenido en cada mensaje.

Tabla ChildMessages

En la tabla ChildMessages B.6 se van a guardar todas las referencias de los mensajes contenidos en otros mensajes como sucede con los correos en formato eml.

5.2.3. Índices

Los índices utilizados son índices únicos y se han indexado:

- Algunos valores de patrones, como las direcciones IP o los dominios.
- Todos los hashes, para poder buscar por hash de manera rápida y evitar que haya datos repetidos cuando no se haya podido indexar el valor, como sucede con el texto de los mensajes.
- Todos los identificadores de las tablas intermedias por separado, para poder buscar en ambas direcciones.

No se han creado más índices porque por un lado, con cada índice se disminuye la velocidad de inserción [**?**] y por otro, con estos índices se cubren la mayoría de las búsquedas.

5.2.4. Vistas

Se han creado vistas para facilitar la búsqueda de información en a través de las relaciones, sobre todo, en las relaciones de muchos a muchos, esto

evita que se tenga que repetir consultas que se van a utilizar en multitud de ocasiones para permitir la navegabilidad en la web.

En concreto se han creado las siguientes vistas:

- `MessageText_view` (Vista B.8): Permite visualizar el mensaje en la web junto con toda la información relacionada con él.
- `MessageData_view` (Vista B.9): Permite visualizar todos los datos relacionados con un mensaje en concreto (Pero no el texto del mensaje)
- `IpAddress_view` (Vista B.10): Permite visualizar la dirección IP en la web junto con toda la información relacionada con él.
- `Domain_view` (Vista B.11): Permite visualizar el dominio en la web junto con toda la información relacionada con él.
- `domain_ip` (Vista B.12): permite buscar información sobre las direcciones IP buscando por el valor o el hash de un dominio concreto.
- `Subdomain` (Vista B.13): Permite saber todos los subdominios de un dominio concreto.
- `IpAddress_MessageData` (Vista B.14): Permite saber todos los mensajes que tienen una dirección IP concreta, buscando por su valor o hash.
- `ip_domain` (Vista B.15): permite obtener información sobre un dominio buscando por el valor o el hash de una dirección IP concreta.
- `Domain_MessageData` (Vista B.16): permite obtener información sobre los mensajes donde aparece un dominio concreto buscando por el hash o por el valor.
- `domain_url` (Vista B.17): permite buscar información sobre los enlaces de un dominio concreto buscando por el valor o el hash.
- `domain_email` (Vista B.18): permite buscar información sobre las direcciones de correo de un dominio concreto buscando por el valor o el hash.
- `EmailAddress_view` (Vista B.19): Permitie visualizar una dirección de correo y la información relacionada con ella.
- `EmailAddress_MessageData` (Vista B.20): permite obtener información sobre los mensajes donde aparece una dirección de correo concreta buscando por el hash o por el valor de la dirección.
- `Url_view` (Vista B.21): Permitie visualizar un enlace y la información relacionada con él.

- `Url_MessageData` (Vista B.22): permite obtener información sobre los mensajes donde aparece un enlace concreto buscando por el hash o por el valor del enlace.
- `MessageData_IpAddress_view` (Vista B.23): permite obtener información sobre todas las direcciones IP que aparecen en un mensaje buscando por el hash del mensaje.
- `MessageData_EmailAddress_view` (Vista B.24): permite obtener información sobre todas las direcciones de correo que aparecen en un mensaje buscando por el hash del mensaje.
- `MessageData_Domain_view` (Vista B.25): permite obtener información sobre todos los dominios que aparecen en un mensaje buscando por el hash del mensaje.
- `MessageData_Url_view` (Vista B.26): permite obtener información sobre todos los enlaces que aparecen en un mensaje buscando por el hash del mensaje.
- `MessageData_child_view` (Vista B.27): permite obtener información sobre todos los mensajes contenidos en el mensaje actual. (Se usa en los mensajes con formato EML)
- `MessageData_parent_view` (Vista B.28): permite obtener información sobre todos los mensajes que contienen en el mensaje actual. (Se usa para ver si el mensaje actual, está en alguno o en varios mensajes en formato EML)

5.3. Código

La aplicación está hecha en PHP y se ejecuta con Apache, como gestor de base de datos, se ha usado MySQL.

5.3.1. Estructura de carpetas

Se ha seguido el modelo "Modelo-vista-controlador" (ver punto 5.3.3) con la siguiente estructura de carpetas (Ver figura 5.1):

- Modelos: Son los encargados de acceder a la base de datos y obtener la información necesaria para generar la vista.
- Vista: En ella se mostrará el contenido que se ha obtenido del modelo.
- Controlador: Es el encargado de elegir la vista y el modelo adecuado. El controlador contiene todas las posibles rutas válidas de la aplicación.

- Dependencias: En esta carpeta están todas las dependencias usadas como Twig o la integración con Virus Total.

- Config: es donde se almacenan datos sensibles que no se deben escribir en código como usuarios y contraseñas de la base de datos, o la clave de la API de Virus Total.

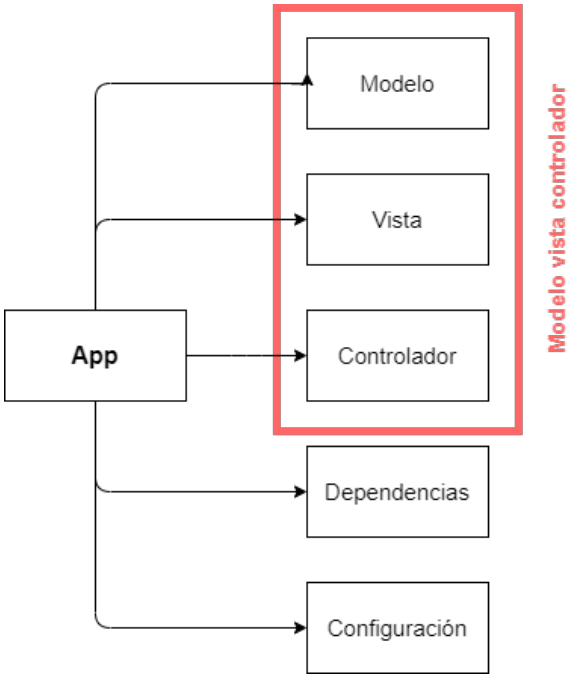


Figura 5.1: Estructura de carpetas de la aplicación

5.3.2. Diagrama de clases

Para facilitar la comprensión y que los diagramas se puedan interpretar de manera más sencilla, el diagrama de clases se va a dividir en cuatro partes.

1. Análisis: Esta parte se corresponde con el análisis de los mensajes. Estaría dentro del controlador del index y es la parte encargada de extraer los patrones.

2. Controladores.

3. Modelos.

4. Vistas.

Análisis

La parte del análisis está compuesta por tres partes (Ver figura 5.2):

1. Analyzer: Esta clase es la encargada de leer el mensaje en función del formato.

2. PatternSearcher: Es la clase encargada de buscar y extraer los patrones mediante las expresiones regulares que están almacenadas en la constante *PATTERNS*. También valida que en el caso de los dominios (O los enlaces o las direcciones de correo) tentan un TDL correcto.

3. PatternContainer: En esta clase se almacenarán todos los patrones encargados. Evitará que haya repetidos, añadirá las características y las relaciones necesarias y se encargará de almacenar los datos en la base de datos mendiante el método *save()*

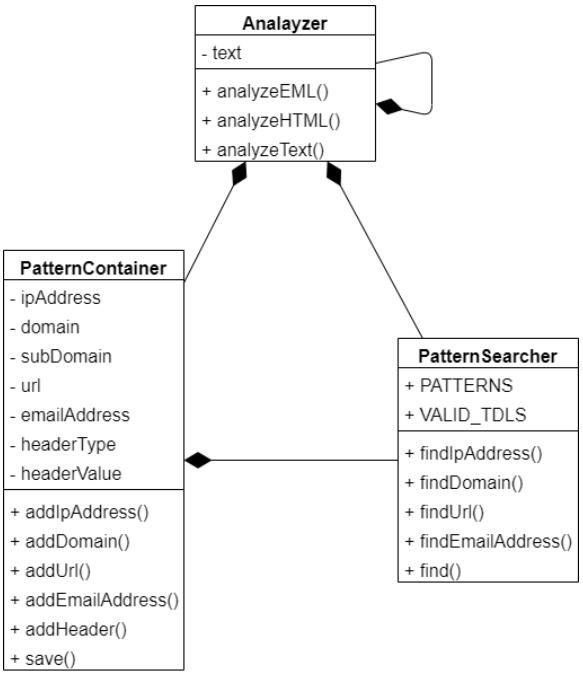


Figura 5.2: Estructura de carpetas de la aplicación

Controladores

Los controladores permiten al usuario navegar por la web, en el controlador index es donde se permite al usuario enviar a analizar los correos, los demás son de consulta (correos, direcciones IP, dominios, enlaces y direcciones de correo).

Un ejemplo de controlador sería el código 5.19

```
class Domain extends Controller
{
    function __construct($request){
        parent::__construct('Domain.html',
            $request);
    }
    public function render(){

        $value = $this->request->getNextElement();
        if($value){
            $info = DataAccessObjectFactory::
                getDataAccessObject("
                DomainDAO_SQL_PDO")->getInfo($value
                );
            $this->addArgument("info", $info);
        }
        return parent::renderDirectly();
    }
}
```

Código 5.19: Ejemplo de controlador

Todos los controladores heredan de una clase abstracta llamada Controller (Ver figura 5.3), en ella se almacenan todos los métodos genéricos, como la llamada a la librería de Twig o el nombre de la vista. También se crea un objeto Request en el que almacena información sobre la dirección en la que el usuario a clicado, esto permite al controlador saber qué vista debe generar y a qué modelo llamar.

