Universidad Internacional de La Rioja (UNIR)

ESIT

Máster universitario en Seguridad Informática

MarkeTOR: Un mercado en la red TOR

Trabajo Fin de Máster

Presentado por: García Alonso, David

Director/a: Sánchez Rubio, Manuel

Ciudad: Palencia

Fecha: Julio 2019

Resumen

Cuando las personas oyen hablar de la Deep Web, Dark Web, la red TOR o la red I2P, se piensa automáticamente en que tienen contenido 'perturbador', donde existen mercados donde se negocia con contenidos oscuros e ilegales entre otras mercancías ilícitas

Un 'dark market' en la red TOR es un sitio web que opera en esa red y para ser accedido se necesita un navegador especial que se conecte a ella y permita acceder a este servicio de forma anónima. Para ello aprovechan las propiedades de la red TOR, que permite operar estos servicios de forma anónima y garantizan lo máximo posible la anonimidad de sus usuarios.

Tradicionalmente han funcionado como mercados negros, vendiendo o realizando transacciones como intermediario, relacionadas con la venta de armas, ciber-armas, drogas, medicamos, dinero falsificado y otras mercancías ilícitas.

Estos mercados, también son conocidos como 'cripto-mercados', ya que se caracterizan por usar como medio de pago criptomonedas como Bitcoin, Monero entre otras.

En este trabajo se desarrollará y desplegará un mercado en la red TOR como un Servicio Onion, garantizando lo mejor posible la privacidad de los compradores y vendedores, con el objetivo de comprender el funcionamiento de los servicios en la red TOR, el funcionamiento de este tipo de mercados, las relaciones y comunicaciones entre sus usuarios y el uso de criptomonedas en los pagos.

Palabras Clave: Dark Web, Red Tor, Servicios Onion, bitcoin, 'dark markets'.

Abstract

When people hear about the Deep Web, Dark Web, TOR or I2P network, they automatically think it is full off ‘disturbing’ content, that there are markets where dark and illegal content among other illicit merchandise is traded.

A ‘dark market’ on the TOR network is a website that operates on that network and to be accessed it is needed a special browser that connects to it and allows access to this service anonymously. To achieve this goal, it is used TOR network’s properties and advantages, which allows these services to be operated anonymously and guarantee the maximum possible anonymity of their users.

They have traditionally functioned as black markets, selling or conducting transactions as an intermediary related to the sale of weapons, cyber-weapons, drugs, counterfeit money and other illicit goods.

These markets are also known as ‘crypto-markets’, as they are characterized by using cryptocurrencies such as Bitcoin, Monero or others digital coins as payment.

In this work, a market will be developed and deployed in the TOR network as an Onion Service, guaranteeing the best possible privacy to buyers and sellers, with the aim of understanding the operation of the services in the TOR network, the operation of this type of markets, the relationships and communications between its users and the use of cryptocurrencies in payments.

Key Words: Dark Web, Tor network, Onion Services, bitcoin, ‘dark markets’.

Índice

Índice de Ilustraciones

1. Especificación del Trabajo

1.1 Motivación

La motivación de este trabajo, la creación y despliegue de un mercado en la red TOR, se basa fundamentalmente en conocer y profundizar en las particularidades de este tipo de servicios ocultos, su funcionamiento, características de diseño y su despliegue en la red TOR.

Son importantes las implicaciones de diseño que se deben de tener en cuenta a la hora de desarrollar estos sistemas, como pueden ser la latencia y particularidades de funcionamiento de la red TOR, las características de privacidad y seguridad que estos mercados requieren. Otros aspectos importantes que estudiar e implementar son como se realizan las transacciones económicas mediante criptomonedas y las comunicaciones entre los usuarios de estos servicios de forma privada, usando cifrado basado en infraestructuras de clave pública.

1.2 Planteamiento

Este trabajo plantea el diseño, implementación de un mercado en la red TOR, que permita el intercambio de mercancías entre compradores y vendedores garantizando su privacidad.

También debe de desplegarse correctamente en la red TOR como un Servicio Onion, mediante una arquitectura que garantice la seguridad y privacidad de los usuarios y operadores del servicio.

El servicio debe permitir comprar y vender mercancías de la forma anónima. Permitir las transacciones económicas entre los usuarios mediante bitcoins y su comunicación de forma privada mediante el uso de cifrado. También debe de estar correctamente diseñado para su correcto funcionamiento dentro de la red TOR y sus características de particulares de funcionamiento.

1.2 Objetivos

El objetivo de este trabajo es desarrollar un mercado en la red TOR totalmente operativo y funcional, con capacidad real para operar en el entorno de privacidad y seguridad de la red TOR. Conocer las características concretas que hacen de estos servicios útiles para sus operadores y ser capaz de entender y exponer los requisitos fundamentales desde el punto de vista de estos.

Plantear, desarrollar e implementar un sistema con medidas de privacidad y seguridad en el ámbito de las transacciones financieras mediante el uso de criptomonedas y comunicaciones privadas y anónimas entre los usuarios de estos servicios ocultos.

Otro de los objetivos es intentar observar y analizar el tráfico que llega al servicio, observar los accesos de los usuarios, así como patrones en el uso por parte de los usuarios.

2. Contexto, Estado del Arte y fundamentos teóricos

2.1 Deep Web. Una breve introducción.

Internet es una red, formada por la conexión de múltiples redes, que como objetivo principal tiene compartir de recursos. Es algo ya común en nuestras vidas, en todos los ámbitos, la formación, la investigación, el ocio. Su tamaño es enorme y con la llegada del Internet de las cosas (IoT) se calcula que en la actualizada hay 17000 millones de dispositivos conectados de distintos tipos (Lueth, 2018).

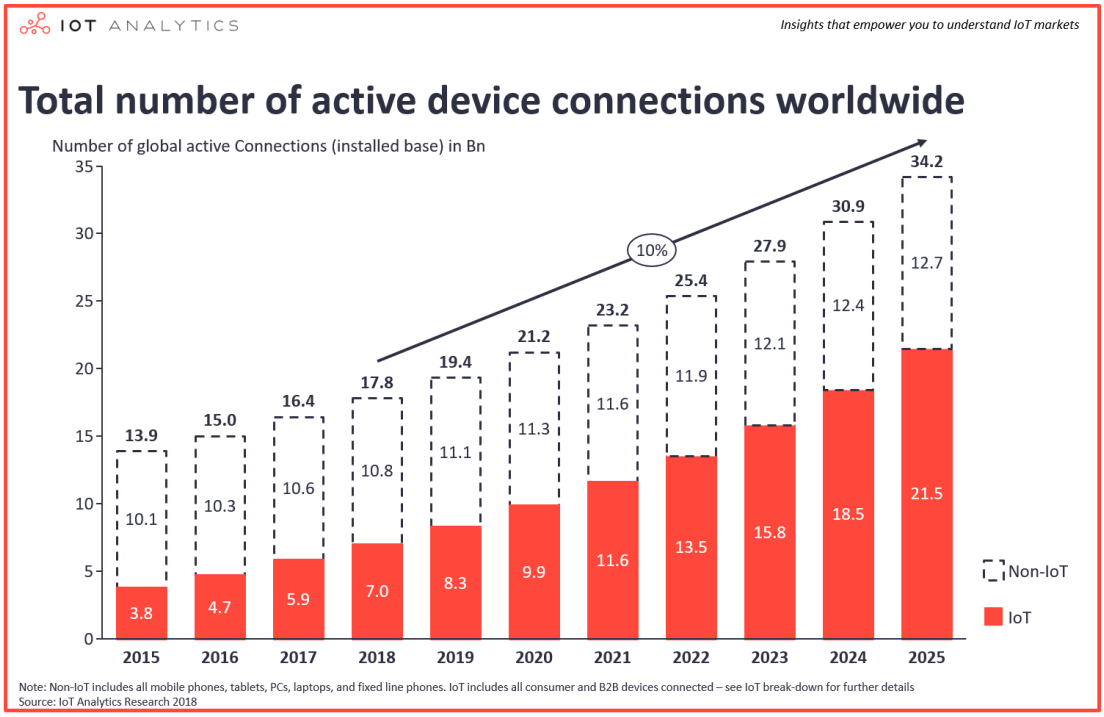

Ilustración 1 - Tendencia de crecimiento de dispositivos conectados a Internet

De estos dispositivos muchos exponen contenidos o servicios que son indexados por los motores de búsqueda como Google, Bing u otros.

