Seguridad de las Tecnologías de la Información y de las Comunicaciones - MISTIC

TOR Hidden Service Discovery

Tutor: Guillermo Navarro Arribas
Alumno: David Fuentes Iglesias
Universidad: Universitat Oberta de Catalunya (UOC)

Abstract

El presente documento describe el funcionamiento de Tor y los servicios ocultos que esta aloja, y plantea la dificultad actual de la realización de descubrimiento automático de estos últimos. Para ello, se analizan y comparan múltiples posibles soluciones, y se implementa la que se considera la más adecuada.

En el proyecto se analiza cómo se ha implementado el crawler, que recursos son necesarios y que riesgos se corren en la implementación de un crawler para una darknet como es Tor.

This document describes the functionality of Tor and its hidden services, and raises the current difficulty in the realization of automatic discovery of them. To do this, multiple possible solutions are analyzed and compared, choosing the most appropriate solution for it's implementation.

The project analyzes how the crawler has been implemented, what resources are needed and what risks are taken in the implementation of a crawler for a darknet such as Tor.

Contenido

Índice de ilustraciones

1. Introducción

El proyecto planteado pretende estudiar el funcionamiento de la red TOR, y de los servicios que aloja, y así poder aplicar una técnica de descubrimiento automático de servicios ocultos.

En primer lugar, cabe definir qué es TOR. TOR es una implementación del “onion routing” o enrutamiento cebolla, que en sus principios estuvo financiado por la marina estadounidense, que fue creado con el objetivo de proteger las comunicaciones gubernamentales. Ahora se encuentra financiado por múltiples sponsors[1].

TOR o The Onion Router por sus siglas en inglés, es un proyecto que permite el desarrollo de una red superpuesta sobre Internet, y que permite la navegación anónima y segura de sus usuarios gracias al protocolo que implementa.

Este protocolo permite anonimizar y securizar las conexiones a servicios, tanto propios de la red de TOR (servicios ocultos, definidos en el apartado 2.3. Servicios ocultos), como de Internet, mediante el uso de relays o nodos que usuarios de todo el planeta ofrecen desinteresadamente.

Los servicios alojados en TOR tienen la extensión “onion” o cebolla en inglés, esto es debido al protocolo usado por TOR, que cifra las comunicaciones por capas, enrutando cada petición por un camino aleatorio o circuito, haciendo que las comunicaciones sean anónimas y seguras.

Dentro de la red, los servicios ocultos solo pueden ser accedidos si se sabe la dirección a la que se quiere acceder. Esto es debido a que en TOR no existe el concepto de DNS, sino que trabaja en base a claves públicas y privadas, y esto es lo que hace que los servicios no sean accesibles de la forma habitual, haciendo difícil crear servicios como buscadores o indexadores de contenidos. Los servicios ocultos, como se verá más adelante, se generan a partir de la clave privada del nodo que los aloja, por lo que realizar búsquedas por fuerza bruta en base a los nombres de los servicios se vuelve más complicado.

Cabe destacar que TOR es lo que se considera una darknet, es decir, una red dentro de internet de la cual no se indexan los contenidos, y a la que solo se puede acceder mediante software específico, que en este caso es TOR Browser, un Firefox modificado para su uso con TOR.

En el proyecto se busca profundizar en los conocimientos sobre el protocolo de TOR y sus servicios, analizar los mecanismos actuales de la búsqueda automatizada de servicios TOR ocultos, y finalmente, implementar un prototipo capaz de realizar el descubrimiento de este tipo de servicios de manera automatizada, basándose para ello en las técnicas comentadas anteriormente.

1.1. Objetivos

El objetivo principal del proyecto es el de estudiar, analizar y finalmente implementar una prueba de concepto de un método de descubrimiento automático de servicios ocultos, y para eso, se han definido varios objetivos principales.

[1] "Sponsors - Tor Project." Se consultó el diciembre 22, 2017. https://www.torproject.org/about/sponsors.html.en.

Los objetivos a conseguir durante la realización del proyecto son los siguientes:

- Estudiar la arquitectura y el funcionamiento de la red TOR y los servicios que aloja.
- Estudiar las técnicas de detección de servicios ocultos actuales.
- Realizar un análisis comparativo de las técnicas de detección, y concluir con la más adecuada para el proyecto.
- Implementar una prueba de concepto con la técnica seleccionada.

1.2. Metodología

Para alcanzar los objetivos anteriores, se van a definir tres fases, en las cuales se irán completando los objetivos especificados.
En la primera fase se procederá a una recogida de información, y se realizará una formación sobre la arquitectura y el protocolo de comunicaciones de Tor utilizando diversas fuentes.

La segunda fase consiste en realizar un análisis de las técnicas utilizadas hasta la fecha para proyectos existentes de descubrimiento de servicios en redes Tor, enfatizando en su funcionamiento y eficiencia, y finalmente se realizará una comparativa entre las técnicas analizadas.

En la última fase se procederá a implementar una prueba de concepto de uno de los métodos de descubrimiento de servicios en las redes Tor analizado anteriormente.

Finalmente se obtendrán las conclusiones sobre el desarrollo del proyecto, y las mejoras y/o correcciones realizadas sobre la técnica seleccionada si las hubiera.

1.3. Tareas y planificación

Para definir las tareas a realizar, las relaciones, el orden y la estimación de tiempos se ha definido un diagrama de Gantt, el cual se adjunta junto con el documento actual. La carpeta contiene el diagrama de Gantt generado, junto con el mismo exportado a varios formatos para facilitar su visualización.

2. TOR y los servicios ocultos

TOR es una red privada, que ofrece anonimato y seguridad en las comunicaciones a sus usuarios. Esta red consta de una capa de anonimización tanto para conexiones a servicios dentro como fuera de la red de Tor. Esta funcionalidad es posible gracias a que TOR está implementado sobre el protocolo de “onion routing” o enrutamiento cebolla, que se define en el apartado 2.2. Protocolo de TOR, The Onion Routing.

Para entender lo que es TOR (Tor Project, s.f.) y cómo funciona, primero hay que conocer el concepto de las darknets, su funcionamiento y finalidad.

2.1. Surface Web, Deep Web y Dark Web

La red tal y como la conocemos hoy en día está compuesta de diferentes niveles, que dividiremos en la clear web, también llamada surface web, la deep web y finalmente la dark web.

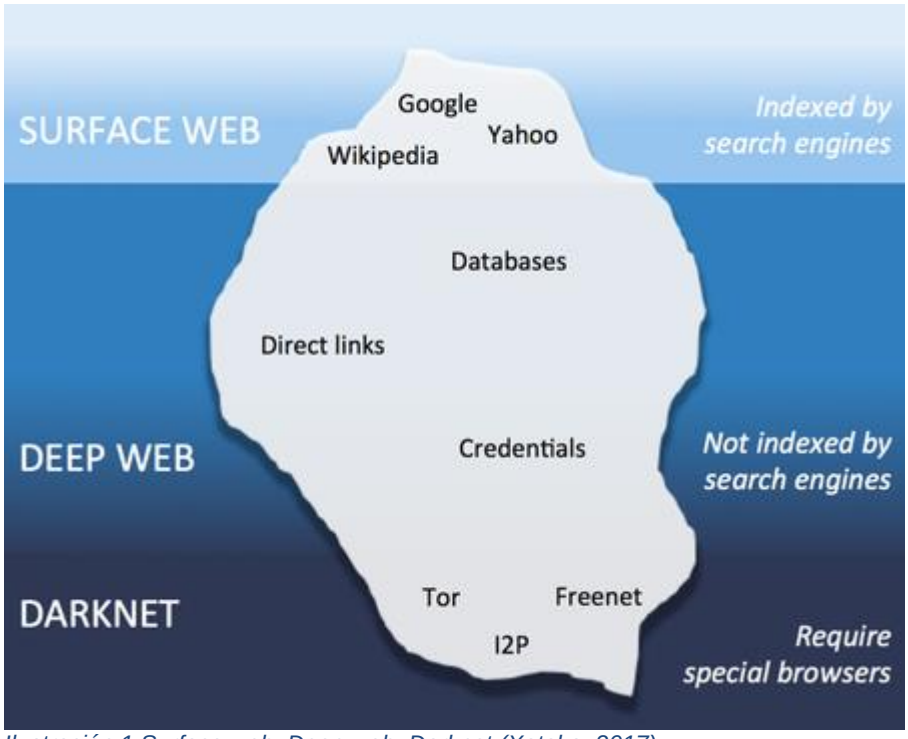

Ilustración 1 Surface web, Deep web, Darknet (Xataka, 2017)

La siguiente ilustración describe la división entre niveles, reflejando como claramente, existen más contenidos en los niveles inferiores que en la surface web, que como veremos, es el que la mayoría de usuarios utilizan en el día a día.

La **Surface Web** es parte de la red que todos conocemos y se encuentra indexada en la mayoría de buscadores que usamos en nuestro día a día, como pueden ser Google o Bing.
Es todo el contenido accesible fácilmente mediante buscadores, que se encuentra en la red de manera pública. En este nivel se engloban webs como blogs, Wikis, webs de noticias, etc. Al ser este el nivel de la red más accesible, también es el más controlado.

La **Deep Web** está formada por todo el contenido no indexado, o indexable de la red. Toda la información de este nivel no es accesible de forma pública, y engloba todo tipo de contenidos, como correos privados, contenidos almacenados en cuentas de almacenamiento como pueden ser Dropbox o Google Drive, perfiles privados de redes sociales, páginas dinámicas, webs con disallow en el robots.txt, etc.

Finalmente está la **Dark Web.** Este nivel engloba todas las redes superpuestas sobre Internet que necesitan de un software específico para ser accedidas. A cada una de estas redes superpuestas se le denomina Darknet**.**
Una **Darknet** es una red privada dentro de Internet a la que solo se puede acceder mediante un software específico. Dentro de las darknets las más destacadas son TOR, I2P y FreeNET, entre otras, y cada una de ellas ofrece un conjunto de funcionalidades que no se pueden encontrar en la World Wide Web.

TOR es la Darknet más conocida, y ofrece la ventaja de dar anonimato a sus usuarios, además de cifrar las comunicaciones utilizado el protocolo “Onion Routing”.

Para acceder a esta red, el usuario deberá utilizar un Onion Proxy(OP), que permite enrutar los datos a la red de Tor, adaptando los contenidos al protocolo the Onion Routing, del que se hablará a continuación.

Tor Project también ofrece un software específico más amigable para usuarios no avanzados llamado Tor Browser, que consiste en un Mozilla Firefox modificado que cuenta con un proxy que permite la entrada a la darknet, y que cuenta con la configuración adecuada para proteger la privacidad del usuario.

Una vez definido el concepto de darknet, y sabiendo que TOR no es la única que existe en la Dark web, se definirá el protocolo y las ventajas que ofrece con respecto a otras darknets.

2.2. Protocolo de TOR, The Onion Routing

TOR, como ya se ha mencionado, es una implementación del protocolo “onion routing”[2] o enrutamiento cebolla. Entre las ventajas del uso de este protocolo destacan la anonimización de las comunicaciones, y la securización de estas. Estas funcionalidades son posibles gracias a la arquitectura que se utiliza en este protocolo.

TOR está formado por nodos. Cada uno de estos nodos consta de ciertas características que hacen que dicho nodo ofrezca una funcionalidad u otra dentro de la red.

Según las características de cada nodo se pueden diferenciar en varios tipos, pero los más destacados son los siguientes:

- **Authority Directory relay:** Estos nodos conforman las autoridades de directorio, que son las encargadas de mantener y distribuir la lista del resto de nodos de la red de tor.
- **HSDir relay:** Estos nodos tienen información sobre cómo acceder a ciertos servicios ocultos de la red de TOR.
- **Guard relay**: Estos nodos conforman los nodos de entrada de la red TOR.
- **Middle relay:** Nodos verificados de la red de tor que se encargan simplemente de enrutar las comunicaciones que reciben.
- **Exit relay:** Nodos de la red de TOR que se encargan de redirigir el tráfico del nodo cliente a su destino.

2.2.1. Creación de un circuito

Con estos nodos, TOR es capaz de crear un circuito, que enruta el tráfico y anonimiza las conexiones. Por defecto, TOR utiliza una longitud de circuito de tres nodos por circuito, un nodo de entrada, un nodo intermedio, y un nodo de salida, aunque esto es configurable, y el circuito se puede extender, con la consecuencia de que la conexión será más lenta, ya que necesitará de más capas de encriptación.