Clase Controller De esta clase destacan los siguientes métodos:

- notFound: se utiiza si se intenta acceder a una URL que el controlados no conoce.
- addArgument: permite a las clases que heredan de ella añadir argumentos que se pasarán a la vista.
- render: es el método que genera la vista y la envía una vez que tiene todos los argumentos necesarios para generarla.

Clase Request De esta clase destacan los siguientes métodos:

- getMethod: indica si el método con el que se hace la petición a Apache es get o Post.

- getNextElement: devuelve la siguiente parte del path si este está separado mediante "/".

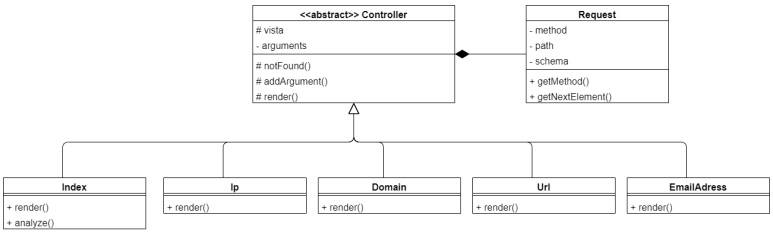


Figura 5.3: Diagrama de clases de los controladores

Modelos

Los modelos son los únicos encargados de interactuar con la base de datos, para hacer esto se ha hecho uso de la clase PDO de PHP, ya que aporta seguridad respecto a mysqli [**?**] y permite una migración sencilla a otro tipo de gestor de base de datos, como ya se ha mencionado (Ver apartado 4.8.4).

Para esto se ha utilizado la figura del Objeto de acceso a datos (DAO del inglés Data Access Object). Para evitar duplicar código, se ha creado una clase abstracta genérica llamada DataAccessObjectPDO de la cual extenderán las demás (Ver figura 5.4).

Esta clase está compuesta por un objeto de la clase DataBasePDO, en él se leerá un archivo de configuración con los parámetros de conexión a la base de datos. El separar estos dos objetos se hace para que puedan coexistir dos bases de datos al mismo tiempo.

Clase DataAccessObjectPDO De esta clase destacan los siguientes métodos:

- getConnection: Se utiliza para realizar la conexión con la base de datos.
- getConnectionReadOnly: Es similar al método anterior, pero no se podrán leer datos.
- getItems: Es un método genérico para poder obtener todos los datos de la tabla del modelo actual.
- pagination: permite obtener una fracción de los datos de la tabla del modelo actual.
- prepare: genera una consulta preparada.
- hash: se utiliza para generar el hash dado el valor.

- findOrInsertFromArrayValues: permite obtener información a partir de un array de datos y en caso de no encontrarse, se insertan los elementos.

- insertOrUpdateFromArrayValues: Se inserta un array de datos y en caso de existir ya el elemento, lo actualiza.

Las clases que hereden de esta, tendrán que implementar dos métodos, *getInfo()*, que se usa para obtener la información necesaria para la vista e *insert()* que permite insertar objetos desde los métodos *findOrInsertFromArrayValues()* o *insertOrUpdateFromArrayValues()*.

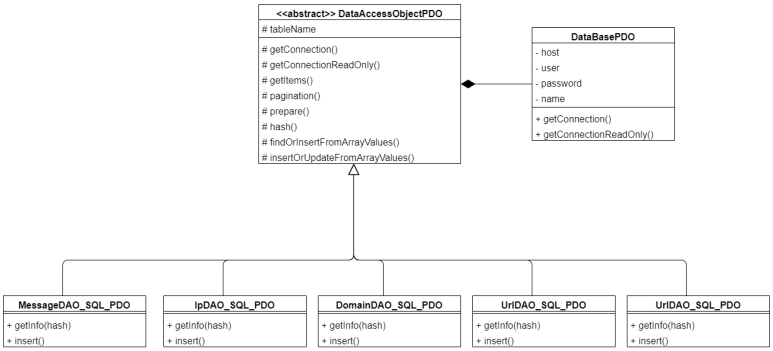


Figura 5.4: Diagrama de clases de los modelos

Vistas

Al ser una aplicación web, las vistas están hechas con HTML, CSS y JavaScript. En esta parte se ha utilizado Twig como gestor de plantillas y Bootstrap como librería para el diseño de la web.

Twig Twig es un gestor de plantillas desarrollado por Symfony [**?**] para PHP. Esto facilita la reutilización de código y el paso de información del modelo a la vista.

Gracias a la propiedad de herencia de Twig [**?**], se ha creado una plantilla base de la que heredarán las demás, de esta manera toda la información que es común se mantiene a lo largo de todas las vistas sin tener que repetir código, como por ejemplo la parte del header o del footer (ver figura 5.5). También se importará Bootstrap.

Bootstrap Bootstrap es una librería desarrollada por Twitter con una gran cantidad de opciones y facilidades para el diseño web [**?**]. Se ha utilizado sobre todo para hacer la web adaptable a dispositivos móviles.

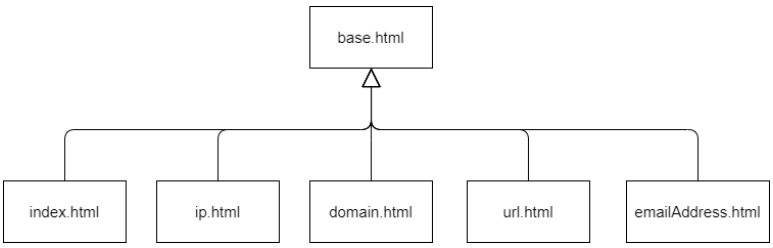


Figura 5.5: Diagrama de clases de las vistas

5.3.3. Diagrama de flujo

Modelo Modelo-Vista-Controlador

El modelo Modelo-Vista-Controlador se puede explicar mediante la secuencia sigueinte (Ver figura 5.6):

1. El cliente realiza una acción que es capturada por el controlador.
2. El controlador elige un modelo, que se encarga de acceder a la base de datos y obtener la información necesaria.
3. El modelo le envía la información obtenida al controlador.
4. El controlador elige la vista adecuada y le pasa la información recibida del modelo.
5. La vista interpreta la información, en base a ella genera la vista final y se la envía al controlador.
6. El controlador le envía la vista ya formada al cliente.

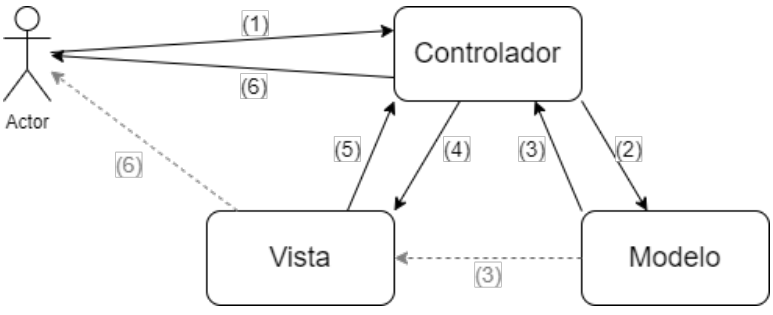


Figura 5.6: Diagrama de flujo del modelo Modelo-Vista-Controlador

Diagrama de flujo al llegar un nuevo mensaje

Al llegar un nuevo mensaje se siguen los siguientes pasos:

1. Se consulta si el correo ya ha sido analizado, en caso de haber sido analizado, se devuelve la información que haya en la base de datos.

2. En caso de no haber sido analizado, en función del formato del mensaje se siguen los siguientes pasos.

    a) EML: Si el formato es EML y tiene algún mensaje en el interior, se enviarán al punto 1 todos los mensajes que tengan. A continuación se analizarán las cabeceras, extrayendo por un lado el tipo y por otro el valor.

    `<Tipo de cabecera>:<valor>`

    Dentro de valor se intentarán extraer todos los posibles patrones como si se tratase de un texto.

    b) HTML: Si el mensaje está en formato HTML se observarán las etiquetas <a>, en concreto el valor del atributo "href" y del texto mostrado al usuario.

    c) Texto plano: Si el formato es texto plano, simplemente se extraen todos los patrones mediante expresiones regulares y se almacenan.

Hay que tener en cuenta que un texto en formato EML puede tener contenidos varios textos en varios formatos. Ver figura 5.7

Diagrama de flujo para analizar una etiqueta `<a>`

Para analizar la etiqueta `<a>`, esta debe tener un atributo "href" y texto. Si ambos coinciden, no se hace nada, en caso contrario, se aumenta el score del patrón del atributo "href". Ver figura 5.8.

Diagrama de flujo para analizar una etiqueta

En este apartado se irá comprobado si en el texto se encuentran patrones de los que se han obtenido las expresiones regulares en el punto 5.1. En caso de que haya, se comprobará si tiene características maliciosas como las mencionadas en el punto 4.4.1, en caso de tener alguna de ellas, se aumentará su score y se almacenará. Ver figura 5.9.