A internet también se le ha descrito como un iceberg, cuando se usa esta analogía nos referimos a las capas que lo forman, en base a los que está indexado y accesible y lo que no. Se calcula, como en la analogía del iceberg, que solo entre el 4% y el 10% de la información de lo que llamamos Internet, es visible al público general. De manera oculta y bajo esa capa aparece lo que se conoce como la Deep Web, generalmente fuera del alcance de los

buscadores web en algunos casos u oculto en redes como TOR, donde solamente navegadores como TOR browser1 pueden acceder.

El Término 'Deep Web' fue acuñado por Mike Berg en el año 2000, estableciendo una comparativa entre los motores de búsqueda y la pesca en el océano. Y es que si los pescadores echan sus redes en la superficie, Google, Bing y similares hacen lo mismo con sus 'crawlers', dejando toda una fauna debajo, lo mismo pasa en Internet. (Rodriguez de Luis, 2018)

La Deep Web, la forman una telaraña compuesta por pequeñas redes P2P y otras más grandes como Freenet, I2P, Tor, entre otras. Esta zona de Internet es un lugar muy interesante para los defensores de la privacidad, que la aprovechan para proteger su anonimato. Edward Snowden utilizó la Deep Web para obtener la información que llevó a una controversia mundial. Periodistas de todo el mundo la utilizan para buscar información sensible o peligrosa y para comunicarse de forma segura y anónima (Varios Editores, 2019).

Las características de la Deep Web la han convertido en un paraíso para los delincuentes de todo tipo, traficando, desde drogas ilegales hasta tarjetas de crédito robadas e incluso pornografía infantil.

'The Silk Road', un sitio web que operaba en la red TOR, como un mercado en línea impulsado por la criptomoneda bitcoin, fue cerrado por el F.B.I.[2] en 2013 (BBC News, 2013). El sitio era conocido como uno de los principales destinos para la venta ilícita de drogas (heroína, cocaína y metanfetaminas), y su desaparición hizo que apareciesen sucesores ansiosos por capitalizar su caída.
También las agencias gubernamentales y de seguridad, utilizan estas redes, tanto por su preocupación por la piratería, el tráfico ilegal y las fugas de información confidencial y otros delitos, como para sus propias operaciones clandestinas.

2.1.1 Deep web y Dark Web: Las capas profundas de Internet

Internet es enorme y está cambiando continuamente. Cada día se añaden nuevos contenidos o servicios. Acceder a esta información y tenerla organizada de alguna forma es imposible, salvo que confiemos en los buscadores o 'search engines' como Google, Bing o DuckDuckGo. Estos servicios se encargan de indexar la red y permitirnos encontrar lo que necesitamos en

[1] TOR Browser: https://www.torproject.org/download/
[2] *Federal Bureau of Investigation*

esa ingente cantidad de información. Lo que no encontramos usando los buscadores, parece como que no existiera (Schober, 2015).

Se puede hacer un símil, imaginando una biblioteca inmensa con multitud de salas en la que una parte está catalogada pero otra parte más grande no lo está y la información o volúmenes que contiene no la podemos encontrar en esos catálogos. Es más, siguiendo con el símil, todavía hay salas que están cerradas y solo se pueden acceder de formas muy particulares.

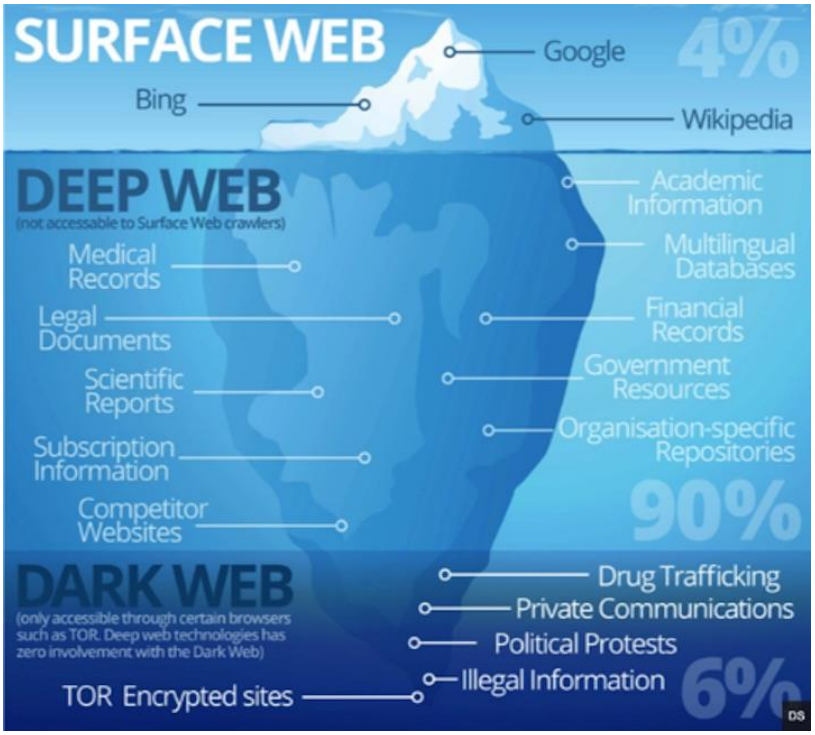

Ilustración 2 - Capas de la Deep Web

Las capas en las que se puede dividir la web en base a su acceso es la siguiente (OWL - El blog del lado Oscuro, 2017):

Clear Web. - Es el primer nivel, el que usa la inmensa mayoría de los usuarios. Esta capa es la que los 'crawlers' de los buscadores tradicionales analizan para ofrecernos resultados. Se calcula que representa el 10% de lo que es Internet. Lo más importante desde este punto de vista es que esta información es pública, está accesible sin ningún tipo de restricción, en la mayoría de los países. Algunos países censuran incluso esta parte de Internet (Hochstadt, 2019).

Deep Web. - Es el nivel que los indexadores tradicionales no pueden acceder para obtener su contenido, debido a múltiples factores. Estos factores son:

- **Documentos o información oculta**: PDFs que no se encuentran en las páginas indexadas, listas de datos no públicas (sobre todo los ciber-criminales) ...
- **Web contextual**: páginas cuyo contenido varía dependiendo del contexto (por ejemplo, la dirección IP del cliente, usuario que ha iniciado sesión).
- **Contenido dinámico**: páginas dinámicas obtenidas como respuesta a parámetros, por ejemplo, datos enviados a través de un formulario.
- **Contenido de acceso restringido**: páginas protegidas con contraseña, contenido protegido por un Captcha, etc.
- **Contenido No HTML**: contenido textual en archivos multimedia, otras extensiones como exe, rar, zip, etc.
- **Software**: Contenido oculto intencionadamente, que requiere un programa o protocolo específico para poder acceder (ejemplos: Red Tor, I2P, Freenet)
- **Páginas no enlazadas**: páginas de cuya existencia no tienen referencia los buscadores; por ejemplo, páginas que no tienen enlaces desde otras páginas.
- **Dark Web. -** Es la capa más profunda de la Deep Web, si la Deep Web es aproximadamente el 90% de Internet, la Dark Web ocuparía sobre el 1% o menos de la misma. Su acceso es restringido y oculto intencionadamente a los motores de búsqueda. En esta capa podemos encontrar diferentes darknets[3], que son redes que solo se pueden acceder con programas específico, como la red TOR, Freenet, ZeroNet o I2P. Suelen almacenar contenido ilegal como drogas, venta de armas, terrorismo, mercados negros, malware, entre otras, pero también información útil que no es dañina, pero que puede haber sido censurada en algunos lugares y utilizada por activistas.