La comunicación entre el nodo cliente y los nodos seleccionados del circuito se puede observar en la siguiente imagen en la cual vemos como el mensaje original se encripta por capas.

[2] "Protocol-level Hidden Server Discovery - Computer Science - UMass ...." Se consultó el diciembre 22, 2017. http://www.cs.uml.edu/~xinwenfu/paper/HiddenServer.pdf.

Los cells o cédulas mencionadas en las comunicaciones a continuación, constituyen la unidad mínima de comunicaciones del protocolo de Tor, y se diferencian entre cédulas de control y de transmisión que definen las acciones a realizar por los nodos que las reciben.

Este proceso se realiza recogiendo, en una primera instancia, las claves públicas de cada uno de los nodos que se van a utilizar en el circuito. Para esto se descarga la información de todos los onion routers de la red desde el Authority Direcotry Relay, que incluye como ya hemos dicho, la clave pública de dichos nodos.

Una vez recogida la información necesaria se procede a generar un nuevo circuito para la conexión. Para ello se realizan los siguientes pasos:

1. El cliente selecciona un router de salida que soporte el protocolo del stream que enviará en cliente.
2. El cliente selecciona un router de entrada o entry guard de la lista descargada con anterioridad.
3. Se selecciona el router intermedio.

2.2.2. Protocolo de comunicaciones en el circuito

Una vez generado el circuito, se procede a iniciar las comunicaciones entre el cliente y el destino.

El primer paso consiste en crear una conexión segura con el router de entrada o entry guard. Para ello, el nodo del cliente establece una conexión TLS con el router de entrada, y envía una cédula de tipo CREATE_CELL que indica el comienzo de creación de un circuito.

Cuando el nodo de entrada responde, se inicia un protocolo para negociar una clave simétrica común entre ambos nodos, que llamaremos $K_1 = g^{xy}$. Para ello se utilizan las claves públicas de ambos nodos.

El protocolo mencionado es Diffie-Hellman, que consiste en un protocolo de establecimiento de claves entre dos entidades mediante un canal seguro, que permite la generación de una clave simétrica compartida a partir de dos pares de claves asimétricas, una por cada entidad.

A partir de la clave común generada, se generan dos claves simétricas, una para la transmisión, que pasaremos a llamar K_{f1}, y una para la recepción, que llamaremos K_{b1}.

Este proceso se realiza para cada nodo del circuito, obteniendo al final de este una conexión tls entre los nodos, y los siguientes parámetros:

- Claves simétricas de transmisión: K_{f1}, K_{f2}, K_{f3}
- Claves simétricas de recepción: K_{b1}, K_{b2}, K_{b3}

Una vez establecidas las claves de transmisión y recepción, se comienza el protocolo de comunicación enviando una cédula de tipo RELAY_COMMAND_BEGIN al nodo de salida del circuito. La información contenida en el paquete está encriptada con las claves de transmisión definidas anteriormente, y contiene la IP y el puerto a los cuales se quiere conectar.

Ilustración 2 Capas de encriptación en la recepción

Cada nodo quita su capa de encriptación mediante su clave compartida a medida que se avanza por el circuito, hasta llegar al nodo de salida, el cual quita la última capa de encriptación, desvelando la IP y el puerto de destino.

Con esta información el nodo de salida realiza una conexión TCP con el servidor de destino y devuelve como respuesta una cédula de tipo RELAY_COMMAND_CONNECTED. A esta respuesta se le irán añadiendo capas de encriptación a medida que se avance por el circuito hasta el nodo del cliente, mediante las claves de recepción calculadas en el apartado anterior.

Ilustración 3 Capas de encriptación en el envío

En la siguiente imagen se puede observar el protocolo de comunicaciones en un circuito:

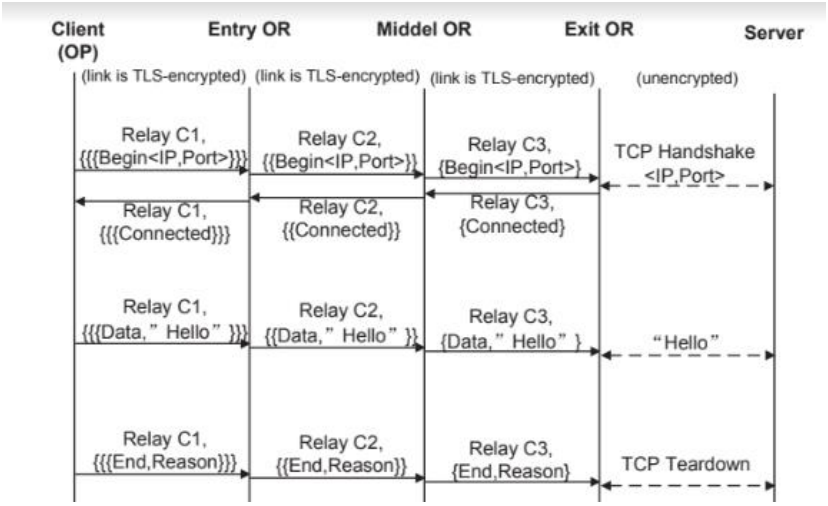

Ilustración 4 Workflow de comunicaciones de un circuito (Ling, Luo, Wu, & Fu, 2013)

Este tipo de funcionamiento es llamado funcionamiento out-proxy, ya que se considera que, una vez llegado al nodo de salida, este redirecciona la petición fuera de la red de TOR.

Las llamadas in-proxy, funcionan de una manera similar, con la diferencia de que la totalidad de las comunicaciones se realizan dentro de la red de Tor, sin llegar a salir de esta, y de manera totalmente anónima, ya que, al realizarse todas las conexiones mediante circuitos, los servidores de la red de Tor son anónimos. A continuación se definen los servicios ocultos, que son los servicios a los que hacer referencia el modo in-proxy.

2.3. Servicios ocultos

Al igual que TOR nos permite anonimizar conexiones a diferentes servicios de la clear web, como pueden ser el correo, una página web, etc. también existen servicios que se alojan dentro de la propia red de TOR. A estos servicios se les denomina Hidden Services[3], y como su propio nombre indica, se considera que estos servicios están ocultos dentro de la red de TOR. Esto es debido a la forma en la que se realizan las comunicaciones entre

[3] "Transparent Discovery of Hidden Service - J-Stage." Se consultó el diciembre 22, 2017. https://www.jstage.jst.go.jp/article/transinf/E99.D/11/E99.D_2016EDL8100/_article/-char/ja.

cliente y servidor, ya que al contrario que en la clear web, tanto el cliente como el servicio quedan ocultos gracias a las capacidades de anonimización de la red de TOR.

Debido a funcionamiento de la red TOR, los servicios alojados solo puedan ser accedidos por la gente que conoce la url a la que se quiere acceder, ya que, al realizarse las comunicaciones mediante circuitos, tanto el cliente como el servidor que ofrece el servicio son totalmente anónimos, y es necesario que el propio servicio exponga su dirección para poder ser accedido.

El acceso a servicios ocultos se considera in-proxy, ya que las comunicaciones realizadas nunca llegan a salir de la red de TOR.

Para poder entender el funcionamiento de la creación de un servicio oculto, y de la comunicación de los clientes con este, es necesario definir los elementos que toman parte en el protocolo:

Introductory point (IPO): Los IPOs son nodos publicados en la red Tor que contienen descriptores de los servicios ocultos alojados en la red. En el protocolo de conexión con los servicios ocultos realizan el papel de intermediarios entre los servicios ocultos y los clientes, redirigiendo información entre los nodos para poder realizar las comunicaciones.

Rendezbous point (RPO): Punto de encuentro entre el servidor oculto y el cliente, este nodo es seleccionado por el propio cliente y queda acordado con el servidor en la conexión al IPO. Ambos nodos realizan una conexión mediante un circuito hacia el RPO, el cual hace de puente entre ambos para permitir su comunicación.

Hidden server: Un servidor oculto que ofrece servicios TCP como aplicaciones web o servidores IRC.

Las comunicaciones necesarias para realizar la conexión a un servicio oculto comienzan en el propio servicio, el cual es el encargado de darse a conocer. Para este fin, el servicio oculto selecciona tres nodos de la red de tor de forma aleatoria, y crea los circuitos necesarios para conectarse a ellos de manera anónima. Estos nodos son los definidos anteriormente como IPOs, y se encargan de recibir todas las peticiones de los clientes y enrutarlas al servicio oculto.
La conexión entre el servidor y cada uno de los IPOs comienza cuando el servidor envía una cédula de tipo RELAY_COMMAND_ESTABLISH_INTRO, tras lo cual, si todo ha ido correctamente, el IPO responderá con un RELAY_COMMAND_INTRO_ESTABLISHED para comunicar que la información se ha recibido correctamente y se ha creado el punto de introducción al servicio oculto de forma satisfactoria.

De esta forma el servicio ya es accesible para los clientes, pero los clientes aún no tienen forma de llegar hasta él, ya que el servicio no está publicado en la red de TOR. Para esto, el servicio crea un fichero conocido como “Hidden Service Descriptor” (HSD) que no es más que un fichero que contiene la dirección “onion” del servicio, su clave pública y el listado de “Puntos introductorios” mencionados anteriormente.

Este descriptor se envía a uno de los HSDir, el cual se encarga de registrar el servicio y procesar las peticiones de los clientes, devolviéndoles el descriptor del servicio solicitado para que puedan conectarse a él.

Con la intención de no saturar los servidores HSDir, la red de Tor cuenta con un sistema que permite que los descriptores se almacenen de forma equivalente entre los servidores disponibles en la red. Este sistema se basa en el fingerprint de los HSDir, que son el hash

de la clave pública del nodo, y en el identificador de descriptor, que es un cálculo realizado para cada descriptor.

Realizando una comparación de los dos valores, el primero de los HSDir con un valor superior al identificador del descriptor será el seleccionado para guardar dicho descriptor. A su vez, los dos descriptores siguientes también guardarán dicho descriptor a modo de copia. Este método se define como rango de fingerprints, y se visualiza como un circulo sobre el cual se enumeran los HSDirs en base al valor de su fingerprint.

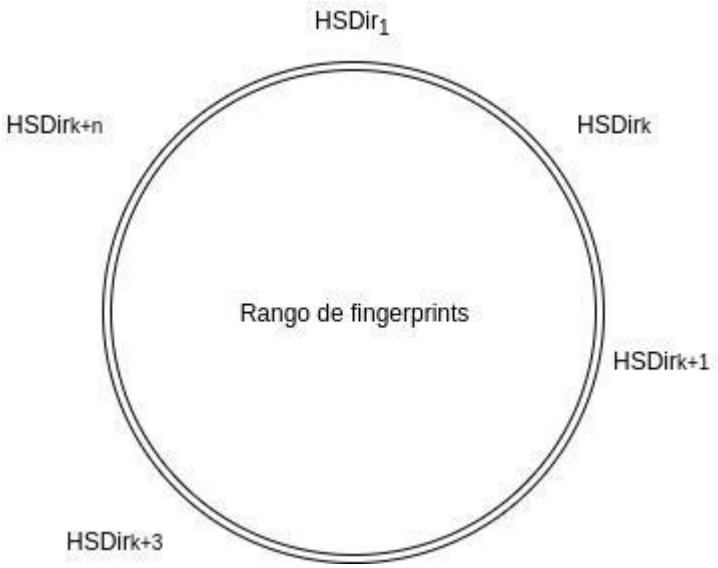

Ilustración 5 Rango de fingerprints (Biryukov, Pustogarov, & Weinmann, 2013)

Así, el servicio oculto creará un circuito al HSDir correspondiente para conservar su anonimato, y enviará su descriptor para que sea almacenado.

Una vez el servicio está publicado, un usuario puede conectarse a él únicamente si sabe su dirección “onion”. Asumiendo que el cliente tiene dicha dirección, podrá obtener el HSD (Hidden Service Descriptor) con los puntos introductorios y la clave pública del servicio desde el servidor HSDir.

Por tanto, una vez el cliente realiza la conexión hacia el HSDir y pide el descriptor del servicio oculta al que se quiere conectar, este selecciona un nodo de la red de forma aleatoria que hará las veces de punto de encuentro o “rendezvous point” RPO. Para ello envía al nodo seleccionado una cédula de tipo RELAY_COMMAND_STABLISH_RENDEZVOUS, junto con una cookie de identificación de la conexión. El nodo seleccionado devolverá como respuesta una cédula de tipo RELAY_COMMAND_RENDEZVOUS_STABLISHED si todo funciona correctamente.