5.4. Capturas de pantallas

5.4.1. Index

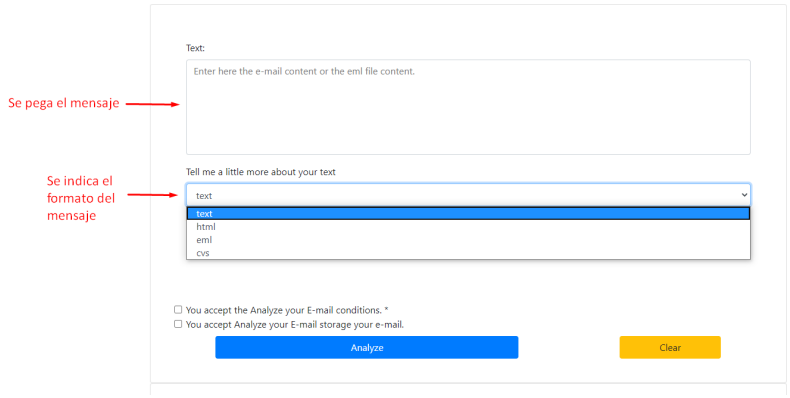
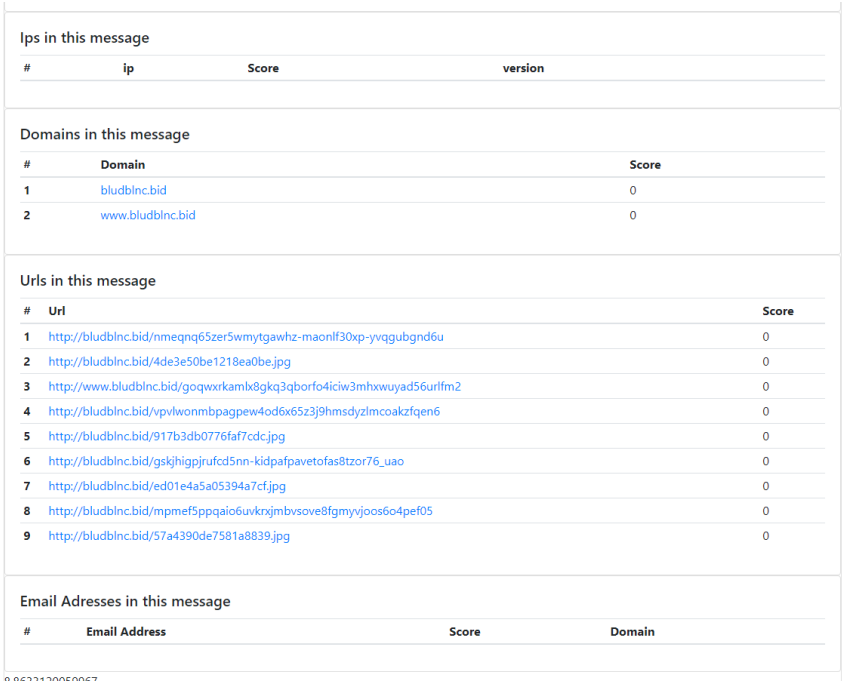


Figura 5.10: Index de la aplicación

5.4.2. Mensaje

Figura 5.11: Vista de un mensaje: Información básica



Ips in this message

#	ip	Score	version

Domains in this message

#	Domain	Score
1	bludblnc.bid	0
2	www.bludblnc.bid	0

Urls in this message

#	Url	Score
1	http://bludblnc.bid/nmeqnq65zer5wmytgawhz-maonlf30xp-yvqgubgnd6u	0
2	http://bludblnc.bid/4de3e50be1218ea0be.jpg	0
3	http://www.bludblnc.bid/goqwxrkamlx8gkq3qborfo4iciw3mhxwuyad56urlfm2	0
4	http://bludblnc.bid/vpvlwonmbpagpew4od6x65z3j9hmsdyzlmcoakzfqen6	0
5	http://bludblnc.bid/917b3db0776faf7cdc.jpg	0
6	http://bludblnc.bid/gskjhigpjrufcd5nn-kidpafpavetofas8tzor76_uao	0
7	http://bludblnc.bid/ed01e4a5a05394a7cf.jpg	0
8	http://bludblnc.bid/mpmef5ppqaio6uvkrxjmbvsove8fgmyvjoos6o4pef05	0
9	http://bludblnc.bid/57a4390de7581a8839.jpg	0