Como se ha indicado, la Dark web es una parte pequeña de la Deep web y su contenido se oculta de forma deliberada. Se utilizan dominios propios que en la 'Clearnet'[4] los servidores DNS no pueden resolver como '.onion', '.i2p', etc. Su acceso solo es posible mediante programas especiales como por ejemplo TOR.

En este trabajo, nos centraremos en la red TOR, en esta red encontramos sitios y servicios ocultos antes denominados "*hidden services" y actualmente se denominan "onion services"*. (Lavín, 2018)

[3] https://es.wikipedia.org/wiki/Darknet
[4] https://en.wikipedia.org/wiki/Clearnet_(networking)

2.2 La red TOR

2.2.1 Qué es la red TOR.

TOR o **T**he **O**nion **R**outer (Enrutamiento Cebolla), es una red distribuida que trabaja superpuesta sobre internet. Su principal objetivo es garantizar lo mejor posible, *la privacidad y el anonimato* de quien la use.

TOR protege la identidad de sus usuarios, encriptando su tráfico mediante un mínimo de tres capas, formadas por computadoras de voluntarios por todo el mundo. El primer nodo, recibe información desde la ubicación del usuario, pero solamente sabe eso, y la tiene que enviar a otro nodo, y así sucesivamente.

Esta información va cifrada entre cada salto de nodo, por lo que no se puede saber su contenido. El siguiente nodo sabe que recibe información del anterior y que tiene que enviárselo a un tercero, pero no sabe de donde proviene esa información originalmente, ni qué contiene. Así va pasando por nodos, con un mínimo de tres, hasta que finalmente la información llega a su destino sin desvelar la dirección IP de origen.

El enrutamiento cebolla, que es el principio fundamental en el que se apoya la red TOR para garantizar el anonimato y la privacidad de sus usuarios y proveedores de servicios.
En Internet, normalmente se usa un enrutamiento directo, es decir va por un camino directo, aunque no siempre el mismo, por ejemplo, para ir desde un navegador hasta un sitio web concreto. Además, cualquiera que esté escuchando la conversación entre el equipo que inicia la comunicación y el servidor web de destino, sabrá perfectamente como mínimo, nuestra IP y la del destino.

La red TOR, sin embargo, utiliza un enrutamiento indirecto, utilizando varios nodos, de forma aleatoria.

TOR (The Onion Router) envía los paquetes de información por varios nodos intermedios, como ya se ha indicado. Primero se calcula una ruta pseudo aleatoria hacía del destino, se obtienen las claves públicas de cada uno de estos nodos.

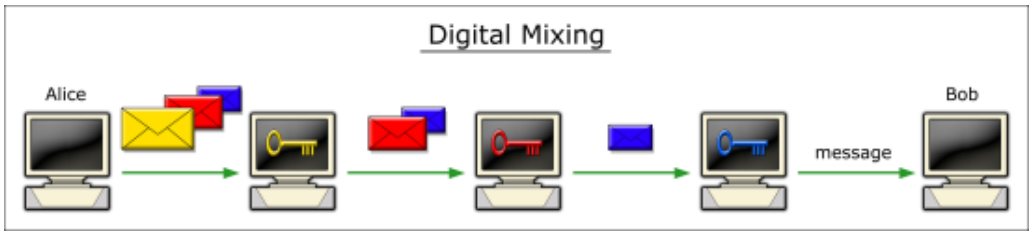

Ilustración 3 - Onion Routing, cifrado en capas

Después, el paquete se cifra usando esas claves públicas de cada nodo forma progresiva, semejando las capas de una cebolla. Primero se cifra el paquete que contiene el mensaje, el destino y ruta a usar. En cada nodo se va cifrando el paquete, añadiendo capas, usando las claves públicas de los nodos intermedios, que se obtuvieron previamente. Estas capas se van deshaciendo, por los nodos, usando sus claves privadas. En el penúltimo nodo, este descifra la última capa del paquete para encaminarlo hacia el servidor de destino. (Ardións, 2017)

2.2.2 TOR: Un poco de historia.

El proyecto TOR fue concebido en el año 1995, por David Goldschlag, Mike Reed, y Paul Syverson que en ese momento estaban trabajando en el U.S. Naval Research Lab (M. Goldschlag, G. Reed, & F. Syverson).

Tenían la intención de crear conexiones de Internet que fueran seguras y privadas. Quieran construir un sistema en el que usando Internet no se pudiera saber quién se estaba comunicando con quien, incluso en un entorno en el que la red estuviera monitorizada. Esto dio lugar a la aparición de los primeros prototipos de “enrutamiento de cebolla” (M. Goldschlag, G. Reed, & F. Syverson, Anonymous Connections and Onion Routing, 1997).

Más tarde se fueron uniendo otros personajes destacados al proyecto, como Roger Dingledine y Nick Mathewson. Es en ese momento es cuando se empezó a hablar del proyecto TOR (The Onion Routing), en octubre del año 2002.

El planteamiento de la red TOR, es usar una red descentralizada, operada por entidades con intereses dispares y con suposiciones de confianza entre las partes. El software usado en su construcción debe de ser libre y abierto.

A finales del año 2003, la red TOR tenía solo una docena de nodos voluntarios, los cuales se encontraban en Estados Unidos, y uno en Alemania. La Electronic Frontier Foundation (EFF)[5] comenzó a financiar el proyecto y a estos investigadores a partir del año 2004, reconociendo el beneficio de la red TOR para los derechos digitales. Es en el año 2006 cuando su funda El Proyecto TOR[6], como una organización sin ánimo de lucro, y con el objetivo de mantener el desarrollo del proyecto (The TOR Project, s.f.).

En ese momento es cuando TOR empezó a ganar popularidad entre los activistas y usuarios expertos en tecnología con interés en la privacidad. Para personas con menos conocimientos técnicos, todavía era difícil su configuración, había que configurar el proxy TOR correctamente, tarea que no era trivial.

En el año 2008 se comenzó a desarrollar el navegador TOR (TOR Browser[7]), lo que hizo que TOR fuera mucho más accesible para los usuarios de todo tipo (Murdoch, 2008).

Hoy en día, la red TOR tiene millones de usuarios y miles de nodos operados por voluntarios.

El proyecto ha sido atacado continuamente con la intención de hackearlo, romperlo, por entes de todo tipo, como agencias de seguridad gubernamentales o entidades con intereses ocultos y en cada ocasión el proyecto se ha recompuesto.

En 2019 se filtró que un grupo de hackers conocidos como 0v1ru$ habían conseguido aprovechar una brecha masiva en SyTech, uno de los principales contratistas de seguridad que trabaja para el FSB Ruso (Servicio Federal de Seguridad)[8], centrado en labores de contrainteligencia, espionaje, vigilancia y de seguridad interna. El ataque que se llevó a cabo el 13 de julio de 2019, logró recuperar una ingente cantidad de información sobre proyectos internos. Uno de esos proyectos era un intento de hackeo de TOR browser. La intención era des anonimizar a los usuarios de esta red (BBC News, 2019).

Actualmente, TOR, sigue siendo una de las mejores formas de anonimato en Internet, acceso a la Deep Web y es relativamente sencillo de usar.

[5] https://www.eff.org/
[6] https://www.torproject.org
[7] https://www.torproject.org/download/
[8] https://es.wikipedia.org/wiki/Servicio_Federal_de_Seguridad

2.3 Funcionamiento de TOR y comparación con la navegación clásica de Internet.

Para comprender el funcionamiento de la red TOR es importante entender como viaja la información por es esta red, ya que no funciona como lo hace la Internet clásica.