Una vez definido el RPO, el cliente realiza crea un circuito a uno de los IPOs contenidos en el descriptor del servicio, enviándole una cédula de tipo RELAY_COMMAND_INTRODUCE1, con la información sobre el RPO seleccionado y la cookie de identificación de la conexión, y parte de los datos (g^x) necesarios para generar una clave compartida mediante el algoritmo Diffie-Hellman entre el cliente y el servicio oculto.

Una vez el IPO recibe la petición de conexión, devuelve un RELAY_COMMAND_INTRODUCE_ACK para verificar que se ha recibido la petición. En ese momento, el cliente cierra el circuito con el IPO.El IPO a continuación empaqueta el

paquete recibido (RELAY_COMMAND_INTRODUCCE1), como RELAY_COMMAND_INTRODUCCE2, incluyendo la información que contenía.

Con esta información, el servicio oculto genera la parte del algoritmo de Diffie-Hellman restante (g^y), y genera una clave compartida $K = g^{xy}$, que se utilizará para encriptar la conexión punto-a-punto entre el cliente y el servidor.

El servicio oculto crea un circuito hasta el RPO con una cédula de tipo RELAY_COMMAND_RENDEZVOUS1, incluyendo la clave (g^y), el hash de la clave creada ($H(K=g^{xy})$, y la cookie de identificación de la comunicación. El RPO recibe la petición y compara la cookie recibida con las del cliente. Si coinciden, el RPO elimina la cookie del paquete y lo reenvía al cliente.

El cliente, con dicha información, genera la clave común para ambos ($K = g^{xy}$) y verifica el hash de la clave generada con el hash recibido en el paquete. Una vez generadas las claves, el cliente envía una cédula de tipo RELAY_COMMAND_BEGIN que inicializa el envío de datos por el circuito generado.

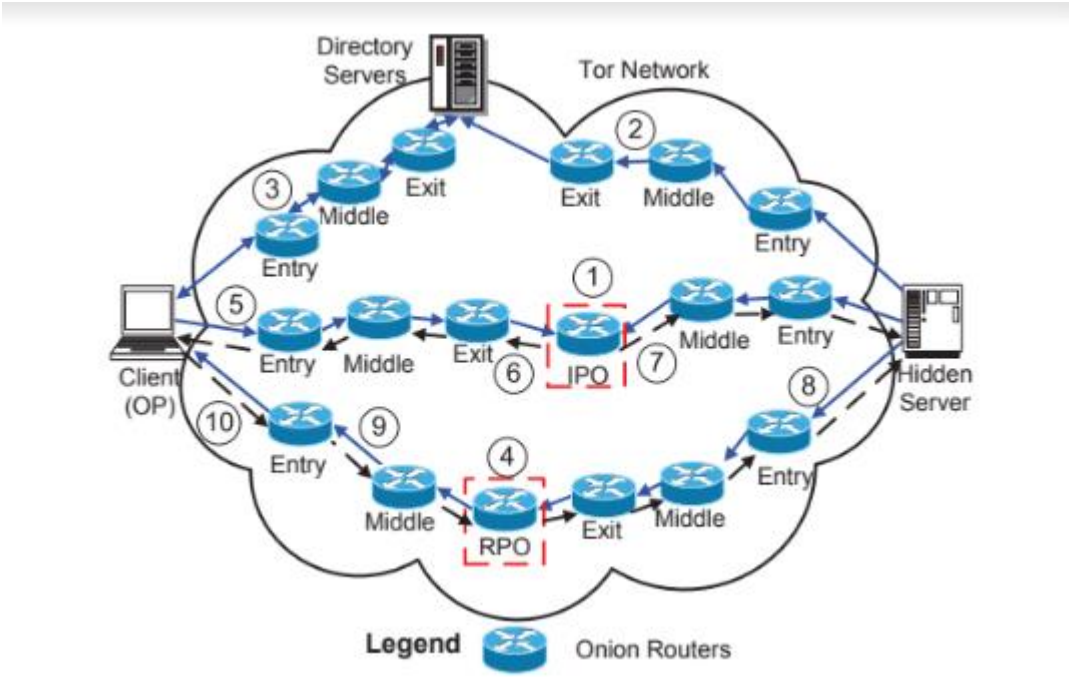

Ilustración 6 Esquema de comunicaciones de un servicio oculto (Ling, Luo, Wu, & Fu, 2013)

Cabe destacar que, aunque se puede realizar una comunicación, tanto el cliente como el servicio desconocen la ubicación del otro, ya que cuentan con circuitos que anonimiza la conexión. La complejidad del protocolo del "punto de encuentro" se ve reflejada en la velocidad de las conexiones a los servicios ocultos, que suele ser bastante pobre.

Debido a la anterior definición del funcionamiento de los servicios ocultos, se hace notorio que la manera de indexar contenido en la red de TOR no es algo fácil. Es por eso que en el siguiente apartado se analizarán las técnicas más utilizadas en la actualidad para conseguir extraer de manera automatizada enlaces válidos de la red de TOR, consiguiendo así crear un índice de contenidos de la misma.

3. Técnicas de descubrimiento de servicios ocultos

Una vez definido el funcionamiento de TOR y de los servicios ocultos que aloja, se plantea la problemática de la indexación de dichos servicios de manera automatizada, de forma que puedas ser indexados mediante técnicas de descubrimiento automático.

Con este fin, a continuación, se analizan algunas de las técnicas utilizadas actualmente a la hora de realizar descubrimiento automático de servicios ocultos, enfocando el análisis a la efectividad de dichas técnicas.

3.1. Crawling de servicios ocultos

Una de las técnicas más utilizadas por los indexadores de contenidos, tanto para servicios ocultos de la red de Tor, como para páginas web de internet, es el crawling.

El crawling consiste en definir una serie de fuentes con enlaces a páginas web a partir de los cuales, un crawler realiza un análisis del código de cada página en busca de nuevos enlaces y contenido para categorizar el enlace, consiguiendo realizar una búsqueda sobre cada enlace indexado.

Un crawler es un programa relativamente simple que, siguiendo una serie de pautas, realiza una búsqueda en profundidad de nuevos enlaces sobre el enlace proporcionado. En este caso dicho enlace es una dirección onion. Mediante el uso de crawlers, el investigador puede decidir seguir las buenas prácticas de internet, y respetar los robots.txt que puedan contener las páginas a la hora de analizar el contenido de un enlace, para definir que es lo que pueden y lo que no pueden rastrear dichos crawler dentro de una web.

Para realizar un análisis mediante técnicas de crawling, nuestra primera necesidad es la de encontrar una serie de enlaces que nos sirvan a modo de fuentes de datos iniciales, para que el crawler pueda empezar a trabajar sobre ellos.

Existen fuentes de enlaces dentro de la red de Tor con listados de enlaces a direcciones onion verificadas que se pueden utilizar como lista de fuentes iniciales para nuestro crawler, como pueden ser la Tor Hidden Wiki, o indexadores de contenidos como ahima que consta de una web en Internet, al igual que un servicio oculto en Tor.

También es habitual encontrar páginas con direcciones onion en Internet que pueden ser usadas a modo de fuentes de datos, como puede ser la web de pastebin.com, una web utilizada como un borrador de notas para compartir notas de manera anónima, o el mismo buscador de Google, que permite, mediante la utilización de técnicas de hacking en buscadores, encontrar enlaces a direcciones onion.

Finalmente, existen webs de enlaces a contenidos ocultos open source, como Ahima, ofrecen los enlaces de su base de datos para su uso público. Esta es una gran fuente de datos inicial, sobre la cual se puede realizar el análisis. Dicha información es accesible a través del siguiente enlace: contenidos Ahima.

Una vez definidas las fuentes de datos iniciales, a continuación se definen los pasos para la puesta en marcha de un crawler sobre la red de Tor.

El primer punto a realizar consiste en configurar un Onion Proxy que permita al crawler conectarse a la red de Tor. Este proxy redirecciona todo el tráfico del crawler a la red de Tor, modificado los datos enviados a nivel de protocolo para que se envien en grupos de cédulas que el protocolo de Tor pueda interpretar. Una posibilidad es la de utilizar un enrutador de tráfico de un dispositivo como proxychains con este fin.

Tras configurar el Onion Proxy, se procederá a configurar el crawler para iterar por los enlaces extraídos de las fuentes de datos definidas en el apartado anterior, realizando una búsqueda en profundidad en cada uno de ellos, e indexando los nuevos enlaces que se vayan descubriendo en una base de datos para su consulta.
Se deberá realizar un crawler que contemple solo los enlaces de tipo onion, ya que se pueden encontrar enlaces a otras redes que no son de interés para este proyecto.

Es conveniente, por el tipo de contenidos que se alojan en ciertas webs, filtrar los enlaces a archivos, imágenes, videos o cualquier otro contenido que no sea texto plano (entiéndase, páginas web). Por tanto, deberemos realizar un filtrado de los contenidos extraídos antes de su indexación en base de datos.

Los indexadores de contenido contemplados que utilizan esta técnica para el descubrimiento de servicios ocultos definen dos apartados claramente separados.

Uno de ellos, el de la búsqueda de nuevos contenidos sobre las fuentes de datos, en el cual se realiza una búsqueda de nuevos contenidos cada cierto tiempo sobre las fuentes configuradas. Esto se debe a que estas fuentes son webs dinámicas que pueden cambiar cada cierto tiempo, añadiendo o eliminando información que deberemos reindexar.

También en la propia indexación de contenidos, a la hora de categorizar el enlace obtenido desde las fuentes, se realiza un análisis del contenido del enlace, en busca de enlaces a otros servicios ocultos, que se añaden al listado de enlaces pendientes de procesar.

Por otro lado, se realiza la búsqueda sobre las páginas ya indexadas, realizando una búsqueda en profundidad sobre los datos de las webs, en busca de enlaces que se añadirán a la lista de los enlaces no procesados.

Como ya hemos comentado anteriormente, los servicios ocultos son (o pueden ser) dinámicos, y es conveniente re-indexarlos cada cierto tiempo en busca de cambios, servicio ocultos no disponibles y nuevos enlaces que procesar.

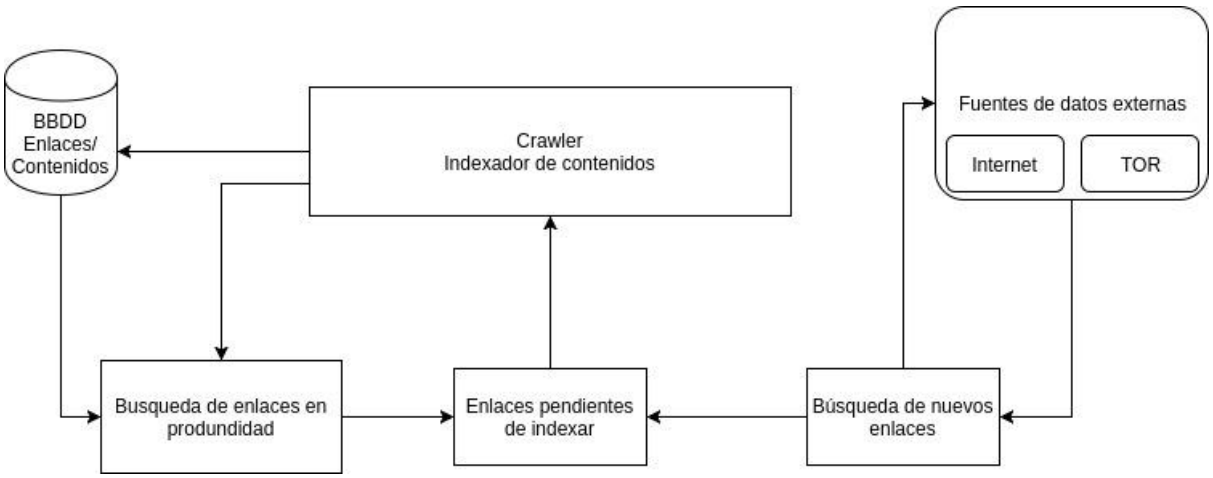

Ilustración 7 Arquitectura de un Crawler

Los servicios ocultos son, en su mayoría, servicios altamente inestables. Debido a la búsqueda de anonimato que persiguen, ciertos servicios ocultos cambian su dirección cada pocos meses. Por tanto, una de las partes más relevantes de un crawler para la red Tor es la re-indexación de contenidos cada poco tiempo, para poder mantener un listado de direcciones activas lo más preciso posible.

3.2. Ataques de fuerza bruta sobre servicios ocultos.

Otra técnica utilizada hasta ahora ha sido la generación de enlaces onion mediante fuerza bruta, mediante la cual se generan y testean todas las posibles combinaciones de letras y números para las posibles direcciones onion.