Email Adresses in this message

#	Email Address	Score	Domain

Figura 5.12: Vista de un mensaje: Patrones

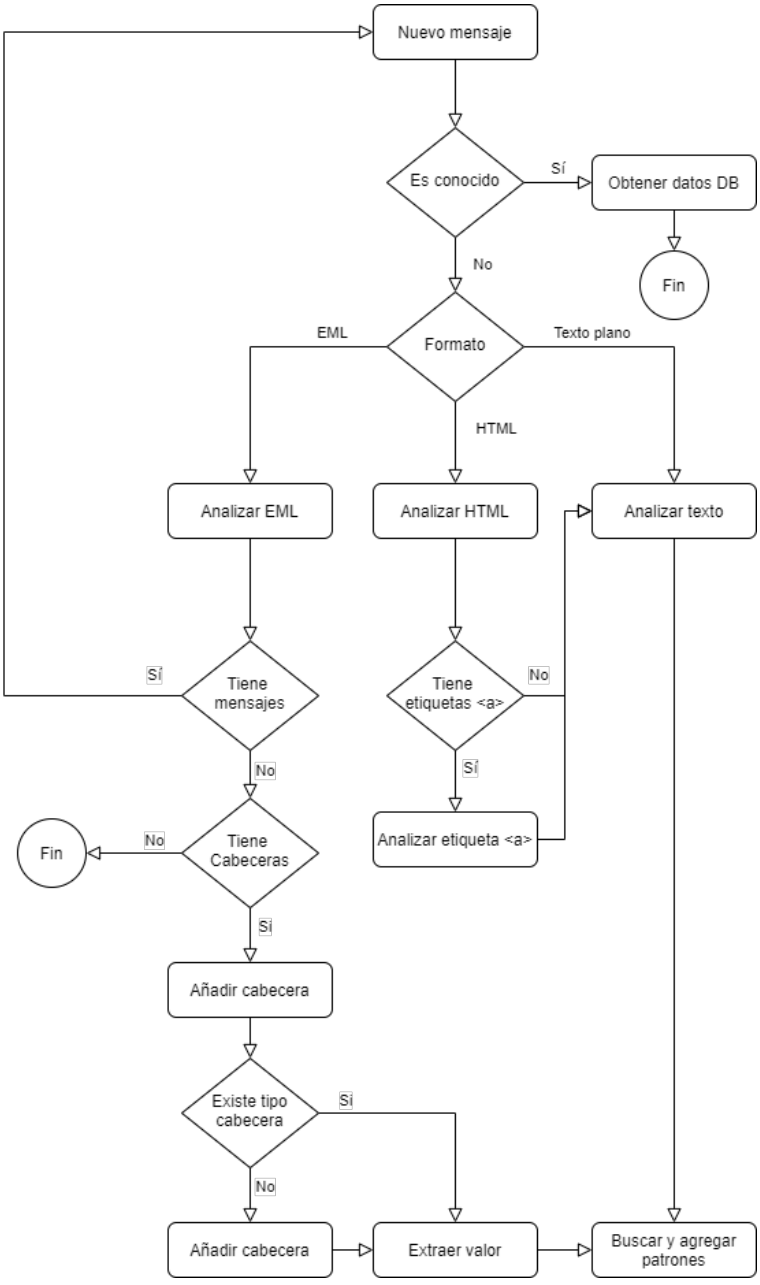


Figura 5.7: Diagrama de flujo cuando llega un nuevo mensaje

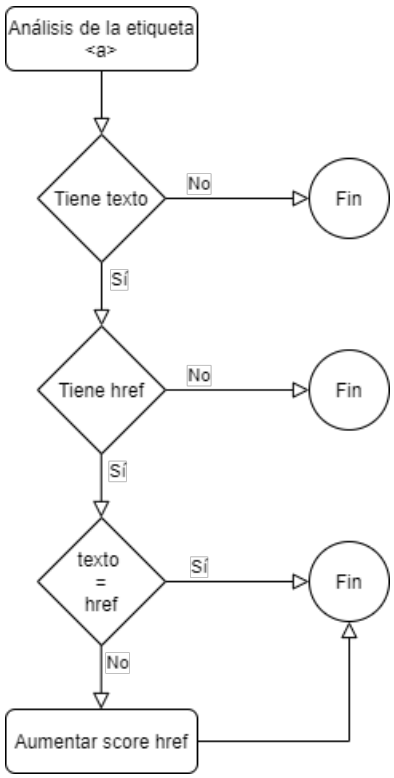


Figura 5.8: Diagrama de flujo para analizar las etiquetas `<a>`

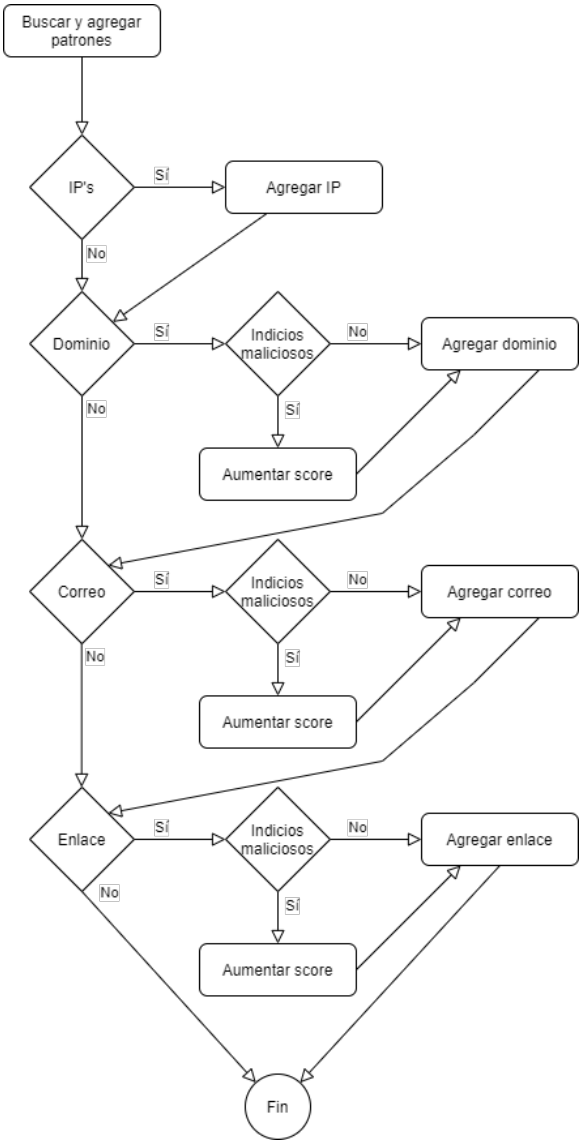


Figura 5.9: Diagrama de flujo para buscar y almacenar patrones

Apéndice A

Documentos

A.1. Ejemplo formato EML

```
Return-Path: <SRS0=v5D8h6=TB=rankgator.live=
    kelly@untroubled.org>
Delivered-To: bruce@home.untroubled.org
Received: from rankgator.live (rdns.bitacel.com
    [194.34.104.232])
  by pt02.futurequest.net ([69.5.6.173])
  with ESMTP via TCP; 01 May 2019 14:41:04 -0000
Message-ID: <xat683W4TM32ruh.ilcIL221JGS1tho@rankgator
    .live>
From: "Arctic Pain Relief" <kelly@rankgator.live>
Date: Wed, 01 May 2019 10:09:17 -0400
MIME-Version: 1.0
Subject: Are  you -part -of— the— 'worlds- biggest
    arthritis cover-up?
To: "bruce@untroubled.org" <bruce@untroubled.org>
Content-type: multipart/alternative; boundary="---=
    _Part_1426_87C624P49.F740FP9";
Content-Length: 1606

-----=_Part_1426_87C624P49.F740FP9
Content-Type: text/plain; format=flowed; charset="UTF
    -8"
Content-Transfer-Encoding: 7bit

Are you part of the 'worlds biggest arthritis cover-up?
```

```
http://efs5I69IX4S3elw.rankgator.live/208
    fdcae990695eb0665b0903677f315_8da8c95c-0101030c0001
    /1/

Are you part of the 'worlds biggest arthritis cover-up?

------=_Part_1426_87C624P49.F740FP9
Content-Type: text/html; charset=us-ascii
Content-Transfer-Encoding: 7bit

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
    Transitional//EN">
<html>
  <head>
    <title></title>
  </head>
  <body>
    <table>
      <tr>
        <td colspan='2' align='center' valign='middle'
            class='preview-mid'>
          <br>
          <center>
            <a href="http://sdvPL9B7FL5Hwfm.rankgator.
                live/208
                fdcae990695eb0665b0903677f315_8da8c95c
                -0101030c0001/1/"><img src="http://
                ifq7BTAN28Q1vfo.rankgator.live/images
                //4b128133049378e37097925975e252e1"
                border="0" alt=""></a>
          </center>
          <div align="center">
###STUFF##
            <font face="Verdana, Arial, Helvetica,
                sans-serif" size="1"><br>
            If you'd prefer not to receive future
                emails, <a href="http://pkvG0P5I93MHwgd
                .rankgator.live/208
                abda8ac8695eb0665b0903678f315_8da8c95c
                -0101030c0001/1/"><font color
                ="#666666">Unsubscribe Here</font></a
                >.<br>
            616 Corporate Way, | Suite 2-5335 Valley
                Cottage | NY NY 10989</font>
```

```
            </div>
          </td>
        </tr>
      </table>
    </body>
</html>

<center>



<img src="http://pmsG8XP16WP5bqb.rankgator.live
    /2082301ea94695eb0665b090365f030b_8da8c95c-0101030
    c0001/2/">

------=_Part_1426_87C624P49.F740FP9--
```

Código A.1: Ejemplo archivo EML

Apéndice B

Base de datos

B.1. Diagrama entidad relación

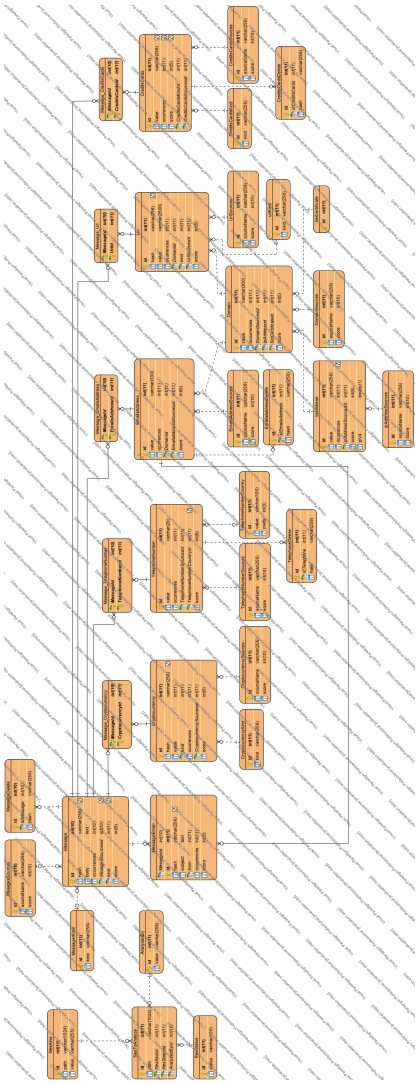


Figura B.1: Diagrama entidad entidad relación de la Base de datos

B.2. Tablas e índices

B.2.1. Mensajes

Formato de los mensajes

```
create table MessageFormat (
id    tinyint UNSIGNED not null auto_increment,
value varchar(256) not null,
primary key (id));
INSERT INTO MessageFormat(value) VALUES ("eml");
INSERT INTO MessageFormat(value) VALUES ("html");
INSERT INTO MessageFormat(value) VALUES ("text");
INSERT INTO MessageFormat(value) VALUES ("csv");
```

Tabla B.1: Tabla MessageFormat

Analizado por

```
create table AnalyzedBy (
id    tinyint UNSIGNED not null auto_increment,
value varchar(256) not null,
primary key (id));
INSERT INTO AnalyzedBy(value) VALUES ("php");
INSERT INTO AnalyzedBy(value) VALUES ("node");
```

Tabla B.2: Tabla AnalyzedBy

Estado del mensaje

```
create table MessageStatus (
id     tinyint UNSIGNED not null auto_increment,
value varchar(100) not null,
primary key (id));
INSERT INTO MessageStatus(value) VALUES ("pending"
    );
INSERT INTO MessageStatus(value) VALUES ("
    analyzing");
INSERT INTO MessageStatus(value) VALUES ("analyzed
    ");
INSERT INTO MessageStatus(value) VALUES ("error");
```

Tabla B.3: Tabla MessageStatus

Datos del mensaje

```
create table MessageData (
id              int UNSIGNED not null
    auto_increment,
hash            binary(20) not null,
score           smallint default 0 not null,
format          tinyint UNSIGNED not null,
analyzedBy      tinyint UNSIGNED null DEFAULT NULL,
status          tinyint UNSIGNED not null DEFAULT
    1,
analysisTime    DECIMAL(9,6) null DEFAULT NULL,
addeddAt        TIMESTAMP(6) NOT NULL DEFAULT
    CURRENT_TIMESTAMP(6),
analyzedAt      TIMESTAMP(6) NULL DEFAULT NULL,
updatedAt       TIMESTAMP(6) NULL DEFAULT NULL ON
    UPDATE CURRENT_TIMESTAMP(6),
UNIQUE INDEX iq_messageData_hash (hash ASC),
CONSTRAINT FK_MessageFormat_Message FOREIGN KEY (
    format) REFERENCES MessageFormat (id) ON DELETE
     NO ACTION ON UPDATE NO ACTION,
CONSTRAINT FK_AnalyzedBy_Message FOREIGN KEY (
    analyzedBy) REFERENCES AnalyzedBy (id) ON
    DELETE NO ACTION ON UPDATE NO ACTION,
CONSTRAINT FK_MessageStatus_Message FOREIGN KEY (
    status) REFERENCES MessageStatus (id) ON DELETE
     NO ACTION ON UPDATE NO ACTION,
primary key (Id));
```

Tabla B.4: Tabla MessageData

Texto del mensaje

```
create table MessageText (
id              int UNSIGNED not null
    auto_increment,
value           text CHARACTER SET utf8 COLLATE
    utf8_unicode_ci NOT null,
CONSTRAINT FK_MessageData_MessageText FOREIGN KEY
    (id) REFERENCES MessageData (id) ON DELETE
    CASCADE ON UPDATE NO ACTION,
FULLTEXT ftidx_MessageText_value (value),
```

```
    primary key (id));
```

Tabla B.5: Tabla MessageText

Submensajes

```
    create table ChildMessages (
    parentId          int UNSIGNED  not null,
    childId           int UNSIGNED  not null,
    CONSTRAINT
        FK_MessageData_Message_ChildMessages_Parent
        FOREIGN KEY (parentId) REFERENCES MessageData (
        id) ON DELETE CASCADE ON UPDATE NO ACTION,
    CONSTRAINT
        FK_MessageData_Message_ChildMessages_Child
        FOREIGN KEY (childId) REFERENCES MessageData (
        id) ON DELETE CASCADE ON UPDATE NO ACTION,
    primary key (parentId, childId));
```