Si alguien quiere acceder a un servicio, como una web, en Internet, el usuario abre un navegador en su equipo, introduce la dirección a la que quiere ir y ya es suficiente. Luego ya entra en juego, los sistemas clásicos de DNS, enrutado y demás para que se pueda obtener la web correspondiente y ofrecérsela al usuario (Derechos Digitales América Latina, 2018).

Acceder de esta forma a Internet es perfecta para la mayoría de los usuarios, no así para los preocupados por su privacidad. Esta forma de acceder a Internet tiene sus desventajas desde el punto de vista de la privacidad, como pueden ser:

El servidor de destino puede saber desde donde lo estamos visitando, país, ciudad, ISP.
Se la conexión está intervenida por un tercero, ese tercero puede obtener la misma información, y en algunos casos, como si no se está utilizando TLS/SSL para la comunicación, toda la información transmitida.

El siguiente gráfico, indica como navegaríamos hasta un sitio web usando TLS/SSL, que es lo más habitual actualmente. Si no fuera el caso, la comunicación no iría cifrada y se podría obtener por todas las partes todos los datos comunicados, de ahí la importancia de usar TLS/SSL en el acceso a la web.

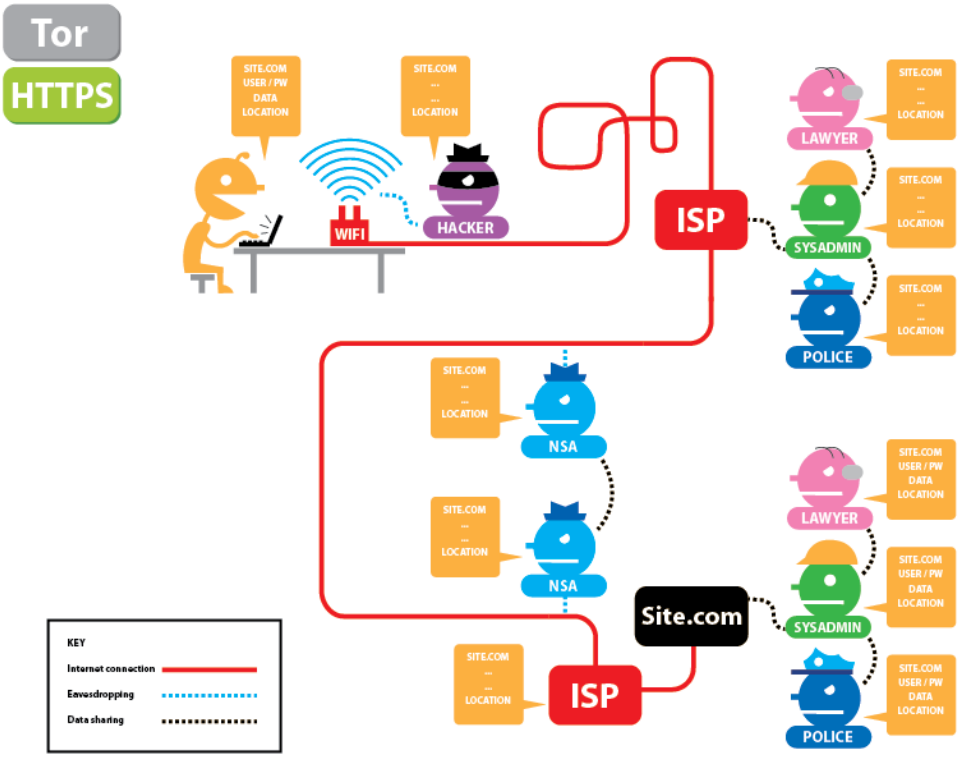

Ilustración 4 - Acceso a sitio web Internet con TLS (Electronic Frontier Foundation)

Cuando se utiliza la red TOR, la comunicación se realiza de forma que antes de llegar a destino se ponen por medio y de forma aleatoria, al menos tres nodos (o relays[9]).

El cliente de la red TOR (origen) prepara el mensaje a enviar por el circuito elegido (conjunto de nodos), para ello, el mensaje se cifra por capas como si de una cebolla se tratase. El origen cifra el mensaje con la clave pública del último nodo de la ruta para que solo ese nodo pueda descifrarlo, además de incluyen las instrucciones cifradas para llegar al destino. Después se vuelve a cifrar todo el paquete con la clave pública del penúltimo nodo, de forma que así solo lo puede descifrar el penúltimo nodo de la ruta y este proceso se repite de igual forma para todos los nodos de la ruta.

Cuando el mensaje llega al primer nodo lo descifra con su clave privada y sigue las instrucciones para enviarlo al siguiente nodo, que realizará la misma tarea, quitando capas en cada nodo hasta llegar al nodo de salida que descifrará el paquete con su clave privada y podrá leer el mensaje enviado.

[9] Más adelante en este trabajo se describen los tipos de nodos posibles en la red TOR y su función, para más información: https://trac.torproject.org/projects/tor/wiki/TorRelayGuide#TheTorRelayGuide

El cifrado en capas propuesto en el enrutamiento cebolla se puede observar de forma gráfica en la siguiente imagen.

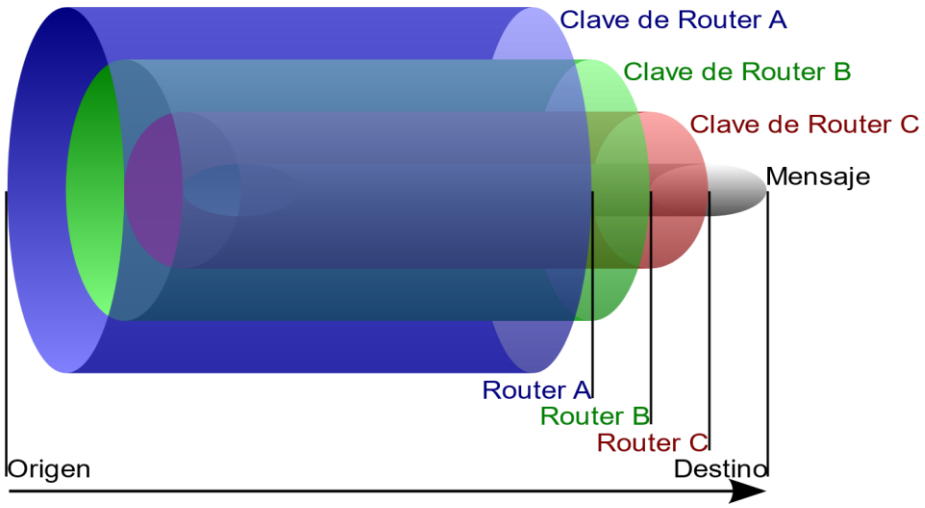

Ilustración 5 - Cifrado en capas del enrutamiento cebolla

Este modelo de comunicación permite que, si alguien está monitorizando la conexión, sea muy difícil que identifica a los extremos de esta.

Se puede apreciar la diferencia en la siguiente imagen en contrapunto con la expuesta anteriormente en una comunicación normal en Internet usando TLS.