Para esto basta con utilizar cualquier programa de ataques de fuerza bruta, como puede ser john the ripper, y definir la lista de caracteres alfanuméricos que debe utilizar para generar el listado de posibles direcciones onion.

Esta técnica es válida siempre que tengamos en cuenta que las direcciones onion tienen una longitud de 16 caracteres en base32, lo que significa que podemos utilizar cualquier letra, y cualquier número excepto el 0, el 8 y el 9. Esto significa que hay 32^{16} posibles combinaciones de letras y números, o lo que es lo mismo 1208925819614629174706176 direcciones que generar.

Esto, como es obvio, implica una cantidad muy elevada de recursos, y tiempo para generar y analizar todas las posibles opciones. Además, otro de los problemas que pueden surgir es que, al ser tan costoso en tiempo y recursos, una vez finalicemos de generar todas las posibles direcciones y éstas se hayan indexado, muchas de ellas ya no serán válidas, ya que las direcciones de los servicios ocultos van cambiando en el tiempo, por lo que deberemos realizar el análisis de nuevo cada poco tiempo.

Por tanto, se puede concluir en que este tipo de técnicas de descubrimiento de servicios ocultos no es viable para el proyecto actual, ya que su eficiencia no es la adecuada.

Un contra añadido es que, en la siguiente versión de Tor, que en el momento de la redacción de esta memoria se encuentra en la versión 0.3.2.1-alpha, el protocolo se ha modificado para que los servicios creados tengan una longitud de 54 caracteres, tal y como se indica en las release notes de la versión, lo que nos supondría un problema aún mayor, teniendo que generar una cantidad de 32^{54} combinaciones para poder probar todos los posibles enlaces onion.

3.3. Descubrimiento automático de servicios ocultos mediante errores de diseño de Tor

Finalmente, otra de las técnicas analizadas consiste en aprovecharse de las vulnerabilidades de implementación y diseño de los servicios ocultos, para el descubrimiento automático de servicios ocultos.

Hasta ahora, una de las maneras más eficaces que había para realizar búsquedas automatizadas de servicios ocultos había sido crear HSDirs maliciosos, que analizarán y extrajeran información de los descriptores de servicios ocultos recibidos y los enviaran a un servidor para su indexación.

El problema a la hora de realizar este tipo de ataques es que son los relays de autoridad de la red de Tor los que gestionan qué nodos son seleccionados como HSDirs, y estos

cambian cada cierto tiempo. Por tanto, un atacante necesitaría tiempo y recursos suficientes como para generar nodos que sean seleccionados como HSDirs de la red.

Sin embargo, en el estudio realizado por Alex Biryukov, Ivan Pustogarov y Ralf-Philipp Weinmann en el paper “Trawling for Tor Hidden Services: Detection, Measurement, Deanonymization” (Biryukov, Pustogarov, & Weinmann, 2013) , utilizan una técnica llamada Shadowing para mejorar los resultados del ancho de banda mediante “shadow relays” y así definir nodos de un atacante como los nodos de HSDirs seleccionados.

La técnica de shadowing consiste en aprovechar un bug de implementación de Tor que permite alterar las estadísticas del protocolo de Tor. Esto es debido a cómo gestiona Tor los nodos de la red, ya que, aunque en el documento de consenso se define que solo se pueden tener dos relays por cada dirección IP, investigando el código descubrieron que, aunque no se publican, la red de Tor también recolecta estadísticas sobre los nodos no publicados, de tal forma que los nodos de una red se definen en dos grupos, los activos, que son los que aparecen en el documento de consenso. Y los “shadow relays”, que son relays que no aparecen en el consenso.

También detectaron que, cuando un nodo activo no era accesible, uno de los “shadow relays” se convertía en el nodo activo, pero en vez de mantener los flags del nodo anterior, este obtiene los flags correspondientes a su estado actual, y no al de los definidos en el consenso.

Por tanto, para que los nodos definidos por el atacante sean seleccionados como nodos de la red, los investigadores han realizado un estudio y han concluido en que, al ofrecer más ancho de banda para los flujos de medición de los nodos de autoridades, y minimizando el del resto de streams mediante los “shadow relays”, da como resultado que los relays obtengan un alto nivel de ancho de banda en los resultados de las mediciones, dando como resultado que los nodos del atacante sean más propensos a convertirse en nodos HSDir durante un largo plazo de tiempo.

Esto permitiría recolectar descriptores de una zona del rango de fingerprints, pero no todos los descriptores de la red.

Con este objetivo, los autores de la investigación afirman que, asumiendo una cantidad de unos 1200 HSDir en la red, si obtuviéramos 50 IPs distintas, y creando 24 nodos en cada una de ellas, se conseguiría lo siguiente:

1. Se obtendrían 100 HSDirs, debido a que, por cada IP, el consenso de Tor define que solo puede haber dos relays activos. Para esto se utilizará la técnica definida en el apartado anterior.
2. Como resultados veríamos que en la lista de HDirs, tan solo constaría de los 100 relays del atacante, ya que el resto serían relays ocultos o “shadow relays”, y por tanto no serían accesibles ni válidos.

De este modo los descriptores se almacenarán en los 100 servidores válidos definidos, y se podrían extraer los descriptores de todos ellos.

3.4. Comparativa

Una vez analizadas las posibles técnicas a implementar, a continuación, se realiza un análisis de la eficiencia, complejidad y compatibilidad en el tiempo de las técnicas analizadas.

La técnica de crawling analizada tiene como punto a favor que, al ser una de las técnicas más utilizadas para indexar contenidos tanto en Internet como en la Deep Web, tiene un gran soporte por la comunidad que puede agilizar el proceso de creación y ayudar con el mantenimiento de la infraestructura creada.

Se considera que esta técnica tiene una eficiencia media-alta, debido a que existen fuentes de enlaces que se actualizan de manera continua, y proporcionan un buen punto de partida para recorrer gran parte de la red.

Otro de los beneficios de esta técnica es que no es dependiente de la versión de Tor, ni de vulnerabilidades encontradas sobre la red. Es una técnica aplicada a nivel de aplicación, que realiza peticiones a servicios en busca de nuevos servicios que no tenga indexados.

Como punto negativo, cabe destacar que la eficacia de esta técnica no es 100% efectiva, ya que no se asegura conseguir todos los servicios ocultos existentes en la red.

Con respecto a la técnica de bruteforcing, aunque ya se ha dado una conclusión en la propia descripción de la técnica, es necesario reiterar que los recursos y el tiempo necesarios para realizar un barrido de todas las posibles direcciones no es válido para el tipo de contenidos que se quieren indexar. Los enlaces de la red de Tor, en su mayoría, cambian cada poco tiempo, debido a que los dueños de muchos de los servicios ocultos no quieren que se conozca su identidad y dirección.

A su vez, debido a los cambios que se van a implementar sobre los servicios ocultos (V3), estos pasarán de tener 16 caracteres a tener 54, lo que dificulta aún más su procesado.

Por tanto, esta técnica queda descartada.

La siguiente técnica consiste en aprovecharse de un fallo de diseño de la red Tor para conseguir que un atacante pueda crear nodos de HSDir con los cuales indexar el contenido de los descriptores de servicios ocultos que reciba.

Tal y como se a descrito en el apartado de la técnica actual, esta es la técnica más eficiente, ya que, si se realiza el ataque descrito, se obtendrían todos los servicios ocultos disponibles en la red.

Sin embargo, en comparación con la técnica de crawling descrita como primera opción, existen varios puntos negativos.

El primero de ellos es la complejidad de la aplicación de la técnica, ya que necesitamos, en el caso descrito como ejemplo en la descripción de la técnica, como mínimo 50 IPs distintas para conseguir nuestro objetivo, además de configurar 24 nodos distintos por cada IP, y la posibilidad de que para la fecha en la que sea lea este documento, la vulnerabilidad de la que se esté haciendo uso ya se haya solucionado.

Por tanto, la técnica actual es dependiente tanto de la versión de Tor, como de la vulnerabilidad encontrada en el diseño, y por tanto no es seguro que perdure su usabilidad a lo largo del tiempo.

3.5. Conclusión

Tras el análisis realizado, se consider que la técnica más equilibrada es la de crawling de la red Tor.

Los motivos son los definidos en el propio análisis, la eficiencia, la usabilidad a lo largo del tiempo, la no dependencia con respecto a las versiones, y la comunidad detras de la creación de crawlers hacen que esta técnica[4] sea la más adecuada de implementar.

4. Diseño del Crawler

A continuación se detallan los requisitos necesarios, desde el diseño y las especificaciones, hasta los riesgos que conlleva el desarrollo de la solución planteada.

La solución, como se ha mencionado en el apartado anterior, consiste en crear un crawler web que analice las páginas proporcionadas desde unas fuentes de información iniciales, y que a partir de ellas extraiga los enlaces existentes, y realizar con estos el mismo proceso, de tal forma que, mediante los enlaces entre páginas podamos analizar el mayor número de enlaces posibles.

Para ello el primer paso consiste en definir la arquitectura que deberá tener el crawler.

4.1. Arquitectura backend

La arquitectura seleccionada debe, por un lado, proporcionar la mayor flexibilidad posible, esto se debe a que puede haber cargas de trabajo puntuales que necesiten de una mayor capacidad computacional que la que pueda ofrecer un solo servidor. Para ello, se ha decidido implementar el crawler mediante la tecnología Docker, con el fin de que sea fácilmente escalable y mantenible, y ofrezca un soporte multiplataforma.

La tecnología Docker es una tecnología que permite ejecutar aplicaciones en entornos controlados llamados contenedores, algo similar a lo que se realizaba hasta ahora con las máquinas virtuales, con la diferencia de que en el caso de los contenedores, el sistema operativo es común, es decir, todos los contenedores se ejecutan sobre un mismo kernel, y lo que se encapsula en los contenedores es la actividad de las aplicaciones ejecutadas en ellos, evitando que los contenedores realicen cambios sobre el sistema.

Para gestionar múltiples de estos contenedores a la vez, se utiliza Docker-Compose, al cual también se le dará uso durante la implementación del proyecto.

Para la propia aplicación, se utilizará el framework de Spring Framework, que permite realizar aplicaciones web basadas en Java de una forma ágil, y facilita la conectividad a bases de datos y sistemas externos.

Como base de datos usaremos MongoDB, una base de datos no relacional orientada al almacenamiento de grandes cantidades de datos, y con capacidad de escalar en caso de ser necesario.

Como onion proxy se utilizará un contenedor Docker con privoxy, previamente configurado para permitir el acceso a la red de Tor.

Una vez definidas las tecnologías a utilizar, se procede a definir la arquitectura que se seguirá en la implementación del proyecto.

El proyecto consta de cuatro elementos fundamentales, enumerados tal y como se puede apreciar en la ilustración:

[4] "Documentation - Tor Project." https://www.torproject.org/docs/documentation.html.en. Se consultó el 22 dic.. 2017.

1. La base de datos MongoDB
2. El crawler
3. El comprobador del estado de los enlaces guardados
4. El onion proxy

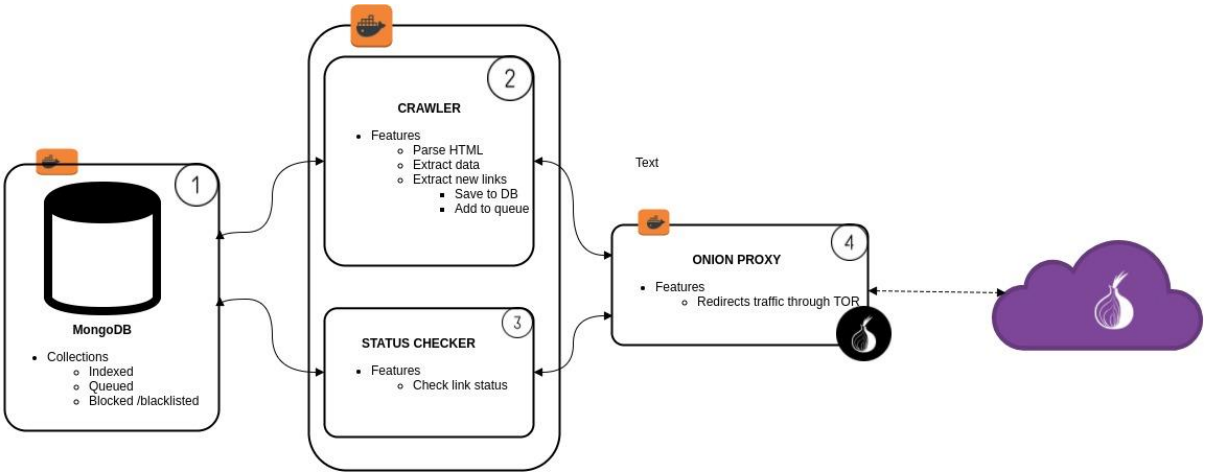

Ilustración 8 Arquitectura backend del Crawler

Cada uno de los elementos anteriores, como ya se ha mencionado, es un contenedor de Docker, lo que facilita su escalabilidad.