Tabla B.6: Tabla ChildMessages

B.2.2. Patrones

```
    create table IpAddress (
  id          int UNSIGNED  not null auto_increment,
  hash        binary(20) not null,
  value       varchar(50) not null,
  score       smallint default 0 not null,
  ipv4        tinyint(1) not null,
  UNIQUE INDEX iq_IpAddress_value (value ASC),
  UNIQUE INDEX iq_IpAddress_hash (hash ASC),
  primary key (id));

create table Domain (
  id          int UNSIGNED  not null auto_increment,
  hash        binary(20) not null,
  value       varchar(330) not null,
  score       smallint default 0 not null,
  analyzedAt  TIMESTAMP(6) NOT NULL DEFAULT
    CURRENT_TIMESTAMP(6),
  parentId    int UNSIGNED  null,
  INDEX idx_domain_parentId (parentId ASC),
```

```
  UNIQUE INDEX iq_Domain_hash (hash ASC),
  primary key (id));
  ALTER TABLE Domain ADD CONSTRAINT
      fk_domain_parentId_id FOREIGN KEY (parentId)
      REFERENCES Domain (id) ON DELETE NO ACTION ON
      UPDATE NO ACTION;

create table Domain_IpAddress (
  DomainId        int UNSIGNED not null,
  IpAddressId     int UNSIGNED not null,
  CONSTRAINT FK_Domain__Domain_IpAddress FOREIGN KEY (
      DomainId) REFERENCES Domain (id) ON DELETE
      CASCADE ON UPDATE NO ACTION,
  CONSTRAINT FK_IpAddress__Domain_IpAddress FOREIGN
      KEY (IpAddressId) REFERENCES IpAddress (id) ON
      DELETE CASCADE ON UPDATE NO ACTION,
  primary key (DomainId, IpAddressId));


create table EmailAddress (
  id           int UNSIGNED not null auto_increment,
  hash         binary(20) not null,
  value        varchar(330) not null,
  score        smallint default 0 not null,
  analyzedAt   TIMESTAMP(6) NOT NULL DEFAULT
      CURRENT_TIMESTAMP(6),
  domain       int UNSIGNED not null,
  UNIQUE INDEX iq_EmailAddress_hash (hash ASC),
  CONSTRAINT FK_Domain__EmailAddress FOREIGN KEY (
      domain) REFERENCES Domain (id) ON DELETE NO
      ACTION ON UPDATE NO ACTION,
  primary key (id));


create table UrlType (
  id    tinyint UNSIGNED not null auto_increment,
  value varchar(256) not null,
  primary key (id));

INSERT INTO UrlType(value) VALUES ("link");
INSERT INTO UrlType(value) VALUES ("img");
INSERT INTO UrlType(value) VALUES ("video");

create table Url (
```

```
  id          int UNSIGNED  not null auto_increment,
  hash        binary(20) not null,
  value       varchar(2500) not null,
  score       smallint default 0 not null,
  analyzedAt  TIMESTAMP(6) NOT NULL DEFAULT
     CURRENT_TIMESTAMP(6),
  domain      int UNSIGNED  not null,
  type        tinyint UNSIGNED  not null,
  UNIQUE INDEX  iq_Url_hash (hash ASC),
  CONSTRAINT FK_Domain__Url FOREIGN KEY (domain)
     REFERENCES Domain (id) ON DELETE NO ACTION ON
     UPDATE NO ACTION,
  primary key (id));

create  table MessageHeaderTypes (
  id                      mediumint UNSIGNED  not null
      auto_increment,
  value                   text not null,
  primary key (id));

create  table MessageHeaderValues (
  id                      int UNSIGNED  not null
     auto_increment,
  hash                    binary(20) not null,
  value                   text not null,
  emailHeaderTypesId      mediumint UNSIGNED  not null
     ,
  UNIQUE INDEX  iq_Url_hash (hash ASC),
  INDEX idx_MessageHeaderValues_emailHeaderTypesId (
     emailHeaderTypesId ASC),
  CONSTRAINT
     FK_MessageHeaderTypes__MessageHeaderValues
     FOREIGN KEY (emailHeaderTypesId) REFERENCES
     MessageHeaderTypes (id) ON DELETE NO ACTION ON
     UPDATE NO ACTION,
  primary key (id));



create  table MessageDelete (
  id        int UNSIGNED  not null auto_increment,
  idMessage int UNSIGNED  not null,
  hash      varchar(256) not null,
```

```
  CONSTRAINT FK_MessageData_MessageDelete FOREIGN KEY
      (idMessage) REFERENCES MessageData (id) ON DELETE
      CASCADE ON UPDATE NO ACTION,
  primary key (id));

create  table MessageDataSources (
  id          SMALLINT UNSIGNED  not null
     auto_increment,
  value varchar(100) not null,
  score       smallint default 0 not null,
  UNIQUE INDEX uq_MessageDataSources_value (value DESC
     ),
  primary key (id));
INSERT INTO MessageDataSources(value) VALUES ("fromWeb
   ");
INSERT INTO MessageDataSources(value) VALUES ("onEml")
   ;


create  table MessageData_MessageDataSources (
  MessageDataId           int UNSIGNED  not null,
  MessageDataSourcesId  SMALLINT UNSIGNED  not null,
  date                TIMESTAMP(6) NOT NULL DEFAULT
     CURRENT_TIMESTAMP(6),
  CONSTRAINT
      FK_MessageData__Message_MessageDataSources
     FOREIGN KEY (MessageDataId) REFERENCES
      MessageData (id) ON DELETE CASCADE ON UPDATE NO
     ACTION,
  CONSTRAINT
      FK_MessageDataSources__Message_MessageDataSources
      FOREIGN KEY (MessageDataSourcesId) REFERENCES
      MessageDataSources (id) ON DELETE NO ACTION ON
     UPDATE NO ACTION,
  primary key (MessageDataId, MessageDataSourcesId,
      date));




create  table IpAddressSources (
```

```
  id          smallint UNSIGNED  not null
     auto_increment,
  value varchar(100) not null,
  score       smallint  default 0 not null,
  UNIQUE INDEX uq_IpAddressSources_value (value DESC),
  primary key (id));
INSERT INTO IpAddressSources(value) VALUES ("
   textAnalysis");
INSERT INTO IpAddressSources(value) VALUES ("
   emlHeaders");
INSERT INTO IpAddressSources(value) VALUES ("onDomain"
   );
INSERT INTO IpAddressSources(value) VALUES ("aHref");
INSERT INTO IpAddressSources(value) VALUES ("aText");


create  table IpAddress_IpAddressSources (
  IpAddressId          int UNSIGNED  not null,
  IpAddressSourcesId smallint UNSIGNED  not null,
  date                 TIMESTAMP(6) NOT NULL DEFAULT
     CURRENT_TIMESTAMP(6),
  CONSTRAINT FK_IpAddress__IpAddress_IpAddressSources
     FOREIGN KEY (IpAddressId) REFERENCES IpAddress (
     id) ON DELETE CASCADE ON UPDATE NO ACTION,
  CONSTRAINT
     FK_IpAddressSources__IpAddress_IpAddressSources
     FOREIGN KEY (IpAddressSourcesId) REFERENCES
     IpAddressSources (id) ON DELETE NO ACTION ON
     UPDATE NO ACTION,
  primary key (IpAddressId, IpAddressSourcesId, date))
     ;



create  table DomainSources (
  id          smallint UNSIGNED  not null
     auto_increment,
  value varchar(100) not null,
  score       smallint  default 0 not null,
  UNIQUE INDEX uq_DomainSources_value (value DESC),
  primary key (id));
INSERT INTO DomainSources(value) VALUES ("textAnalysis
   ");
```

```
INSERT INTO DomainSources(value) VALUES ("emlHeaders")
    ;
INSERT INTO DomainSources(value) VALUES ("onUrl");
INSERT INTO DomainSources(value) VALUES ("onSubdomain"
    );
INSERT INTO DomainSources(value) VALUES ("
    onEmailAddress");
INSERT INTO DomainSources(value) VALUES ("aHref");
INSERT INTO DomainSources(value) VALUES ("aText");

create  table Domain_DomainSources (
  DomainId            int UNSIGNED  not null,
  DomainSourcesId     smallint UNSIGNED  not null,
  date                TIMESTAMP(6) NOT NULL DEFAULT
     CURRENT_TIMESTAMP(6),
  CONSTRAINT FK_Domain__Domain_DomainSources FOREIGN
     KEY (DomainId) REFERENCES Domain (id) ON DELETE
     CASCADE ON UPDATE NO ACTION,
  CONSTRAINT FK_DomainSources__Domain_DomainSources
     FOREIGN KEY (DomainSourcesId) REFERENCES
      DomainSources (id) ON DELETE NO ACTION ON UPDATE
     NO ACTION,
  primary key (DomainId, DomainSourcesId, date));

create  table DomainMaliciousChecks (
  id          smallint UNSIGNED  not null
     auto_increment,
  value       varchar(250) not null,
  score       smallint default 0 not null,
  primary key (id));

create  table Domain_DomainMaliciousChecks (
  DomainId                  int UNSIGNED  not null,
  DomainMaliciousChecksId smallint UNSIGNED  not null,
  CONSTRAINT FK_Domain__Domain_DomainMaliciousChecks
     FOREIGN KEY (DomainId) REFERENCES Domain (id) ON
     DELETE CASCADE ON UPDATE NO ACTION,
  CONSTRAINT
      FK_DomainMaliciousChecks__Domain_DomainMaliciousChecks
      FOREIGN KEY (DomainMaliciousChecksId) REFERENCES
       DomainMaliciousChecks (id) ON DELETE NO ACTION
     ON UPDATE NO ACTION,
  primary key (DomainId, DomainMaliciousChecksId));
```

```
create table EmailAddressDelete (
  id          int UNSIGNED not null auto_increment,
  idEmailAddress int UNSIGNED not null,
  hash        varchar(256) not null,
  CONSTRAINT FK_EmailAddress_EmailAddressDelete
     FOREIGN KEY (idEmailAddress) REFERENCES
     EmailAddress (id) ON DELETE CASCADE ON UPDATE NO
     ACTION,
  primary key (id));

create table EmailAddressSources (
  id          smallint UNSIGNED not null
     auto_increment,
  value varchar(100) not null,
  score       smallint default 0 not null,
  UNIQUE INDEX uq_EmailAddressSources_value (value
     DESC),
  primary key (id));
INSERT INTO EmailAddressSources(value) VALUES ("
    textAnalysis");
INSERT INTO EmailAddressSources(value) VALUES ("
    emlHeaders");
INSERT INTO EmailAddressSources(value) VALUES ("aHref"
    );