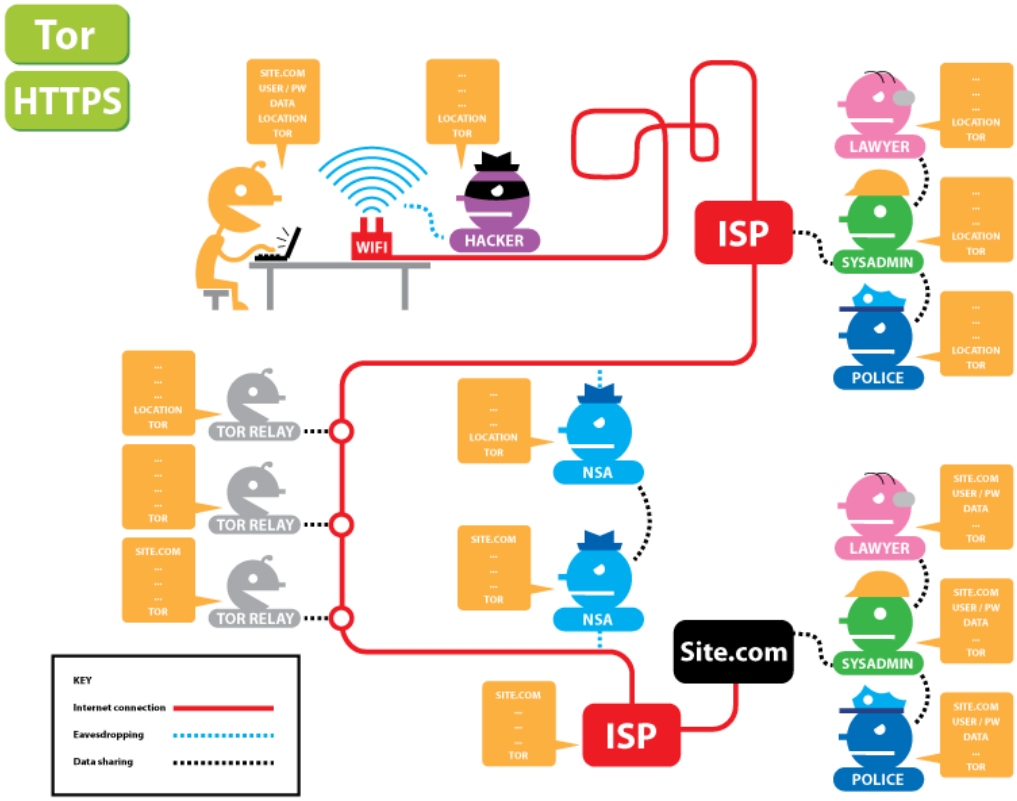

Ilustración 6 - Acceso con TOR a un sitio web con TLS (Electronic Frontier Foundation)

En el caso que usemos TOR solamente para acceder a servicios onion o hidden services, el último paso del gráfico no se produce, es decir no volvemos a Internet, nos quedamos en la red TOR con lo que la privacidad aumenta.

Cada nodo solamente sabe que recibe la información de un nodo y que la tiene que enviar a otro, pero no sabe nada más ni de los nodos anteriores, ni posteriores ni por supuesto del contenido del mensaje. Solo quien hace la solicitud puede saber qué nodos han participado en el circuito.

El mínimo de nodos por los que se salta es tres, pero se debe de indicar a más número de nodos, más rendimiento se pierde, debido a que la necesidad de proceso y el número de saltos aumenta.

Por otro lado, solo se usarán dos nodos, un atacante que analiza el tráfico del nodo de salida podría conocer fácilmente el nodo de entrada y por ende al usuario conectado y sus datos.

Se procede ahora a explicar en detalle el proceso de comunicación entre dos equipos A (Alice) y B (Bob) que se comunican y envían un mensaje, usando la arquitectura de "enrutado cebolla" u Onion Router.

El equipo A se conecta a la red TOR y realiza una conexión a un servidor central que contiene las direcciones de los nodos Tor, estos nodos son públicos y están listados. Con la lista de direcciones A se conectará a un nodo elegido aleatoriamente a través de una conexión cifrada. Después se escogerá otro nodo aleatorio con otra conexión cifrada y, así hasta llegar al nodo Bob.

El nodo de salida hará una conexión no encriptada con Bob. Es importante indicar que los nodos Tor que forman el circuito se eligen al azar y ningún nodo puede ser utilizado dos veces, además el circuito entre A y B cambia a los 10 minutos de uso.

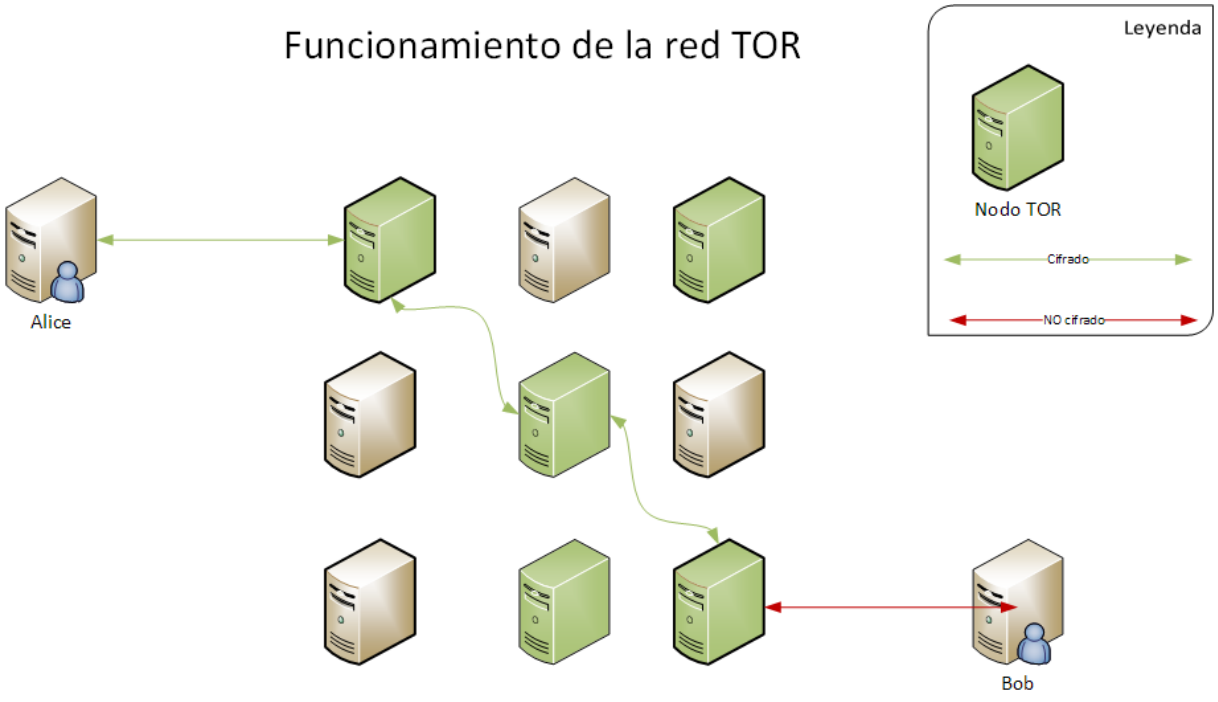

Ilustración 7- funcionamiento Onion Router en TOR

El proceso de enrutado cebolla, usado por la red Tor, en más detalle, pero sin complicarlo demasiado y utilizando 3 nodos de ejemplo, se podría enumerar en los siguientes pasos:

El cliente A tiene las tres claves de encriptación K1, k2 y k3 y encripta el mensaje tres veces, como una cebolla, que luego se irán quitando una cada vez. El mensaje encriptado tres veces, se envía al primer nodo del circuito.

El nodo 1 solamente sabe la clave k1, así que descifra el mensaje y obtiene a que nodo tiene que enviarlo, el nodo 2, pero no entiende el mensaje, ya que sigue encriptado 2 veces, con k2 y k3. En este momento, se lo envía al nodo 2. El nodo 2 tiene la clave k2, así que descifra la segunda capa, todavía queda una capa de cifrado) y se lo envía al nodo 3 o de salida. El nodo 3, que es el nodo de salida, usa su clave k3, para quitar la última capa del mensaje y lo descifra completamente y lo envía al destino ya sin cifrar.

La respuesta sigue el mismo sistema, pero con el camino inverso.

2.3.1 Componentes fundamentales de la red TOR.

La red TOR, como cualquier otro sistema, requiere de una serie de componentes con diferentes roles. Para funcionar correctamente. Los componentes principales son los nodos o relays.

El anonimato en TOR funciona transmitiendo datos a través de un '**circuito'**, que es el componente principal de funcionamiento del mecanismo de enrutado cebolla.