4.1.1. Base de datos MongoDB

La base de datos seleccionada es MongoDB. El motivo de seleccionar una base de datos no relacional, y en particular la base de datos MongoDB, es por su rapidez a la hora de procesar grandes cantidades de datos, además de facilitar las labores de programación debido a la orientación a objetos de este tipo de bases de datos.

Dicha base de datos constará con una colección que almacenará los resultados obtenidos por el crawler mediante las búsquedas realizadas. En la base de datos tan solo se almacenarán datos de tipo texto, enlaces y estados de conexión de cada uno de ellos. Los datos de tipo binario no se tratarán ni almacenarán por motivos que se expondrán en el apartado de riesgos.

La base de datos, indexará, en la medida de lo posible, datos que puedan servir para categorizar el enlace detectado a un ámbito en concreto. Para ello se extraerán ciertos elementos del HTML que ayudará a este propósito, como los metadatos que contengan las páginas analizadas.

4.1.2. Crawler

El crawler es el elemento principal de la aplicación, ya que es el encargado de listar y procesar los enlaces proporcionados como semilla, y extraer de ellos el listado de nuevos enlaces detectados, junto con los datos identificativos de la página en general.

Dicho crawler realizará una búsqueda en profundidad de los enlaces extraídos, lo que significa que, por cada enlace extraído, este será enviado a la cola de enlaces pendientes de procesar, y una vez el crawler recoja dichos enlaces, realizará la misma acción con los enlaces extraídos de este último.

El crawler también realizará tareas como comprobar el estado de cada enlace detectado dentro de un dominio, realizar un hash del contenido de cada enlace para comprobar cambios, etc.

Otra de las funcionalidades del crawler es la de filtrar los enlaces que se procesan, de forma que no procesamos enlaces con contenidos multimedia o binarios, debido al riesgo que conlleva tratar este tipo de ficheros en TOR, tal y como se explica en el apartado de 4.4. Riesgos. También se realizará un filtrado de los enlaces extraídos de las páginas analizadas, para solo procesar enlaces de servicios ocultos.

Finalmente, y siguiendo la filosofía y las buenas prácticas de los crawlers, el crawler contará con la opción de gestionar y procesar los ficheros de tipo “robots.txt”, y aplicar las reglas contenidas en estos, en caso de configurarlo con tal propósito.

4.1.3. Comprobador de estado de los enlaces

Como se menciona en los análisis, uno de los puntos más críticos a la hora de realizar un crawler para TOR, es la de tener un control sobre el estado de las páginas indexadas, debido a que, en esta red, los enlaces no son muy estables, y es común que muchos enlaces accesibles hoy, pueden no estarlo mañana. Es por esto que se ha decidido implementar un componente específico para este fin, el de gestionar y actualizar de forma constante el estado de los enlaces indexados.

El funcionamiento de este componente es relativamente sencillo. Cada cierto tiempo, configurable desde las propiedades de la aplicación, el componente recogerá la lista de enlaces indexados en la base de datos, y para cada uno de ellos, realizará una conexión, y en base a la respuesta del servidor oculto, actualizará el estado del enlace a activo o no activo.

También se realizará dicha comprobación para los enlaces internos de los dominios analizados, en caso de especificarlo de tal forma.

4.1.4. Onion Proxy

El siguiente elemento, es el contenedor del onion proxy que se deberá utilizar para la conexión de los contenedores con la red TOR.

El funcionamiento de este contenedor es sencillo, simplemente pondrá en marcha un servicio que permita la redirección del tráfico que reciba hacia la red TOR mediante un puerto específico que se deberá configurar en la aplicación, de forma que las conexiones que el crawler realice sean redireccionadas.

El contenedor constará del proxy open-source privoxy, accesible desde su web[5]. El contenedor está ya configurado para realizar la redirección a la red de Tor.

4.2. Arquitectura frontend

Una vez definido el funcionamiento interno de la aplicación, falta definir la interfaz con la que se interactúa con los datos extraídos.

Por tanto, la arquitectura a nivel de frontend contará con los siguientes elementos:

1. Interfaz de búsquedas
2. Interfaz de descripción de un enlace concreto

[5] "Privoxy - Home Page." https://www.privoxy.org/. Se consultó el 6 ene.. 2018.

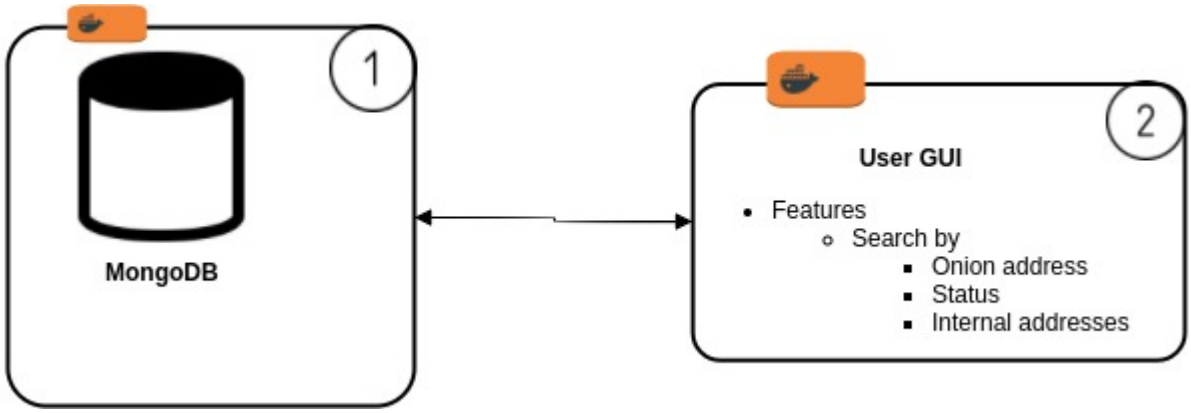

Ilustración 9 Arquitectura frontend del Crawler

4.2.1. Interfaz de búsquedas

La interfaz de búsquedas permitirá, mediante un buscador, realizar búsquedas sobre los enlaces almacenados en la base de datos, pudiendo realizar búsquedas de enlaces concretos, o búsquedas de palabras que los enlaces deberá contener.

También se añadirán filtros específicos para realizar búsquedas, como búsquedas por dirección, o por el estado, que podrán combinarse.

Finalmente, los resultados se mostrarán con un enlace a la página específica, y detalles del enlace que el crawler haya podido obtener.

También se añade en cada resultado un enlace que mostrará la página de detalles del dominio onion seleccionado.

4.2.2. Interfaz de detalles

En el panel de descripción de un enlace onion se listarán los detalles de los enlaces internos asociados al dominio seleccionado, mostrando a su vez, los detalles disponibles para cada uno de ellos, como son:

- Dirección
- Fecha del primer procesamiento
- Fecha del último procesamiento
- Estado de la página
- Hash de los contenidos de la página

4.3. Requisitos

Con respecto a los requisitos, el proyecto se implementa sobre un micro servidor HP con el sistema operativo Ubuntu 16.04.

El software necesario para ejecutar el entorno de trabajo es Docker, junto con Docker-Compose para poder ejecutar el proyecto.

También es aconsejable contar con un cliente de MongoDB para realizar búsquedas sobre los datos proporcionados.

Los requisitos mínimos de la máquina en la que se ejecute el proyecto son:

- 4 GB Ram
- 4 CPUs
- 20 GB Disco Duro

Los requisitos a nivel de software son los siguientes:

- Docker
- Docker-Compose

Aunque el proyecto se ha implementado en un servidor Linux, al tratarse de la tecnología Docker, este proyecto se puede desplegar en cualquier servidor que soporte Docker.

También existe la posibilidad de instalar los elementos mencionados anteriormente de manera individual, y adaptar la configuración de la aplicación para ejecutarla fuera de los contenedores Docker.

4.4. Riesgos

Con respecto a los riesgos a tener en cuenta a la hora de realizar el proyecto, el más destacable es el de los propios servicios ocultos. Como ya se comentaba en un inicio, las dark nets, y en específico Tor, son redes que no se encuentran controladas por gobiernos, y por tanto, no se les aplica ley alguna.

En Tor existen todo tipo de contenidos, desde bibliotecas de libros universitarios ofrecidos por universidades prestigiosas, sitios de la clear net que crean los mismos servicios en Tor para ofrecer más privacidad a sus usuarios, hasta páginas de scams, de compra de sustancias ilegales, pornografía infantil, e infinidad de contenidos ilegales.

Esto es debido a su carácter anónimo y basado en la privacidad que protege la identidad de los autores de servicios ocultos ilegales que se pueden encontrar por la red.

Por tanto, durante la realización de este proyecto, no se realizarán procesamientos de ningún elemento de tipo binario, como fotos, videos, audios, etc. que el crawler haya podido llegar a rastrear. Esto permite no almacenar contenidos ilegales que puedan dar a entender un comportamiento por parte del crawler que no queremos dar.

Es importante que quede clara la intención que se sigue mediante la realización de este proyecto, ya que la policía y diferentes organizaciones de varios países realizan búsquedas constantes sobre la red de Tor en busca de gente malintencionada, y realizan campañas utilizando servicios ocultos falsos a modo de gancho para localizar a dicha gente.

Por tanto, para asegurarnos de que nuestras intenciones quedan claras, el crawler procesará los enlaces utilizando un user-agent modificado, de forma que quede reflejada la intención y la dirección de correo a la cual nos puedan contactar en caso de que surja cualquier problema.

5. Implementación del Crawler

A continuación se listan los detalles técnicos de implementación del proyecto, la preparación del entorno de trabajo y las configuraciones aplicadas a cada elemento del proyecto.

5.1. Entorno de trabajo

El primer punto para poder implementar el proyecto consiste en adecuar el entorno de trabajo a los requisitos del proyecto, para lo cual es necesario realizar la instalación de Docker en el servidor de destino.

En caso de que el servidor de destino tenga un sistema operativo Windows, en la página del proveedor cuentan con una guía de instalación a la que se puede acceder a través del siguiente enlace.

En el entorno actual, al ser el entorno nativo de Docker, la instalación es más sencilla, y se puede realizar utilizando el siguiente comando:

```
dovixman@DVX-LPTP:~/Descargas/TFM draft/PEC4/Docker$ sudo apt-get install docker docker-compose
[sudo] password for dovixman:
```

Ilustración 10 Instalar Docker y Docker-Compose

Dicho comando instalará los dos elementos necesarios para lanzar el proyecto. Por un lado, se ha instalado Docker, el servicio que permite la gestión de contenedores que necesitaremos para ejecutar la aplicación.

Por otro lado, al contar el proyecto con varios contenedores, se ha utilizado docker-compose para gestionar todos ellos desde un solo fichero de configuración. Esto nos permite gestionar tanto la configuración como los detalles de creación de los contenedores que crearán el entorno en el que se ejecutará la aplicación.

Para poder realizar un uso más sencillo de Docker, es recomendable añadir el usuario a utilizar al grupo de docker, para no necesitar permisos de super usuario en cada comando que utilicemos.

Para este fin, se ha utilizado el siguiente comando para crear dicho grupo:

```
dovixman@DVX-LPTP:~/Descargas/TFM draft/PEC4/Docker$ $ sudo groupadd docker
```

Ilustración 11 Añadir grupo de docker

Y el siguiente para añadir el usuario deseado al grupo creado:

```
dovixman@DVX-LPTP:~/Descargas/TFM draft/PEC4/Docker$ sudo usermod -aG docker $USER
```

Ilustración 12 Añadir usuario al grupo de docker

Tras cerrar sesión deberíamos ser capaces de lanzar contenedores sin necesidad de permisos de super usuario. Podemos hacer una comprobación lanzando el contenedor de prueba, que nos debería devolver lo siguiente:

```
dovixman@NAS:~/docker/TFM$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
ca4f61b1923c: Pull complete
Digest: sha256:ca0eeb6fb05351dfc8759c20733c91def84cb8007aa89a5bf606bc8b315b9fc7
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://cloud.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/
```

Ilustración 13 Resultados contenedor 'Hello World'

Una vez preparado el entorno, se procede a configurar la aplicación.