INSERT INTO EmailAddressSources(value) VALUES ("aText"
    );

create table EmailAddress_EmailAddressSources (
  EmailAddressId           int UNSIGNED not null,
  EmailAddressSourcesId    smallint UNSIGNED not null,
  date                     TIMESTAMP(6) NOT NULL
     DEFAULT CURRENT_TIMESTAMP(6),
  CONSTRAINT
     FK_EmailAddress__EmailAddress_EmailAddressSources
     FOREIGN KEY (EmailAddressId) REFERENCES
     EmailAddress (id) ON DELETE CASCADE ON UPDATE NO
     ACTION,
  CONSTRAINT
     FK_EmailAddressSources__EmailAddress_EmailAddressSources
     FOREIGN KEY (EmailAddressSourcesId) REFERENCES
     EmailAddressSources (id) ON DELETE NO ACTION ON
```

```
    UPDATE NO ACTION,
  primary key (EmailAddressId, EmailAddressSourcesId,
    date));

create table EmailAddressMaliciousChecks (
  id          smallint UNSIGNED not null
    auto_increment,
  value       varchar(250) not null,
  score       smallint default 0 not null,
  primary key (id));

create table EmailAddress_EmailAddressMaliciousChecks
    (
  EmailAddressId                  int UNSIGNED not null
    ,
  EmailAddressMaliciousChecksId smallint UNSIGNED not
     null,
  CONSTRAINT
    FK_EmailAddress__EmailAddress_EmailAddressMaliciousChecks
     FOREIGN KEY (EmailAddressId) REFERENCES
    EmailAddress (id) ON DELETE CASCADE ON UPDATE NO
    ACTION,
  CONSTRAINT
    FK_EmailAddressMC__EmailAddress_EmailAddressMC
    FOREIGN KEY (EmailAddressMaliciousChecksId)
    REFERENCES EmailAddressMaliciousChecks (id) ON
    DELETE NO ACTION ON UPDATE NO ACTION,
  primary key (EmailAddressId,
    EmailAddressMaliciousChecksId));

create table EmailAddress_href_value (
  href          int UNSIGNED not null,
  text           int UNSIGNED not null,
  CONSTRAINT
    FK_EmailAddress__EmailAddress_href_value_href
    FOREIGN KEY (href) REFERENCES EmailAddress (id)
    ON DELETE CASCADE ON UPDATE NO ACTION,
  CONSTRAINT
    FK_EmailAddress__EmailAddress_href_value_text
    FOREIGN KEY (text) REFERENCES EmailAddress (id)
    ON DELETE CASCADE ON UPDATE NO ACTION,
  primary key (href, text));
```

```
create table MessageData_IpAddress (
  MessageDataId    int UNSIGNED not null,
  IpAddressId      int UNSIGNED not null,
  CONSTRAINT FK_MessageData__MessageData_IpAddress
     FOREIGN KEY (MessageDataId) REFERENCES
     MessageData (id) ON DELETE CASCADE ON UPDATE NO
     ACTION,
  CONSTRAINT FK_IpAddress__MessageData_IpAddress
     FOREIGN KEY (IpAddressId) REFERENCES IpAddress (
     id) ON DELETE NO ACTION ON UPDATE NO ACTION,
  primary key (MessageDataId, IpAddressId));

create table MessageData_Domain (
  MessageDataId       int UNSIGNED not null,
  DomainId            int UNSIGNED not null,
  CONSTRAINT FK_MessageData__MessageData_Domain
     FOREIGN KEY (MessageDataId) REFERENCES
     MessageData (id) ON DELETE CASCADE ON UPDATE NO
     ACTION,
  CONSTRAINT FK_Domain__MessageData_Domain FOREIGN KEY
      (DomainId) REFERENCES Domain (id) ON DELETE NO
     ACTION ON UPDATE NO ACTION,
  primary key (MessageDataId, DomainId));

create table MessageData_EmailAddress (
  MessageDataId       int UNSIGNED not null,
  EmailAddressId      int UNSIGNED not null,
  CONSTRAINT FK_MessageData__MessageData_EmailAddress
     FOREIGN KEY (MessageDataId) REFERENCES
     MessageData (id) ON DELETE CASCADE ON UPDATE NO
     ACTION,
  CONSTRAINT FK_EmailAddress__MessageData_EmailAddress
      FOREIGN KEY (EmailAddressId) REFERENCES
     EmailAddress (id) ON DELETE NO ACTION ON UPDATE
     NO ACTION,
  primary key (MessageDataId, EmailAddressId));

create table MessageData_Url (
  MessageDataId     int UNSIGNED not null,
  UrlId             int UNSIGNED not null,
  CONSTRAINT FK_MessageData__MessageData_Url FOREIGN
     KEY (MessageDataId) REFERENCES MessageData (id)
```

```
    ON DELETE CASCADE ON UPDATE NO ACTION,
  CONSTRAINT FK_Url__MessageData_Url FOREIGN KEY (
      UrlId) REFERENCES Url (id) ON DELETE NO ACTION ON
       UPDATE NO ACTION,
  primary key (MessageDataId, UrlId));

create table MessageData_MessageHeaderValues (
  MessageDataId          int UNSIGNED not null,
  MessageHeaderValuesId int UNSIGNED not null,
  CONSTRAINT
      FK_MessageData__MessageData_MessageHeaderValues
      FOREIGN KEY (MessageDataId) REFERENCES
      MessageData (id) ON DELETE CASCADE ON UPDATE NO
      ACTION,
  CONSTRAINT
      FK_MessageHeaderValues__MessageData_MessageHeaderValues
       FOREIGN KEY (MessageHeaderValuesId) REFERENCES
      MessageHeaderValues (id) ON DELETE NO ACTION ON
      UPDATE NO ACTION,
  primary key (MessageDataId, MessageHeaderValuesId));




create table UrlSources (
  id          smallint UNSIGNED not null
     auto_increment,
  value       varchar(100) not null,
  score       smallint default 0 not null,
  UNIQUE INDEX uq_UrlSources_value (value DESC),
  primary key (id));
INSERT INTO UrlSources(value) VALUES ("textAnalysis");
INSERT INTO UrlSources(value) VALUES ("emlHeaders");
INSERT INTO UrlSources(value) VALUES ("aHref");
INSERT INTO UrlSources(value) VALUES ("aText");

create table Url_UrlSources (
  UrlId          int UNSIGNED not null,
  UrlSourcesId  smallint UNSIGNED not null,
  date           TIMESTAMP(6) NOT NULL DEFAULT
     CURRENT_TIMESTAMP(6),
  CONSTRAINT FK_Url__Url_UrlSources FOREIGN KEY (UrlId
      ) REFERENCES Url (id) ON DELETE CASCADE ON UPDATE
       NO ACTION,
```

```
  CONSTRAINT FK_UrlSources__Url_UrlSources FOREIGN KEY
      (UrlSourcesId) REFERENCES UrlSources (id) ON
     DELETE NO ACTION ON UPDATE NO ACTION,
  primary key (UrlId, UrlSourcesId, date));

create  table UrlMaliciousChecks (
  id          smallint UNSIGNED  not null
     auto_increment,
  value       varchar(250) not null,
  score       smallint default 0 not null,
  primary key (id));

create  table Url_UrlMaliciousChecks (
  UrlId                   int UNSIGNED  not null,
  UrlMaliciousChecksId  smallint UNSIGNED  not null,
  CONSTRAINT FK_Url__Url_UrlMaliciousChecks FOREIGN
     KEY (UrlId) REFERENCES Url (id) ON DELETE CASCADE
      ON UPDATE NO ACTION,
  CONSTRAINT
     FK_UrlMaliciousChecks__Url_UrlMaliciousChecks
     FOREIGN KEY (UrlMaliciousChecksId) REFERENCES
     UrlMaliciousChecks (id) ON DELETE NO ACTION ON
     UPDATE NO ACTION,
  primary key (UrlId, UrlMaliciousChecksId));

create  table Url_href_value (
  href          int UNSIGNED  not null,
  text          int UNSIGNED  not null,
  CONSTRAINT FK_Url__Url_href_value_href FOREIGN KEY (
     href) REFERENCES Url (id) ON DELETE CASCADE ON
     UPDATE NO ACTION,
  CONSTRAINT FK_Url__Url_href_value_text FOREIGN KEY (
     text) REFERENCES Url (id) ON DELETE CASCADE ON
     UPDATE NO ACTION,
  primary key (href, text));



create  table MessageHeaderValues_IpAddress (
  MessageHeaderValuesId      int UNSIGNED  not null,
  IpAddressId                int UNSIGNED  not null,
  CONSTRAINT
     FK_MessageHeaderValues__MessageHeaderValues_IpAddress
      FOREIGN KEY (MessageHeaderValuesId) REFERENCES
```

```
    MessageHeaderValues (id) ON DELETE CASCADE ON
    UPDATE NO ACTION,
  CONSTRAINT
    FK_IpAddress__MessageHeaderValues_IpAddress
    FOREIGN KEY (IpAddressId) REFERENCES IpAddress (
    id) ON DELETE NO ACTION ON UPDATE NO ACTION,
  primary key (MessageHeaderValuesId, IpAddressId));

create table MessageHeaderValues_Domain (
  MessageHeaderValuesId     int UNSIGNED not null,
  DomainId                  int UNSIGNED not null,
  CONSTRAINT
    FK_MessageHeaderValues__MessageHeaderValues_Domain
     FOREIGN KEY (MessageHeaderValuesId) REFERENCES
    MessageHeaderValues (id) ON DELETE CASCADE ON
    UPDATE NO ACTION,
  CONSTRAINT FK_Domain__MessageHeaderValues_Domain
    FOREIGN KEY (DomainId) REFERENCES Domain (id) ON
    DELETE NO ACTION ON UPDATE NO ACTION,
  primary key (MessageHeaderValuesId, DomainId));

create table MessageHeaderValues_Url (
  MessageHeaderValuesId     int UNSIGNED not null,
  UrlId                     int UNSIGNED not null,
  CONSTRAINT
    FK_MessageHeaderValues__MessageHeaderValues_Url
    FOREIGN KEY (MessageHeaderValuesId) REFERENCES
    MessageHeaderValues (id) ON DELETE CASCADE ON
    UPDATE NO ACTION,
  CONSTRAINT FK_Url__MessageHeaderValues_Url FOREIGN
    KEY (UrlId) REFERENCES Url (id) ON DELETE NO
    ACTION ON UPDATE NO ACTION,
  primary key (MessageHeaderValuesId, UrlId));

create table MessageHeaderValues_EmailAddress (
  MessageHeaderValuesId     int UNSIGNED not null,
  EmailAddressId            int UNSIGNED not null,
  CONSTRAINT
    FK_MessageHeaderValues__MessageHeaderValues_EmailAddress
     FOREIGN KEY (MessageHeaderValuesId) REFERENCES
    MessageHeaderValues (id) ON DELETE CASCADE ON
    UPDATE NO ACTION,
  CONSTRAINT
    FK_EmailAddress__MessageHeaderValues_EmailAddress
```

```
    FOREIGN KEY (EmailAddressId) REFERENCES
    EmailAddress (id) ON DELETE NO ACTION ON UPDATE
    NO ACTION,
  primary key (MessageHeaderValuesId, EmailAddressId))
    ;
```