Un **circuito TOR** es la combinación de una serie de nodos o relays en la red TOR, cado uno con un rol específico y una posición concreta. De esta forma, un circuito en la red TOR está compuesto por un **nodo de entrada o guardia**, un **nodo intermedio** y un **nodo de salida** (The TOR Project, 2018). En algunos casos aparece otro tipo de nodo, el **nodo puente**. Este tipo de nodo solo aparece cuando se sabe que los nodos de entra y salida están bloqueados por terceras partes como gobiernos u organizaciones. Es en estos casos cuando se usan los nodos puente para circunvalar el bloqueo. Todos estos nodos son importantes, pero tienen distintos requisitos técnicos e implicaciones legales.

Nodo de Entrada o Guard relays: Estos nodos se comunican con los clientes TOR y los conectan a la red TOR. Es el primer nodo de los que forman el circuito. Para que un nodo funcione correctamente como guarda o nodo de entrada, debe de ofrecer una conexión estable y rápida de al menos 2 MByte/s.

Nodo medio o Middle relays: Son los siguientes nodos del circuito en la red TOR y solo se comunican con otros nodos, de forma que su tráfico nunca sale de la red TOR. Sus requisitos de ancho de banda no son tan estrictos como el en caso de los nodos de entrada o salida. Los nodos intermedios no suelen tener problemas desde el punto de vista legal, pero pueden ser bloqueados como cualquier tipo de nodo debido a que todos los nodos son públicos y están listados en la lista de relays TOR[10].

Nodo de Salida o Exit Relays: Este tipo de nodos son los últimos del circuito, toman las solicitudes que les llegan y las hacen llegar a su destino, después reciben las respuestas y las envían de vuelta hacia el solicitante. Son los que conectan con los servicios que el solicitante quiere acceder con garantías de anonimato (una página web, chat o cualquier otra cosa). Estos nodos son los que están más expuestos a problemas legales, ya que son el punto desde el que se hacen las solicitudes finales. Suelen ser mantenidos por instituciones y otras entidades con capacidad para enfrentar las posibles consecuencias legales del uso que den los usuarios cuyas conexiones salen por estos nodos.

Nodos de Puente o Bridge Relays: Por diseño, la red TOR publica la dirección IP de todos los nodos. Esto permitiría que, si se quiere bloquear el acceso a la red TOR, bloqueando todas las direcciones IP de los nodos esta quedaría inaccesible. Los nodos puente no están listados en el directorio público de nodos TOR, lo que hace que sean más complicado su bloqueo. Un listado reducido de los nodos puente se entrega a los clientes para evitar que sean bloqueados. Algunos países, entre ellos China e Irán, saben cómo detectar y bloquear los nodos puente, por lo que el proyecto TOR a añadido 'transportes conectables'[11] que son un tipo especial de nodos puente, que añaden una capa de ofuscación adicional haciendo que el tráfico que pasa por ellos no parezca de la red TOR.

Estos nodos son fáciles de operar no requieren un gran ancho de banda para funcionar, pero son fundamentales para los usuarios que bien en zonas donde se aplica la censura y el bloqueo a la red TOR.

Otro componente fundamental de la red TOR son los **Servicios de Directorio**, que son un conjunto de aplicaciones que almacenan y gestionan la información sobre los usuarios de esta red. En TOR este servicio, publica una base de datos donde se indica cada nodo o relay de

[10] Listado de nodos TOR: https://www.eff.org/torchallenge/list.html
[11] https://www.torproject.org/docs/pluggable-transports.html.en

la red e información asociada al mismo. Realmente estos servicios de directorio son un conjunto de nodos en los que se confía y que realizan esta función.

2.3.2 Algoritmos criptográficos en TOR.

La red TOR para garantizar el anonimato, se basa completamente en algoritmos criptográficos. Cuando se ha estado hablando del enrutado cebolla, se ha hablado de cifrado con clave pública de los mensajes, también se debe indicar que la comunicación entre nodos debe de estar protegida y por lo tanto se usan también algoritmos de cifrado tanto simétricos como asimétricos.

Aquí simplemente se abordará de forma muy básica la información sobre estos algoritmos, lo suficiente para permitir y entender las implicaciones de correr un servicio onion, que es el objetivo de este trabajo (The TOR Project, 2019).

Las conexiones entre dos nodos TOR, o entre un cliente y un nodo usan TLS/SSLv3 para autenticación y encriptación. La suite criptográfica que como mínimo se debe de soportar es "SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA" y deberían tener disponible "TLS_DHE_RSA_WITH_AES_128_CBC_SHA", pero si están disponibles mejores alternativas, estas mejoras se deberían utilizar. Existe una lista propuesta fija de suites criptográficas a usar, que los clientes pueden ofertar, cuando negocian la comunicación.

Los clientes para conectarse con los nodos pueden usar principalmente las siguientes suites criptográficas:
TLS_DHE_RSA_WITH_AES_256_CBC_SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA

En cuanto a algoritmos de cifrado simétrico, se usa AES-128[12] en counter-mode[13] (AES-CTR), con un vector de inicialización con todos sus valores a 0. También se requiere AES256.

En el caso de **cifrados con clave pública**, se requería RSA[14] con claves de 1024 bits con exponente fijo 65537 y con relleno OAEP-MGP1, el algoritmo usado como función resumen

[12] AES: https://www.boxcryptor.com/es/encryption/
[13] http://www.crypto-it.net/eng/theory/modes-of-block-ciphers.html
[14] RSA: http://www.criptored.upm.es/crypt4you/temas/RSA/leccion0/leccion00.html

era SHA-1 (también se usa SHA256 y SHA3-256 en algunos lugares). Se pueden usar otras alternativas, si así se especifica.

Desde la versión 0.3.0.6 los clientes y los nodos usan claves Ed25519[15] para autenticar el enlace en vez de RSA 1024. (The TOR Project, 2017)

Para establecimiento de claves usa DH (Diffie-Hellman) con g=2 y para p usa el primo seguro de 1024 bits obtenido de RFC 2409[16]
Son en muchos casos requisitos mínimos y se pueden usar alternativas más robustas si la negociación entre los nodos y clientes lo permite.

Además, según avanza la especificación a nuevas versiones TOR, lo requisitos criptográficos van aumentando. Este es el caso es el caso de la especificación del protocolo 'rendezvous' en su versión 3, que indica todo lo referente al diseño y acceso a Servicios Onion v3.

En la parte de la especificación sobre mejoras sobre las versiones anteriores en el protocolo (The TOR Project, 2019) en el punto '**0.1 Improvements over previous versions'** se indica que se ha mejorado la criptografía con versiones más robustas, en concreto indica que se ha sustituido el conjunto **SHA1/DH/RSA1024** por uno más moderno y robusto, con mejor rendimiento y seguridad: **SHA3/ed25519/curve25519**.

Así, según va evolucionando el proyecto, los requisitos criptográficos se van adaptando a las nuevas exigencias de seguridad y privacidad.

2.3.3 Herramientas útiles en la red TOR

Para usar la red TOR, ya se ha indicado que se necesitan herramientas cliente, como TOR Browser, para ingresar en la red, pero existen multitud de herramientas con otros propósitos. Se va a enumerar alguna de ellas (The TOR Project, 2019). Existe muchísimas más con distintos fines y utilidades.

- **TOR Browser**. Navegador web que contiene todo lo necesario para conectarse a la red. Basado en el navegador Mozilla Firefox ESR con algunas extensiones instaladas

[15] High-speed high-security signatures: https://ed25519.cr.yp.to/ed25519-20110926.pdf
[16] https://tools.ietf.org/html/rfc2409

por defecto, TorButton, TorLauncher, NoScript, HTTPS Everywhere, TOR proxy. Disponible para prácticamente todos los Sistemas Operativos actuales (Windows, MacOS, Linux y Android). El navegador inicia de forma automática los procesos de Tor en segundo plano y enruta el tráfico a través de la red TOR de forma automática. Cuando finaliza una sesión, borra los datos de navegación de esa sesión.