5.2. Componentes de la aplicación

A continuación se describen los componentes utilizados y como configurarlos para realizar la instalación individual de cada uno de ellos en contenedores Docker.

Una vez definidas las configuraciones por separado, se definirá como unificarlas en un solo fichero de configuración, que será el que utilicemos con docker-compose para la gestión del entorno completo.

5.2.1. Contenedor de MongoDB

El contenedor de mongo consta de una configuración bastante simple, tan solo tenemos que utilizar la imagen oficial de MongoDB, accesible mediante el siguiente enlace.

En dicho enlace encontraremos los detalles de la implementación de la imagen, y las opciones de configuración, aunque a nosotros nos interesan dos opciones específicas:

- El puerto a utilizar por la aplicación
- La carpeta que se utilizará para mantener los datos gestionados por el contenedor.

Como se ha descrito en el apartado anterior, los contenedores Docker, son aplicaciones aisladas que no realizan modificaciones sobre el sistema operativo sobre el cual se ejecutan. Para poder mantener cambios y modificar configuraciones, se utilizan mapeos de carpetas y puertos, para configurar la aplicación a nuestro gusto, y mantener los cambios realizados dentro de la aplicación entre ejecuciones.
Por tanto, una vez definidas las opciones que nos interesan, procedemos a crear el contenedor mediante el siguiente comando:

```
dovixman@NAS:~/docker/TFM$ docker run -d -p 27017:27017 -v /home/dovixman/docker/data:/data/db --name mongo mongo:latest
```
Ilustración 14 Crear contenedor MongoDB

- **-d**
  - Esta opción define que queremos que el contenedor se ejecute en segundo plano una vez se haya creado.
- **-p 27017:27017**
  - Esta opción define que queremos exponer el puerto 27017 del contenedor en el puerto 27017 de la máquina en la cual se encuentra. Esto permite acceder a dicho contenedor desde un dispositivo externo.
- **-v /home/dovixman/docker/data:/data/db**
  - La siguiente opción define el mapeo de carpetas entre el contenedor y la máquina host. Permite que los cambios realizados dentro del contenedor se almacenen en una carpeta del host para no perderlos.
  - Una de las ventajas de docker, es que podemos utilizar las carpetas mapeadas entre múltiples contenedores, de manera que podemos compartir información entre ellos sin permitir comunicación alguna entre contenedores.
- **--name mongo**
  - Define el nombre que se le da al contenedor como “mongo”
- **mongo:latest**
  - Finalmente, esta opción define la imagen sobre la que queremos crear el contenedor con la configuración anterior. En este caso se trata de la imagen oficial de mongoDB, en su versión más actual.

Una vez ejecutado el comando, podremos acceder a la base de datos de mongoDB utilizando la IP del servidor y el puerto configurado en las propiedades del contenedor (27017).

```
dovixman@DVX-LPTP:~$ mongo 192.168.1.155:27017
MongoDB shell version: 2.6.10
connecting to: 192.168.1.155:27017/test
```
Ilustración 15 Test del contenedor de MongoDB

Para inicializar el contenedor con los datos utilizados durante la realización del proyecto, se adjunta los datos en formato json para su importación.

Para ello lanzar el siguiente comando contra el contenedor de mongoDB.

```
dovixman@DESKTOP-6IKSRFA:~$ mongoimport --host mongo --db tor-index --collection domains --file domains.json
```
Ilustración 16 Importar datos a MongoDB

Tras lo cual, el programa deberá marcar que ha importado 13103 documentos de manera correcta.

5.2.2. Contenedor de Privoxy

El contenedor de Privoxy es el contenedor encargado de redireccionar el tráfico entrante hacia la red de Tor, para ello, la imagen seleccionada cuenta con la configuración necesaria para realizar la redirección a Tor. Dicha imagen es accesible mediante el siguiente enlace.

Para su configuración necesitamos definir el puerto sobre el cual trabajará el servidor proxy. Para ello ejecutaremos el siguiente comando, que creará un nuevo contenedor Docker:

```
dovixman@NAS:~/docker/TFM$ docker run -d -p 8118:8118 --name privoxy rdsubhas/tor-privoxy-alpine
```

Ilustración 17 Creación de contenedor del Onion Proxy

- **-d**
  - Esta opción define que queremos que el contenedor se ejecute en segundo plano una vez se haya creado.
- **-p 8118:8118**
  - Esta opción define que queremos exponer el puerto del proxy en el puerto 8118 de la máquina host.
- **--name privoxy**
  - Define el nombre que se le da al contenedor como “privoxy”
- **rdsubhas/tor-privoxy-alpine:latest**
  - Finalmente, esta opción define la imagen sobre la que queremos crear el contenedor con la configuración anterior. En este caso se trata de la imagen oficial de tor-privoxy-alpine, en su versión más actual, creada por el usuario ‘rdsubhas’ para la comunidad de Docker.

Si todo se ha realizado correctamente, se podrá realizar una prueba utilizando el proxy para conectar a un enlace onion como el siguiente:

```
dovixman@DVX-LPTP:~$ curl --proxy 192.168.1.155:8118 msydqstlz2kzerdg.onion


<!doctype html>
<html class="no-js" lang="en">
  <head>
      <meta charset="utf-8">
      <meta http-equiv="x-ua-compatible" content="ie=edge">
      <title>Ahmia &mdash; Search Tor Hidden Services</title>
      <meta name="description" content="A search engine for services accessible on the Tor network.">
      <meta name="viewport" content="width=device-width, initial-scale=1">
      <script src="/static/js/jquery.min.js"></script>
      <link rel="icon" href="/static/images/favicon.ico" type="image/x-icon">
      <link rel="stylesheet" href="/static/css/normalize.css">
```

Ilustración 18 Test del contenedor de Privoxy

5.2.3. Contenedor del Crawler

El Crawler es la aplicación desarrollada en el proyecto, utilizando para ello las librerías de Spring framework y la librería de crwler4j modificada, de forma que permita la conexión a servicio ocultos.

Al ser esta una aplicación Java, el contenedor a crear no se puede encontrar en Docker Hub, el repositorio de imágenes de Docker, sino que tendremos que crear nuestra propia imagen para poder utilizarlo.

Para ello, el primer paso consiste en crear un fichero Dockerfile, que contiene los detalles de creación de la imagen que vamos a generar.

El fichero creado para dicho propósito es el siguiente:

```
dovixman@NAS:~/docker/TFM$ cat Dockerfile
FROM openjdk:8-jdk
COPY TorCrawler.jar /app/torcrawler.jar
COPY application.properties /app/conf/application.properties
COPY seed.txt /app/seed/seed.txt
WORKDIR /app
MAINTAINER David Fuentes <dfuentesi@uoc.edu>
ENV JAVA_VER 8
ENV JAVA_HOME /usr/lib/jvm/java-8-oracle
ENV SPRING_CONFIG_NAME=application
ENV SPRING_CONFIG_LOCATION=/app/conf/
EXPOSE 8080
VOLUME ["/app/conf", "/app/seed"]
CMD ["java", "-jar", "/app/torcrawler.jar"]
```

Ilustración 19 Dockerfile

El él estamos definiendo lo siguiente:

- **FROM openjdk:8-jdk**
  - Vamos a crear nuestra imagen, sobre una imagen oficial de openjdk, en la versión “8jdk”. Esta imagen trae consigo la versión 8 de opnjdk preparada para su uso.
- **COPY TorCrawler.jar /app/torcrawler.jar**
  - Copiamos el fichero TorCrawler.jar, que deberá estar en la misma localización desde la que se ejecute el fichero Dockerfile, a la carpeta /app/torcrawler.jar, dentro del contenedor de docker.
- **COPY application.properties /app/conf/application.properties**
  - Copiamos el fichero application.properties a la carpeta /app/conf dentro del contenedor.
- **COPY seed.txt /app/seed/seed.txt**
  - Copiamos el fichero seed.txt, que contiene los enlaces que harán de semilla para el comienzo de la aplicación, a la carpeta /app/seed dentro del contenedor.
- **WORKDIR /app**
  - Definimos el directorio /app como el directorio de la aplicación.
- **MAINTAINER David Fuentes <dfuentesi@uoc.edu>**
  - Definimos una persona de contacto para que, en caso de subir la imagen a un repositorio como Docker Hub, la gente que lo utilice pueda contactar con nosotros.
- A continuación se definen las variables de entorno:
  - JAVA_VER: 8
  - JAVA_HOME: /usr/lib/jvm/java-8-oracle
  - SPRING_CONFIG_NAME:application
    - Definimos el nombre del fichero de configuración que deberá buscar Spring en el arranque de la aplicación.
  - SPRING_CONFIG_LOCATION:/app/conf/
    - Definimos el path al fichero de configuración que usará Spring en la ejecución de la aplicación.
- **EXPOSE 8080**
  - Exponemos el puerto sobre el cual se ejecutará la aplicación, para poder realizar mapeo de puertos en el contenedor que crearemos.
- **VOLUME [“/app/config”, “/app/seed”]**
  - Definimos las carpetas que queremos que el host mapee a un directorio local para no perder información.
- **CMD [“java”, “-jar”, “/app/torcrawler.jar”]**

- Finalmente definimos el comando que queremos ejecutar para inicializar la aplicación. En este caso se trata de lanzar la aplicación java mediante el comando “java -jar /app/torcrawler.jar”.

Una vez definidos los detalles del fichero de configuración Dockerfile, vamos a definir como se utiliza dicho fichero para crear una nueva imagen en el servidor host. Para ello utilizaremos el siguiente comando:

```
dovixman@NAS:~/docker/TFM$ docker build -t dovixman/tor-crawler:latest .
Sending build context to Docker daemon 85.84 MB
Step 1/13 : FROM openjdk:8-jdk
 ---> 7c57090325cc
Step 2/13 : COPY TorCrawler.jar /app/torcrawler.jar
 ---> Using cache
 ---> 5eac3f61e12a
Step 3/13 : COPY application.properties /app/conf/application.properties
 ---> Using cache
 ---> 4d560e789bcd
Step 4/13 : COPY seed.txt /app/seed/seed.txt
 ---> Using cache
 ---> 39820108f2e2
Step 5/13 : WORKDIR /app
 ---> Using cache
 ---> b77c7d006f7b
Step 6/13 : MAINTAINER David Fuentes <dfuentesi@uoc.edu>
 ---> Using cache
 ---> 193619871ca4
Step 7/13 : ENV JAVA_VER 8
 ---> Using cache
 ---> 85e09e12afef
Step 8/13 : ENV JAVA_HOME /usr/lib/jvm/java-8-oracle
 ---> Using cache
 ---> 9f5476da6f86
Step 9/13 : ENV SPRING_CONFIG_NAME application
 ---> Using cache
 ---> ceeb9d4e061c
Step 10/13 : ENV SPRING_CONFIG_LOCATION /app/conf/
 ---> Using cache
 ---> c761ceb17b96
Step 11/13 : EXPOSE 8080
 ---> Using cache
 ---> a05564d9ba2b
Step 12/13 : VOLUME /app/conf /app/seed
 ---> Using cache
 ---> 957da68e4dd2
Step 13/13 : CMD java -jar /app/torcrawler.jar
 ---> Using cache
 ---> f68ed03299fd
Successfully built f68ed03299fd
```

Ilustración 20 creación de la imagen de tor-crawler

En el comando hemos definido que el tag que queremos asignarla a la imagen generada es “dovixman/tor-crawler:latest”. Desglosado, significa que el usuario que ha creado la imagen, en este caso “dovixman”, ha creado una imagen nueva llamada “tor-crawler”, y la ha etiquetado como la versión más reciente o “latest”.

Una vez obtenida la imagen, podemos crear el contenedor con la aplicación utilizando para ello el siguiente comando:

```
dovixman@NAS:~/docker/TFM$ docker run -d -p 80:8080 -v /home/TorCrawler/conf:/app/con
f -v /home/TorCrawler/seed:/app/seed --link mongo:mongo --link privoxy:privoxy --name
tor-crawler dovixman/tor-crawler:latest
```

Ilustración 21 Creación del contenedor de tor-crawler

- **-d**
  - Ejecuta el contenedor en segundo plano
- **-p 80:8080**
  - Mapea el puerto 8080 del contenedor al puerto 80 de la máquina de host.
- **Volúmenes mapeados**
  - se han mapeado los volúmenes de /app/conf y /app/seed a carpetas locales de la máquina host.
- **--link**
  - Permite la comunicación entre contenedores, en este caso se necesita comunicación a los siguientes contenedores, que podremos definir mediante el --name utilizado en la creación de cada uno:
    - mongo
    - privoxy
- **--name tor-crawler**
  - Se ha asignado el nombre de “tor-crawler” al contenedor generado
- **dovixman/tor-crawler:latest**
  - Se ha creado el contenedor a partir de la imagen generado anteriormente.