Tabla B.7: Tablas de la información extraida y auxiliares

B.3. Vistas

```
CREATE OR REPLACE VIEW MessageText_view AS
SELECT        MessageData.hash as hash,
          MessageText.value as value,
          MessageData.score as score,
          MessageFormat.value as format,
          AnalyzedBy.value as analyzedBy,
          MessageStatus.value as status,
          MessageData.analysisTime as analysisTime,
          MessageData.addeddAt as addeddAt,
          MessageData.analyzedAt as analyzedAt,
          MessageData.updatedAt as updatedAt,
          COUNT(MessageData_MessageDataSources.
              MessageDataId) AS ocurrences
FROM MessageData
    INNER JOIN MessageFormat ON MessageData.format =
        MessageFormat.id
    INNER JOIN MessageStatus ON MessageData.status =
        MessageStatus.id
    LEFT JOIN AnalyzedBy ON MessageData.analyzedBy =
        AnalyzedBy.id
    INNER JOIN MessageText on MessageData.id =
        MessageText.id
    INNER JOIN MessageData_MessageDataSources ON
        MessageData.id =
        MessageData_MessageDataSources.MessageDataId
GROUP BY hash;
```

Vista B.8: Vista MessageText_view

```
CREATE OR REPLACE VIEW MessageData_view AS
SELECT        MessageData.id as id,
          MessageData.hash as hash,
```

```
        MessageData.score as score,
        MessageFormat.value as format,
        AnalyzedBy.value as analyzedBy,
        MessageStatus.value as status,
        MessageData.analysisTime as analysisTime,
        MessageData.addeddAt as addeddAt,
        MessageData.analyzedAt as analyzedAt,
        MessageData.updatedAt as updatedAt,
        COUNT(MessageData_MessageDataSources.
            MessageDataId) AS ocurrences
FROM MessageData
    INNER JOIN MessageFormat ON MessageData.format =
        MessageFormat.id
    INNER JOIN MessageStatus ON MessageData.status =
        MessageStatus.id
    LEFT JOIN AnalyzedBy ON MessageData.analyzedBy =
        AnalyzedBy.id
    INNER JOIN MessageData_MessageDataSources ON
        MessageData.id =
        MessageData_MessageDataSources.MessageDataId
GROUP BY hash;
```

Vista B.9: Vista MessageData_view

```
CREATE OR REPLACE VIEW IpAddress_view AS
SELECT  IpAddress.id as id,
        IpAddress.value as value,
        IpAddress.hash as hash,
        IpAddress.score as score,
        IpAddress.ipv4 as ipv4,
        COUNT(IpAddress_IpAddressSources.IpAddressId
            ) AS ocurrences
FROM IpAddress
    INNER JOIN IpAddress_IpAddressSources ON
        IpAddress_IpAddressSources.IpAddressId =
        IpAddress.id GROUP BY(IpAddress.value);
```

Vista B.10: Vista IpAddress_view

```
CREATE OR REPLACE VIEW Domain_view AS
SELECT  Domain.id as id,
        Domain.value as value,
        Domain.hash as hash,
```

```
        Domain.score as score,
        d.value as parentDomain,
        d.hash as parentDomainHash,
        COUNT(Domain_DomainSources.DomainId) AS
            ocurrences
FROM Domain
    LEFT JOIN Domain as d ON d.id = Domain.parentId
    INNER JOIN Domain_DomainSources ON
        Domain_DomainSources.DomainId = Domain.id
    GROUP BY(Domain.value);
```

Vista B.11: Vista Domain_view

```
CREATE OR REPLACE VIEW domain_ip AS
SELECT Domain.value as domain,
        Domain.hash as domainHash,
        ip.value as ip,
        ip.hash as ipHash,
        ip.score as score
FROM Domain
    LEFT JOIN Domain_IpAddress ON Domain.id =
        Domain_IpAddress.DomainId
    INNER JOIN IpAddress as ip ON ip.id =
        Domain_IpAddress.IpAddressId;
```

Vista B.12: Vista domain_ip

```
CREATE OR REPLACE VIEW subdomain AS
SELECT Domain.value as domain,
        Domain.hash as domainHash,
        d.value as subdomain,
        d.hash as subdomainHash,
        d.score as score,
        d.analyzedAt as analyzedAt
FROM Domain
    LEFT JOIN Domain as d on d.parentId = Domain.id;
```

Vista B.13: Vista subdomain

```
CREATE OR REPLACE VIEW IpAddress_MessageData AS
SELECT IpAddress.value as value,
    IpAddress.hash as IpAddressHash,
MessageData.hash as messageHash,
```

```
    MessageData.score as score,
      MessageFormat.value as format,
    MessageStatus.value as status,
      MessageData.addeddAt as addeddAt,
    MessageData.analyzedAt as analyzedAt
FROM IpAddress
    LEFT JOIN MessageData_IpAddress ON IpAddress.id =
        MessageData_IpAddress.IpAddressId
    INNER JOIN MessageData ON MessageData.id =
        MessageData_IpAddress.MessageDataId
      LEFT JOIN MessageStatus ON MessageData.status =
          MessageStatus.id
      LEFT JOIN MessageFormat ON MessageFormat.id =
          MessageData.format;
```

Vista B.14: Vista IpAddress_MessageData

```
CREATE OR REPLACE VIEW ip_domain AS
SELECT ip.value as ip,
       ip.hash as ipHash,
       Domain.value as domain,
       Domain.hash as domainHash,
       Domain.score as score,
       Domain.analyzedAt as analyzedAt
FROM IpAddress as ip
    LEFT JOIN Domain_IpAddress ON ip.id =
        Domain_IpAddress.IpAddressId
    INNER JOIN Domain ON Domain.id =
        Domain_IpAddress.DomainId;
```

Vista B.15: Vista ip_domain

```
CREATE OR REPLACE VIEW Domain_MessageData AS
SELECT Domain.value as value,
    Domain.hash as DomainHash,
  MessageData.hash as messageHash,
    MessageData.score as score,
      MessageFormat.value as format,
    MessageStatus.value as status,
      MessageData.addeddAt as addeddAt,
    MessageData.analyzedAt as analyzedAt
FROM Domain
    LEFT JOIN MessageData_Domain ON Domain.id =
        MessageData_Domain.DomainId
```

```
    INNER JOIN MessageData ON MessageData.id =
        MessageData_Domain.MessageDataId
    LEFT JOIN MessageStatus ON MessageData.status =
        MessageStatus.id
    LEFT JOIN MessageFormat ON MessageFormat.id =
        MessageData.format;
```

Vista B.16: Vista Domain_MessageData

```
CREATE OR REPLACE VIEW domain_url AS
SELECT  Domain.value as domain,
        Domain.hash as domainHash,
        Url.value as url,
        Url.hash as urlHash,
        Url.score as score,
        Url.analyzedAt as analyzedAt
FROM Domain
    INNER JOIN Url ON Domain.id = Url.domain;
```

Vista B.17: Vista domain_url

```
CREATE OR REPLACE VIEW domain_email AS
SELECT  Domain.value as domain,
        Domain.id as id,
        Domain.hash as domainHash,
        EmailAddress.value as email,
        EmailAddress.hash as emailAddressHash,
        EmailAddress.score as score,
        EmailAddress.analyzedAt as analyzedAt
FROM Domain
    INNER JOIN EmailAddress ON Domain.id =
        EmailAddress.domain;
```

Vista B.18: Vista domain_email

```
CREATE OR REPLACE VIEW EmailAddress_view AS
SELECT  EmailAddress.value as value,
        EmailAddress.hash as hash,
        EmailAddress.score as score,
        Domain.value as domain,
        Domain.hash as domainHash,
        COUNT(EmailAddress_EmailAddressSources.
            EmailAddressId) AS ocurrences
```

```
FROM EmailAddress
    LEFT JOIN Domain ON EmailAddress.domain = Domain
        .id
    INNER JOIN EmailAddress_EmailAddressSources ON
        EmailAddress_EmailAddressSources.
        EmailAddressId = EmailAddress.id GROUP BY(
        EmailAddress.value);
```