- **Orbot** (Guardian Project, s.f.). Es una aplicación para dispositivos Android, ha sido desarrollada juntamente con *Guardian Project*. no tiene incorporado un navegador, simplemente redirige el tráfico a la red TOR de las aplicaciones que se seleccionen.

- **Nyx** (Tor Nyx, 2019). Es un monitor de red para TOR, basado en línea de comandos y desarrollado en Python. Permite obtener información detallada en tiempo real sobre el nodo en el que se ejecuta. Ancho de banda, conexiones, logs son algunas de las informaciones que se pueden obtener.

- **Whonix** (Whonix: A High Security Method of Surfing the Internet, s.f.). Sistema operativo de escritorio diseñado para seguridad y privacidad avanzadas basado en Debian, altamente reconfigurada, se ejecuta dentro de múltiples máquinas virtuales (pasarela y estación de trabajo), proporcionando una capa sustancial de protección contra malware y fugas de direcciones IP en el uso de la red TOR.

- **Tails** (boum.org, s.f.). Es un sistema operativo Linux de tipo ‘live’ que encapsula todo el tráfico de la red Tor para preservar la privacidad. Pude ser iniciado casi desde cualquier computadora usando un USB o DVD de arranque. Permite usar el internet anónimamente, para evitar la censura, no deja traza o rastro en el computador que se lo inicia, usa herramientas modernas de encriptación para cifrar archivos, correos y mensajes.

- **Metrics Portal** (The Tor Project, 2009-2018)**.** Analíticas de todo tipo para la red TOR, desde ancho de banda, base du usuarios estimada, tráfico, servicios onion etc…

- **Pluggable Transports** (The Tor Project, 2019). Los transportes conectables (PT) transforman el flujo de tráfico Tor entre el cliente y un nodo puente, mediante ofuscación. De esta forma, terceras partes que monitorean el tráfico entre el cliente y el puente verán un tráfico transformado de aspecto inocente en lugar del tráfico real

de Tor. Se usan en situaciones en los que terceras partes quieren bloquear las comunicaciones mediante Tor.

- **Stem** (The Tor Project, n.d.). Librería Python que permite interactuar con Tor. Usando Stem, se pueden escribir scripts y aplicaciones que interactúan con Tor. Es realmente una implementación de las especificaciones de directorio y control de la red Tor.

2.4 Servicios Onion

El objetivo de este trabajo es crear un mercado en la red Tor. La forma más adecuada es creando este mercado online como un servicio Onion y exponerlo únicamente dentro de la red TOR.

2.4.1 ¿Qué es un servicio Onion?

Los servicios onion o servicios cebolla, (en este trabajo mantendremos la nomenclatura de servicio onion), son servicios que solo pueden ser accedidos a través de la red Tor. Estos servicios, puede ser cualquiera a los que estamos acostumbrados a usar en Internet, es decir un sitio web con información de cualquier tipo, un servicio de chat o de correo.

Originalmente, estos servicios se llamaron “hidden services, y fueron renombrados por “onion services” en 2015, para reflejar el hecho que proporcionan más que un servicio oculto. Estos servicios proporcionan, por ejemplo, seguridad extremo-a-extremo y nombres de dominio generados automáticamente usando criptografía de clave pública.

Los servicios onion ofrecen algunas ventajas sobre los servicios ordinarios, de cara a la privacidad. (The TOR project, s.f.)

- La dirección IP y la ubicación de un servicio onion están ocultos. Esto dificulta que un adversario pueda bloquearlo identificar a sus operadores.

- El tráfico entre los clientes Tor y los servicios onion está completamente cifrada extremo a extremo. No es necesario por lo tanto el uso de TLS/SSL

- La dirección de un servicio onion, se genera automáticamente. No es necesario comprar un nombre de dominio. Las URL de los servicios onion (.onion) ayudan a Tor a asegurarse de que se está conectado a una ubicación correcta y que la comunicación no se está alterando.

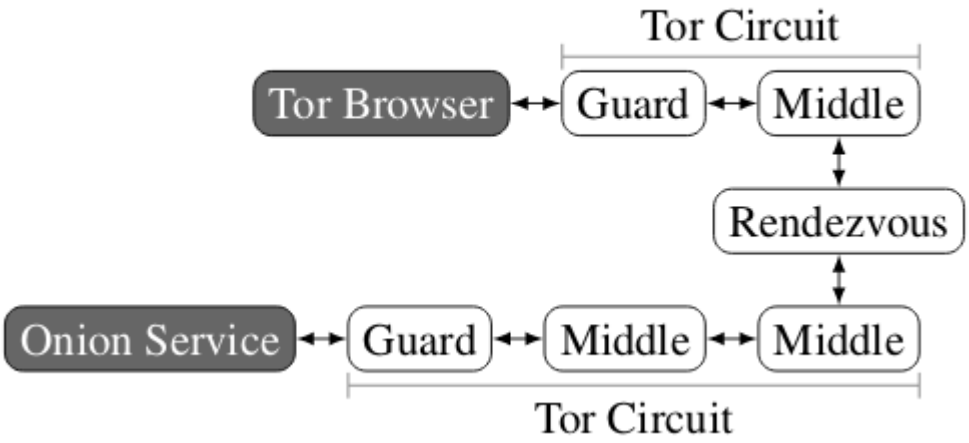

Ilustración 8 - Comunicación entre Tor Browser y un servicio onion

Estos servicios son servicios de red, basados en TCP, solo accesibles desde la red Tor, y proporcionan anonimidad mutua a ambos extremos de la conexión, es decir al servidor y a los clientes conectados. Los clientes acceden a los servicios onion, a través de URLs '.onion', que solamente tienen sentido dentro de la red Tor. (Winter, y otros, 2018)

El camino entre un cliente, como puede ser Tor browser, hasta un servicio onion, necesita seis nodos Tor por defecto. El cliente construye un circuito hasta un nodo Tor "rendezvous" (punto de encuentro), es decir, cliente y servidor se 'citan' en un nodo Tor. El servidor construye otro circuito hasta el mismo nodo donde se han 'citado'. Ambos extremos se comunican de esta forma mediante el nodo elegido, y ninguna de las partes conoce por lo tanto la IP de la otra.

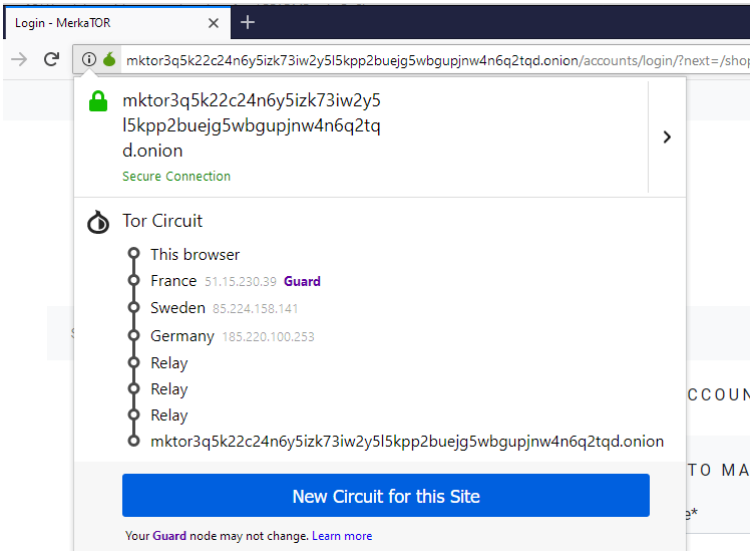

Ilustración 9 - Cominucación Tor Browser - Servicio Onion MarkeTOR

Se puede apreciar el modelo de comunicación utilizado en la comunicación entre un cliente Tor y un servicio '.onion', en la anterior imagen. Se usa TOR Browser para conectar al servicio .onion MarkeTOR, creado en este trabajo, se pueden ver los nodos que se usan en la comunicación.