Una vez creemos el contenedor, podremos comprobar su funcionamiento accediendo a la URL http://localhost/ , o mediante la IP del servidor host.

5.2.4. Docker-Compose

Habiendo generado los componentes por separado, procedemos ahora a ver cómo podemos gestionar todos los elementos anteriores mediante un solo fichero de configuración.

Este fichero de configuración se denomina docker-compose.yml, y define todos los aspectos necesarios para crear los contenedores anteriores. A continuación se puede ver el docker-compose.yml creado para el proyecto actual:

```
dovixman@NAS:~/docker/TFM$ cat docker-compose.yml
version: '2'
services:
  mongo:
    image: "mongo"
    container_name: "mongo"
    ports:
    - "27017:27017"
    volumes:
    - "/home/dovixman/docker/mongodb/data/:/data/db/"
  privoxy:
    image: "rdsubhas/tor-privoxy-alpine"
    container_name: "privoxy"
    ports:
    - "8118:8118"
    - "9050:9050"
  crawler:
    build: .
    container_name: "tor-crawler"
    ports:
    - "80:8080"
    volumes:
    - "/home/dovixman/docker/TorCrawler/config/data/:/app/config/data"
    - "/home/dovixman/docker/TorCrawler/config/:/app/conf/"
    - "/home/dovixman/docker/TorCrawler/seed/:/app/seed/"
    links:
    - mongo:mongo
    - privoxy:privoxy
```

Ilustración 22 docker-compose.yml

Como se puede observar, en él definimos los tres servicios que vamos a utilizar, y para cada uno de ellos definimos la imagen (image), o Dockerfile (build) desde el cual se tienen que crear, los nombres de los contenedores (container_name), y la configuración definida en los apartados anteriores. Todo unificado bajo un solo fichero, y que, mediante docker-compose, permitirá gestionar todos los contenedores al mismo tiempo.

Para gestionar dichos contenedores, tan solo tenemos que acceder a la localización del fichero docker-compose.yml y ejecutar uno de los siguientes comandos.

Para crear y lanzar todos los contenedores, se utiliza el comando:

```
dovixman@NAS:~/docker/TFM$ docker-compose up -d
Creating network "tfm_default" with the default driver
Creating mongo
Creating privoxy
Creating tor-crawler
```

Ilustración 23 Iniciar los contenedores configurados en docker-compose

Este comando creará las imágenes de sus respectivas fuentes y creará un entorno con una subred que aislará las aplicaciones del resto de contenedores.

Para detener los contenedores y destruir el entorno, se utilizará el comando:

```
dovixman@NAS:~/docker/TFM$ docker-compose down
Stopping tor-crawler ... done
Stopping mongo ... done
Stopping privoxy ... done
Removing tor-crawler ... done
Removing mongo ... done
Removing privoxy ... done
Removing network tfm_default
```

Ilustración 24 Parar y eliminar los contenedores configurados en docker-compose

5.4. Funcionamiento

Tras crear el entorno adecuado y realizar las configuraciones anteriores, a continuación se detalla el funcionamiento de la aplicación, y las posibilidades que ofrece.

En este apartado se detallarán las funcionalidades implementadas a nivel de backend, detallando el flujo de trabajo completo que el crawler realiza, y una vez detallado, se mostrará el frontend implementado, junto con sus opciones.

5.4.1. Crawler

El crawler, como hemos comentado en los apartados anteriores, es el elemento encargado de realizar el procesado de las páginas con el fin de descubrir enlaces onion y procesarlos en busca de enlaces no descubiertos.

Dicho crawler necesita de una fuente de enlaces para inicializar la búsqueda a la que llamaremos “seed” o semilla, y la cual se cargará antes de iniciar el crawler a modo de fuente de inicialización.

El crawler cuenta con una propiedad que se puede definir en el application.properties mencionado en el apartado anterior, más concretamente la propiedad “**crawler.config.seedfile.path**”. Dicha propiedad define la localización del fichero de texto que se deberá utilizar para definir los enlaces que contengan enlaces onion y se utilizará para inicializar el crawler en el primer arranque.

Los enlaces definidos en este fichero no tienen porque ser enlaces onion, sino que se podrá hacer referencia a una web de Internet, y deberá estructurarse de forma que haya un enlace por cada línea en el fichero.

Además de la configuración inicial, el crawler cuenta con propiedades que se pueden ajustar en base a las necesidades del usuario que ejecute el proyecto:

- crawler.config.user-agent
  - User-agent a utilizar por el crawler
- crawler.config.timeout
  - Tiempo de timeout de las conexiones realizadas en milisegundos
- crawler.config.crawl.default.thread-number
  - Número de hilos que lanzará el crawler por defecto
- crawle.config.crawl.domain-only
  - Define si solo se procesarán los nuevos dominios, o también se procesarán los enlaces que hagan referencia a dominios ya procesados.
- crawler.config.crawl.max-crawling-depth
  - Define el límite de profundidad para los enlaces procesados, por defecto esta propiedad tiene el valor -1 o ilimitado.
- crawler.config.crawl.follow-redirects

    - Define si el crawler debe permitir redirecciones
- crawler.config.crawl.binaries
    - Define si el crawler debe procesar ficheros binarios como imágenes, videos, etc.
- crawler.config.crawl.https
    - Define si el crawler debe procesar las páginas HTTPS
- crawler.config.crawl.max-outgoing-links
    - Define el número máximo de enlaces externos que procesar desde una misma página
- crawler.config.crawl.resumable
    - Establece si el crawler debe ejecutarse de forma que pueda continuar la ejecución en caso de fallo o de parada de la aplicación.
    - En caso de definir esta propiedad como true, cabe mencionar que el rendimiento se ve afectado.
- crawler.config.robots.enable
    - Especifica si el crawler debe o no tener en cuenta los robots.txt

Estas propiedades sirven para inicializar la configuración del crawler.

Para que el crawler pueda guardar los datos a la base de datos MongoDB configurada con anterioridad, el crawler también cuenta con propiedad para esta configuración.

- spring.data.mongodb.host
    - Host de la base de datos MongoDB, al ejecutar la aplicación en un contenedor, el nombre del host será el nombre que se le haya dado al contenedor.
- spring.data.mongodb.port
    - El puerto de conexión a la base de datos MongoDB. Por defecto el puerto es 27017.
- spring.data.mongodb.database
    - El nombre de la base de datos de MongoDB.

Aunque en el proyecto se proveerá un fichero de configuración pre-configurado, las propiedades anteriores se pueden editar al gusto del usuario.

Una vez realizadas todas las configuraciones necesarias, tras arrancar la aplicación existen dos posibilidades de ejecutar el crawler.

La primera de ellas, permite configurar una tarea programa que se ejecutará cada cierto tiempo en busca de nuevos enlaces a partir de los enlaces existentes.

La segunda es la opción de utilizar la API Rest integrada en la aplicación, de forma que se pueda lanzar la tarea del crawler de manera manual.

5.4.2. Status Checker

El status checker o comprobador de enlaces, es una funcionalidad añadida al crawler que permite, de forma independiente al procesado de enlaces realizado por el crawler, comprobar el estado de los enlaces previamente descubiertos.

Esto permite mantener el estado de los enlaces lo más actualizado posible, de forma que el usuario tenga conocimiento de la última vez que se realizó la comprobación, y el resultado de esta.

El status-checker, tras analizar la comprobación de un enlace, devuelve una de las dos opciones siguientes:

- UP
    - Se ha accedido a la página correctamente en la fecha de la última comprobación.
- DOWN
    - La página no ha podido ser accedida.
    - El hecho de no haber podido ser accedida no implica de forma directa que la página no sea accesible, sino que pueden existir causas, como un tiempo de carga excesivamente alto, o que la página realice una comprobación del user-agent, y nos estén cortando el acceso a ella.

El status-checker tiene la opción, al igual que el crawler, de ejecutar la tarea de comprobación del estado de los enlaces solo sobre los dominios encontrados, o también sobre los enlaces asociados a dichos dominios, en cuyo caso, realizará una comprobación para cada enlace y sub-enlaces asociados.

En este último caso, hay que tener en cuenta que el timeout asociado a las direcciones onion suele ser alto, debido al tiempo de carga que conlleva al generar los circuitos necesarios y realizar las conexiones. Por tanto, puede que la tarea se alargue más de lo esperado.

Al igual que con el crawler, esta tarea contará con la opción de programarla, de forma que se lance automáticamente pasado el periodo de tiempo definido, o se podrá utilizar la API Rest para lanzar la tarea de manera manual.

5.4.3. Tareas programadas

Las tareas programadas permiten lanzar las tareas de las dos funcionalidades anteriores, la del crawler y la del status-checker, de forma automática en base a la configuración especificada para cada tarea.

5.4.3.1. Crawler

La tarea del crawler es la encargada como ya se ha dicho de procesar enlaces para así descubrir nuevos enlaces a partir de ellos.

Esta tarea, por razones de eficiencia, es una tarea que necesita ser ejecutada constantemente, de forma que, en caso de que alguno de los enlaces almacenados en la base de datos cambie, y contenga nuevos enlaces que procesar, sean detectados.

Por tanto, se ha añadido al apartado de configuración varias propiedades que definen cuando se debe ejecutar la tarea programada:

- crawler.config.crawl.schedule.fixed-delay
    - Define el tiempo que debe pasar entre el final de la ejecución de la tarea y el comienzo de esta de nuevo.
- crawler.config.crawl.schedule.initial-delay
    - Define el lapso de tiempo inicial que debe pasar hasta la ejecución de la primera instancia de la tarea.

Una vez definida la configuración de la ejecución programada de la tarea, esta comenzará a ejecutarse en el inicio del programa.

5.4.3.2. Status-Checker

Al igual que el crawler, el status checher es de vital importancia a la hora de procesar los enlaces. Esto es debido a que, en Tor, los servicios ocultos son altamente inestables, y pueden estar disponibles durante cortos periodos de tiempo.

Es por ello que se cree que una tarea que compruebe los enlaces almacenados en base de datos es necesaria. Para ello, al igual que con la tarea programada para el crawler, hay que definir unas propiedades:

- crawler.config.status-checker.schedule.fixed-delay
    - Define el tiempo que debe pasar entre el final de la ejecución de la tarea y el comienzo de esta de nuevo.
- crawler.config.status-checker.schedule.initial-delay
    - Define el lapso de tiempo inicial que debe pasar hasta la ejecución de la primera instancia de la tarea.

Una vez definida la configuración de la ejecución programada de la tarea, esta comenzará a ejecutarse en el inicio del programa.

5.4.4. API Rest

Se ha definido como realizar la automatización de la ejecución de las tareas mencionadas en el apartado de Componentes de la aplicación. Pero hay situaciones que hacen necesario ofrecer un control manual sobre la ejecución de dichas tareas.

Es por ello que se ha definido una API Rest que ofrece la opción de lanzar las tareas de manera manual, de forma que sea el usuario quien decida cuando quiere lanzar una aplicación.

La API Rest definida tiene la siguiente arquitectura:

- /crawler
    - /manage
        - /start
            - Lanza la tarea del crawler, pudiendo definir el parámetro “threadNumber”, que definirá cuantos hilos queremos utilizar.
        - /stop
            - Para los hilos de crawler y finaliza la ejecución.
        - /waitUntilFinish
            - Permite a una petición quedar a la espera hasta que el hilo de ejecución de la tarea actual finalice.
        - /add-seed
            - Añade un enlace a modo de semilla al crawler, para que en la siguiente ejecución este lo procese. El enlace a añadir se debe enviar en el parámetro “ulr”.
    - /config
        - /getConfig
            - Muestra la configuración actual del crawler
        - /updateConfig
            - Permite actualizar la configuración del crawler mediante los siguientes parámetros:
                - timeout
                    - Tiempo de timeout
                - user-agent
                    - User-Agent a utilizar por el crawler

        - maxDepthOfCrawling
            - Profundidad máxima de procesado de enlaces.
        - folloRedirects
            - Define si el crawler debe permitir redirecciones
        - crawlBinaries
            - Define si el crawler debe procesar archivos binarios, como imágenes, videos, etc.
        - crawlHttpsPages
            - Define si el crawler debe procesar pñaginas HTTPS.
        - maxOutgoingLinksToFollow
            - Define el número máximo de enlaces externos que procesar desde un mismo enlace
        - resumableCrawling
            - Define si el crawler se debe lanzar a modo de fallos o no, para poder reanudar su ejecución en caso de error o parada del crawler.
    - /útil
        - /domainExist
            - Devuelve verdadero o falso, en base a si un servicio oculto existe o no.
        - /domainStatus
            - Devuelve UP/DOWN en base a si un servicio oculto existe o no.
- /status-checer
    - start
        - Inicial la tarea del status-checker. Permite definir una propiedad, “domainOnly”, que define si la comprobación se debe realizar solo sobre los dominios, o sobre todos los enlaces de la base de datos.
    - /status
        - Indica si el status-checker se encuentra en funcionamiento o no.