Vista B.19: Vista EmailAddress_view

```
CREATE OR REPLACE VIEW EmailAddress_MessageData AS
SELECT EmailAddress.value as value,
     EmailAddress.hash as emailAddressHash,
 MessageData.hash as messageHash,
   MessageData.score as score,
     MessageFormat.value as format,
   MessageStatus.value as status,
     MessageData.addeddAt as addeddAt,
   MessageData.analyzedAt as analyzedAt
FROM EmailAddress
    LEFT JOIN MessageData_EmailAddress ON EmailAddress
        .id = MessageData_EmailAddress.EmailAddressId
    INNER JOIN MessageData ON MessageData.id =
        MessageData_EmailAddress.MessageDataId
      LEFT JOIN MessageStatus ON MessageData.status =
          MessageStatus.id
      LEFT JOIN MessageFormat ON MessageFormat.id =
          MessageData.format;
```

Vista B.20: Vista EmailAddress_MessageData

```
CREATE OR REPLACE VIEW Url_view AS
SELECT  Url.value as value,
        Url.id as id,
        Url.hash as hash,
        Url.score as score,
        Domain.value as domain,
        Domain.hash as domainHash,
       COUNT(Url_UrlSources.UrlId) AS ocurrences
FROM Url
    LEFT JOIN Domain ON Url.domain = Domain.id
    INNER JOIN Url_UrlSources ON Url_UrlSources.
```

```
    UrlId = Url.id GROUP BY(Url.value);
```

Vista B.21: Vista Url_view

```
CREATE OR REPLACE VIEW Url_MessageData AS
SELECT Url.value as value,
     Url.hash as UrlHash,
  MessageData.hash as messageHash,
    MessageData.score as score,
      MessageFormat.value as format,
    MessageStatus.value as status,
      MessageData.addeddAt as addeddAt,
    MessageData.analyzedAt as analyzedAt
FROM Url
    LEFT JOIN MessageData_Url ON Url.id =
       MessageData_Url.UrlId
    INNER JOIN MessageData ON MessageData.id =
       MessageData_Url.MessageDataId
      LEFT JOIN MessageStatus ON MessageData.status =
         MessageStatus.id
      LEFT JOIN MessageFormat ON MessageFormat.id =
         MessageData.format;
```

Vista B.22: Vista Url_MessageData

```
CREATE OR REPLACE VIEW MessageData_IpAddress_view AS
SELECT  MessageData.hash as hash,
      IpAddress.value AS ip,
          IpAddress.hash as ipHash,
          IpAddress.score as score,
          IpAddress.ipv4 as ipv4
 FROM MessageData
    LEFT JOIN MessageData_IpAddress ON MessageData.id
       = MessageData_IpAddress.MessageDataId
      INNER JOIN IpAddress ON MessageData_IpAddress.
         IpAddressId = IpAddress.id;
```

Vista B.23: Vista MessageData_IpAddress_view

```
CREATE OR REPLACE VIEW MessageData_EmailAddress_view
    AS
SELECT  MessageData.hash as hash,
      EmailAddress.value AS email,
```

```
        EmailAddress.hash as emailAddressHash,
        EmailAddress.score as score,
        Domain.value AS domain,
        Domain.hash as domainHash
FROM MessageData
  LEFT JOIN MessageData_EmailAddress ON MessageData
      .id = MessageData_EmailAddress.MessageDataId
    INNER JOIN EmailAddress ON
        MessageData_EmailAddress.EmailAddressId =
        EmailAddress.id
    INNER JOIN Domain ON EmailAddress.domain =
        Domain.id;
```

Vista B.24: Vista MessageData_EmailAddress_view

```
CREATE OR REPLACE VIEW MessageData_Domain_view AS
SELECT  MessageData.hash as hash,
    Domain.value AS domain,
        Domain.hash as domainHash,
        Domain.score as score
FROM MessageData
  LEFT JOIN MessageData_Domain ON MessageData.id =
      MessageData_Domain.MessageDataId
    INNER JOIN Domain ON MessageData_Domain.
        DomainId = Domain.id;
```

Vista B.25: Vista MessageData_Domain_view

```
CREATE OR REPLACE VIEW MessageData_Url_view AS
SELECT  MessageData.hash as hash,
     Url.value AS url,
        Url.hash as urlHash,
        Url.score as score
FROM MessageData
  LEFT JOIN MessageData_Url ON MessageData.id =
      MessageData_Url.MessageDataId
    INNER JOIN Url ON MessageData_Url.UrlId = Url.
        id;
```

Vista B.26: Vista MessageData_Url_view

```
CREATE OR REPLACE VIEW MessageData_child_view AS
SELECT  MessageData.hash as hash,
```

```
      mp.hash as childHash,
          mp.score as score,
          MessageFormat.value as format,
      MessageStatus.value as status,
          MessageData.addeddAt as addeddAt,
      MessageData.analyzedAt as analyzedAt
FROM MessageData
  LEFT JOIN ChildMessages ON MessageData.id =
    ChildMessages.parentId
    INNER JOIN MessageData as mp ON mp.id =
      ChildMessages.childId
    LEFT JOIN MessageStatus ON mp.status =
      MessageStatus.id
    LEFT JOIN MessageFormat ON mp.format =
      MessageFormat.id;
```

Vista B.27: Vista MessageData_child_view

```
CREATE OR REPLACE VIEW MessageData_parent_view AS
SELECT MessageData.hash as hash,
      mp.hash as parentHash,
          mp.score as score,
          MessageFormat.value as format,
      MessageStatus.value as status,
          MessageData.addeddAt as addeddAt,
      MessageData.analyzedAt as analyzedAt
FROM MessageData
  LEFT JOIN ChildMessages ON MessageData.id =
    ChildMessages.childId
    INNER JOIN MessageData as mp ON mp.id =
      ChildMessages.parentId
    LEFT JOIN MessageStatus ON mp.status =
      MessageStatus.id
    LEFT JOIN MessageFormat ON mp.format =
      MessageFormat.id;
```

Vista B.28: Vista MessageData_parent_view

=======

Apéndice A

Documentos

A.1. Diagrama EML

```
Return-Path: <SRS0=v5D8h6=TB=rankgator.live=
    kelly@untroubled.org>
Delivered-To: bruce@home.untroubled.org
Received: from rankgator.live (rdns.bitacel.com
    [194.34.104.232])
  by pt02.futurequest.net ([69.5.6.173])
  with ESMTP via TCP; 01 May 2019 14:41:04 -0000
Message-ID: <xat683W4TM32ruh.ilcIL221JGS1tho@rankgator
    .live>
From: "Arctic Pain Relief" <kelly@rankgator.live>
Date: Wed, 01 May 2019 10:09:17 -0400
MIME-Version: 1.0
Subject: Are  you -part -of— the— 'worlds- biggest
    arthritis cover-up?
To: "bruce@untroubled.org" <bruce@untroubled.org>
Content-type: multipart/alternative; boundary="---=
    _Part_1426_87C624P49.F740FP9";
Content-Length: 1606

-----=_Part_1426_87C624P49.F740FP9
Content-Type: text/plain; format=flowed; charset="UTF
    -8"
Content-Transfer-Encoding: 7bit

Are you part of the 'worlds biggest arthritis cover-up?
```

```
http://efs5I69IX4S3elw.rankgator.live/208
   fdcae990695eb0665b0903677f315_8da8c95c-0101030c0001
   /1/

Are you part of the 'worlds biggest arthritis cover-up?

------=_Part_1426_87C624P49.F740FP9
Content-Type: text/html; charset=us-ascii
Content-Transfer-Encoding: 7bit

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
   Transitional//EN">
<html>
  <head>
    <title></title>
  </head>
  <body>
    <table>
      <tr>
        <td colspan='2' align='center' valign='middle'
            class='preview-mid'>
          <br>
          <center>
            <a href="http://sdvPL9B7FL5Hwfm.rankgator.
               live/208
               fdcae990695eb0665b0903677f315_8da8c95c
               -0101030c0001/1/"><img src="http://
               ifq7BTAN28Q1vfo.rankgator.live/images
               //4b128133049378e37097925975e252e1"
               border="0" alt=""></a>
          </center>
          <div align="center">
###STUFF##
            <font face="Verdana, Arial, Helvetica,
               sans-serif" size="1"><br>
            If you'd prefer not to receive future
               emails, <a href="http://pkvG0P5I93MHwgd
               .rankgator.live/208
               abda8ac8695eb0665b0903678f315_8da8c95c
               -0101030c0001/1/"><font color
               ="#666666">Unsubscribe Here</font></a
               >.<br>
            616 Corporate Way, | Suite 2-5335 Valley
               Cottage | NY NY 10989</font>
```

```
            </div>
          </td>
        </tr>
      </table>
    </body>
</html>

<center>



<img src="http://pmsG8XP16WP5bqb.rankgator.live
    /2082301ea94695eb0665b090365f030b_8da8c95c-0101030
    c0001/2/">


%\chapter{Bibliografía}

%\input{capitulos/01_Introduccion}
%
%\input{capitulos/02_EspecificacionRequisitos}
%
%\input{capitulos/03_Planificacion}
%
%\input{capitulos/04_Analisis}
%
%\input{capitulos/05_Diseno}
%
%\input{capitulos/06_Implementacion}
%
%\input{capitulos/07_Pruebas}
%
%\input{capitulos/08_Conclusiones}
%
%%chapter{Conclusiones y Trabajos Futuros}
%
%
%%nocite{*}
\bibliography{bibliografia/bibliografia}\
    addcontentsline{toc}{chapter}{Bibliografía}
%\bibliography{bibliografia}\addcontentsline{toc}{
    chapter}{Bibliografía}
\bibliographystyle{unsrt}
%\printbibliography
```