5.4.5. Interfaz gráfica

Finalmente, para permitir la interacción con la aplicación se ha creado un buscador simple, que permite realizar búsquedas por dominio y enlaces internos de los servicios ocultos.

En la imagen a continuación, se puede observar dicha interfaz, que será accesible en el puerto 80 (dependiente de la configuración del contenedor donde se despliegue), en la IP del host de la aplicación.

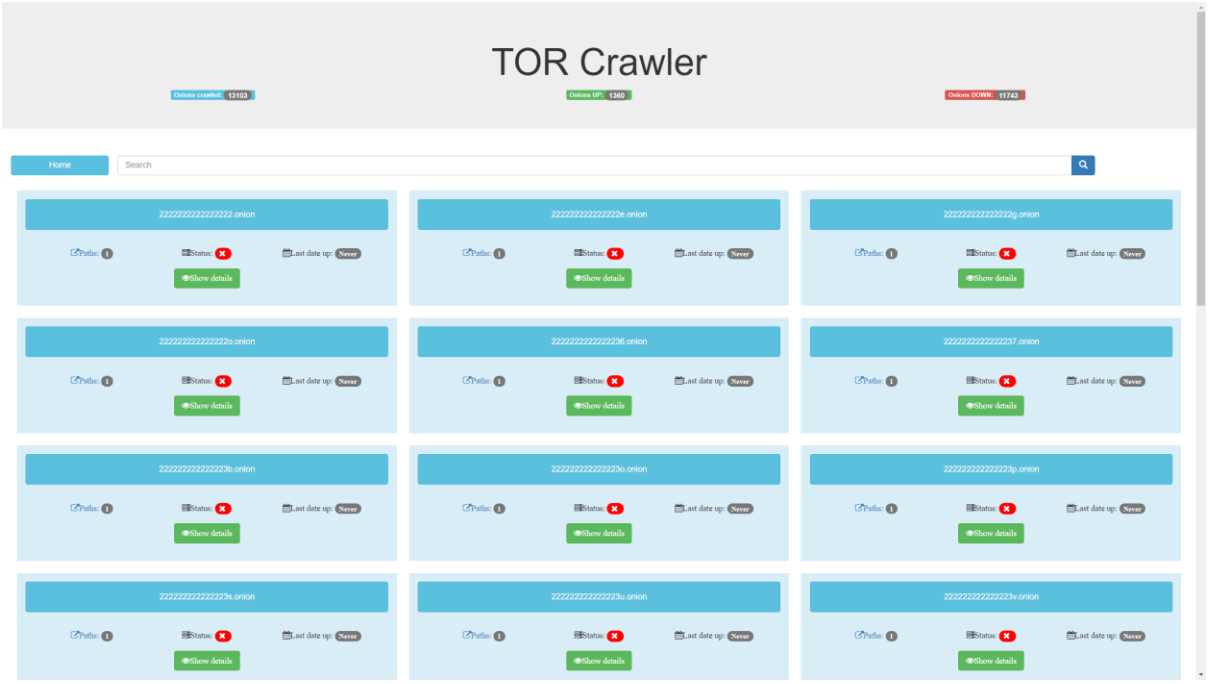

Ilustración 25 Interfaz buscador

Como se puede apreciar, bajo el título se muestran los datos de cuantos servicios ocultos se han procesado, y cuántos de ellos se encuentran online y cuantos no desde la última comprobación.

También podemos observar los servicios ocultos, junto con su último estado, los enlaces internos procesados para cada uno y la fecha de la última vez que se detectaron online.

Debajo del título se encuentra el buscador, que nos permite realizar búsquedas de tipo “regex” tanto sobre en domino, como sobre los enlaces internos de los servicios ocultos procesados.

También cuenta con múltiples filtros que permite realizar búsquedas más concretas, como son:

- site
  - Permite realizar búsquedas en base únicamente al dominio de los servicios ocultos.
  - Ejemplo:
    - site:bitcoin
- status
  - Permite realizar búsquedas en base al estado de los servicios ocultos.
  - Ejemplo:
    - status:up
    - site:bitcoin status:up

Finalmente, cada resultado también cuenta con un enlace a la ventana de propiedades de cada dominio, que mostrará al usuario los detalles almacenados en base de datos de los enlaces internos que referencian el dominio seleccionado:

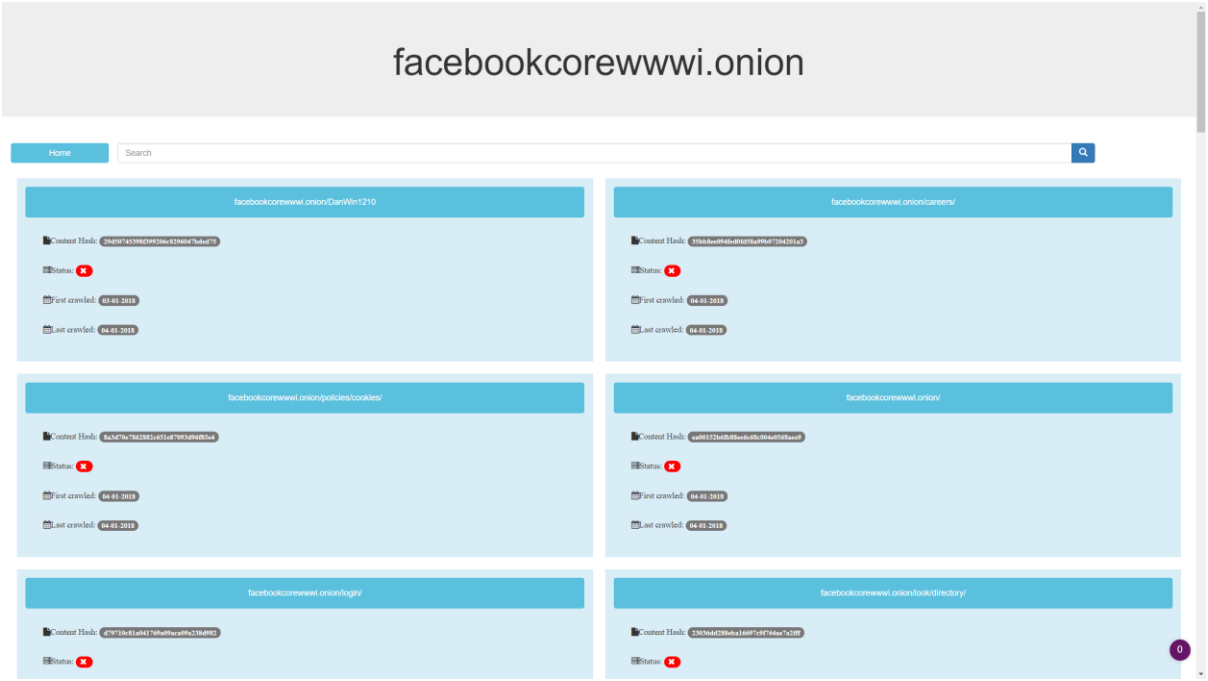

Ilustración 26 Detalles de los enlaces internos del servicio oculto de facebook

6. Conclusiones

Las conclusiones obtenidas de los resultados del proyecto son buenas en términos generales.

A pesar de la falta de tiempo, en dos días el crawler ha sido capaz de procesar y almacenar miles de servicios ocultos, y tras realizar labores de comprobación de dichos servicios, se han observado los comportamientos esperados.

En su mayor parte, los servicios ocultos procesados cambiaban el estado de su conexión de manera continua, manteniendo un comportamiento inestable.

La aplicación ha sido capaz de procesar 13103 enlaces con un seed inicial de 5000 enlaces desde webs como pastebin, o buscadores de servicios ocultos que comparten sus fuentes.

A la hora de escribir este documento, el crawler ha detectado 1360 servicios ocultos activos, y 11743 no activos, que se comprueban cada seis horas para detectar cambios.

A de tenerse en cuenta que, durante el desarrollo de este proyecto se ha utilizado un servidor con recursos limitados, y la ejecución del crawler ha sido inestable por los continuos cambios aplicados, por lo que se espera que, en una ejecución estable, los resultados puedan mejorar.

Además, tal y como se determinó en el análisis, este sistema no es dependiente de la versión ni de las vulnerabilidades de Tor, ya que funciona de igual manera.

Por tanto, tras analizar los resultados, se puede concluir en que, con los recursos necesarios y tras un tiempo de procesado, la eficiencia del crawler puede aumentar de manera significativa, realizando el descubrimiento automático de servicios ocultos planteado como problema en el documento actual.

Los datos analizados en el proyecto se adjuntan en la carpeta Anexos del proyecto, dentro de la carpeta Docker, para facilitar el análisis y la replicación del entorno utilizado.

7. Futuras mejoras

Una de las funcionalidades que se echan en falta, y que no se han desarrollado por límites de tiempo durante la realización del proyecto, ha sido la creación de una interfaz de configuración del crawler, que permite, sin necesidad de tocar los ficheros de configuración y desde una interfaz totalmente web, realizar la configuración y el mantenimiento, tanto del crawler como del status-checker.

Para ello, un primer paso si se ha realizado, la creación de una API Rest que permita de manera manual para el usuario, interactuar con la aplicación.

Otra de las mejoras identificadas, es la de realizar una comprobación más exhaustiva de los enlaces almacenados en la base de datos, para poder actualizar su estado.

Actualmente se realiza un intento de conexión mediante un proxy para detectar si el enlace esta online o no. Una posible mejora sería realizar dicho intento de conexión múltiples veces, en caso de que el primer intento resulte en error. Ayudaría para esto, lanzar las comprobaciones en hilos separados para poder avanzar más rápidamente en las comprobaciones.

A nivel de hardware, una opción es la ejecución del crawler en modo cluster entre múltiples servidores, en caso de contar con los suficientes recursos, que hagan que el procesado de los enlaces sea más rápido.

8 Anexos

La carpeta Docker contiene los recursos necesarios para recrear el entorno de desarrollo utilizado en el proyecto, que se describe en el apartado 5.2. Componentes de la aplicación.

Los recursos disponibles son:

1. application.properties
2. docker-compose.yml
3. Dockerfile
4. domains.json
5. Readme.txt
6. seed.txt
7. TorCrawler.jar

La carpeta de TorCrawler contiene el proyecto implementado.

9 Bibliografía

Biryukov, A., Pustogarov, I., & Weinmann, R. P. (2013). Trawling for tor hidden services: Detection measurement deanonymization. *Proceedings of the 2013 IEEE Symposium on Security and Privacy* (págs. 80-94). Washington DC USA: IEEE Computer Society.

Dingledine, R., Mathewson, N., & Syverson, P. (2004). or: The second-generation onion router. Washington DC: Naval Research Lab .

Elices, J. A., & Pérez-González, F. (2013). Locating Tor hidden services through an interval-based traffic-correlation attack. *Communications and Network Security (CNS)* (págs. 385-386). IEEE: IEEE Conference.

Ling, Z., Luo, J., Wu, K., & Fu, X. (2013). Protocol-level hidden server discovery. *2013 Proceedings IEEE* (págs. 1043-1051). IEEE: INFOCOM.

Raghavan, S., & Garcia-Molina, H. (2000). Crawling the hidden web. Standford.

Tor Project. (s.f.). *Tor Project*. Obtenido de Tor Project: https://www.torproject.org/

Wang, R., Wen, Q., Zhang, H., Qin, S., & Li, W. (2016). Transparent Discovery of Hidden Service. *IEICE TRANSACTIONS on Information and Systems*, (págs. 2817-2820).

Xataka. (2017). *Xataka.* Obtenido de Xataka: https://www.xataka.com/aplicaciones/deep-web-dark-web-y-darknet-cuales-son-las-diferencias