2.4.2 Nombres de dominio ONION

La red Tor tiene la característica de alojar servicios de red usando el seudónimo de nivel superior '.onion', lo que indica que un servicio con una URL de ese tipo tiene una IP anónima y solo accesible mediante la red TOR. Se usan como cualquier otro nombre de dominio, pero no utilizan la infraestructura DNS. Los nombres de dominio '.onion' tienen la funcionalidad combinada de identidad del servicio, ubicación y autenticación. (Dingledine, Mathewson, & Syverson, 2004).

Los nombres de dominio '.onion' se utilizan para proporcionar acceso a cifrado de extremo a extremo, servicios seguros y anónimos; es decir, la identidad y ubicación del servidor está oculto al cliente que se conecta.

Estos nombres de dominio se autentican automáticamente, ya que se derivan de claves criptográficas utilizadas por el servidor, y que son verificables por los clientes en el establecimiento de la conexión.

Otra característica importante, es que la red Tor está diseñada para no estar sujeta a ninguna autoridades de control central, con respecto a enrutamiento y la publicación de servicios, por lo que los nombres de dominio '.onion', a diferencia de los DNS no se pueden registrar, asignar, transferir o revocar. La "propiedad" de un nombre de dominio '.onion' se deriva únicamente del par de claves pública y privada que corresponde de la que se derivó algorítmicamente el nombre (Appelbaum, 2015).

2.4.3 Direcciones '.onion'. Estructura y Formato

Las direcciones ".onion" son opacas, no mnemotécnicas, con una longitud de 16 o 56 caracteres semi alfa-numéricos, generados automáticamente y basados en la clave publica usada en la configuración del servicio ".onion".

Los nombres pueden estar formadas por las letras del alfabeto, y los dígitos decimales del 2 al 7, y representan en base 32, que representan:

En la versión 2: Un hash de 80 bits. Direcciones de 16 caracteres.

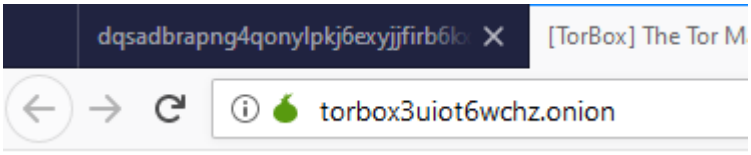

Ilustración 10 - Dir. Onion v2 -16 caracteres.

En la versión 3: Una clave pública ed25519. Direcciones de 56 caracteres.

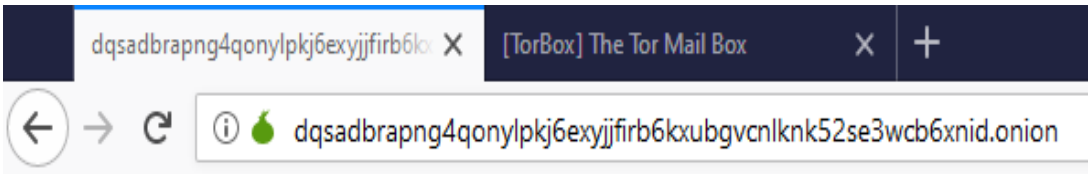

Ilustración 11 - Dir. Onion v3 - 56 caracteres.

2.4.5 Comunicación cliente Servidor Onion.

Cuando un cliente de la red Tor se comunica con un servicio onion, lo hace de un amanera especial, que garantiza el anonimato de ambos extremos. Se utiliza para ello los **puntos de cita o "rendezvous points"** (Dingledine, Mathewson, & Syverson, 2004).

Un servicio onion necesita anunciar su existencia en la red Tor antes de que los clientes puedan conectar con él. El servicio selecciona aleatoriamente algunos nodos), y crea circuitos (túnel formado por 3 nodos) hasta esos nodos, luego les pide que actúen como puntos de introducción indicándoles su clave pública.

Al usar un circuito Tor completo, hasta cada punto de introducción, es difícil para cualquiera asociar un punto de introducción con la dirección IP del servidor de onion. Es decir, los **puntos de introducción** conocen la identidad del servicio onion, al conocer su clave pública, pero no conocen ni su localización ni su IP. También es importante indicar que los **puntos de introducción** no intervienen en la comunicación final entre el cliente y el servicio onion (The TOR Project, 2019).

Este primer paso de la comunicación se puede apreciar en el siguiente esquema:

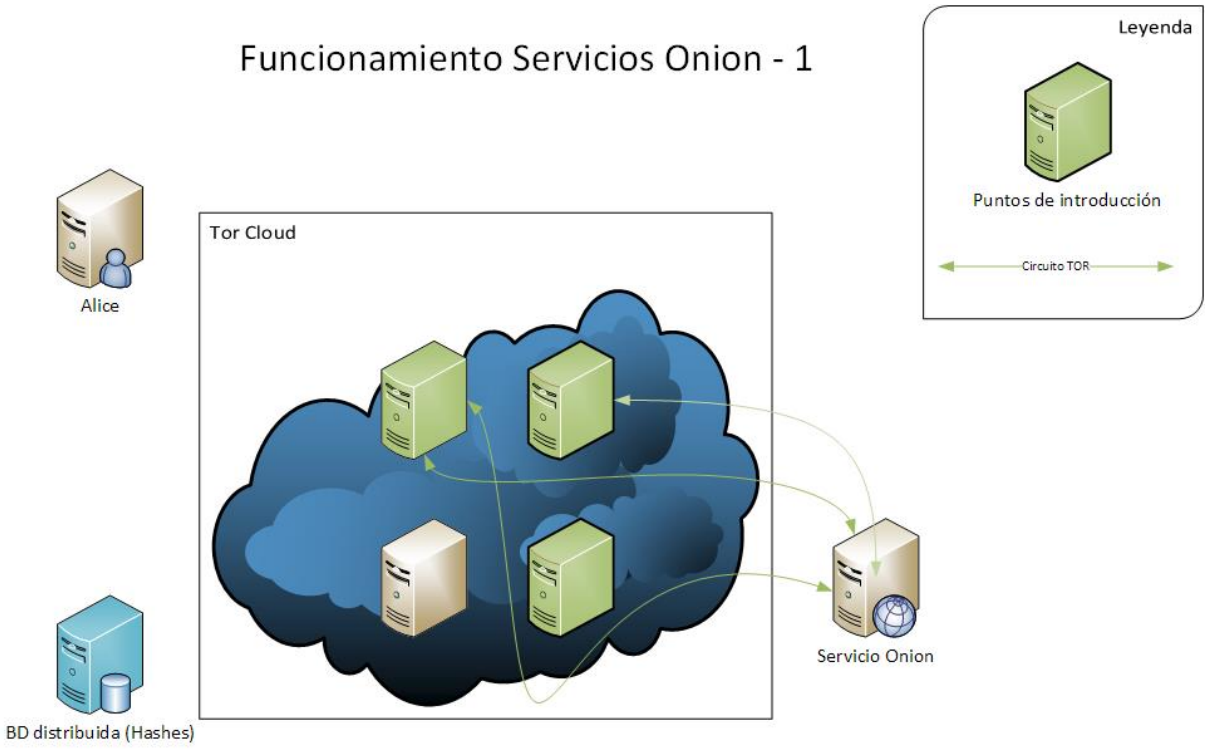

Ilustración 12 - Comunicación con servicios onion - 1

El servicio también construye un “descriptor de servicio onion”, que contiene:

- La clave pública del servicio.
- Información sobre los puntos de introducción elegidos
- Firma digital con su clave privada del descriptor

El descriptor creado por el servicio se sube a una **base de datos distribuidas de hashes**. Esta base de datos distribuida es la que permite encontrar los servicios a los clientes, cuando piden una dirección “.onion”. En este momento, el servicio ya está listo para aceptar peticiones.

Este sistema, cumple un objetivo importante: todas las partes implicadas, los puntos de introducción, el directorio o base de datos de la tabla hash distribuida y, por supuesto, los clientes, pueden verificar que están hablando con el servicio de cebolla correcto.

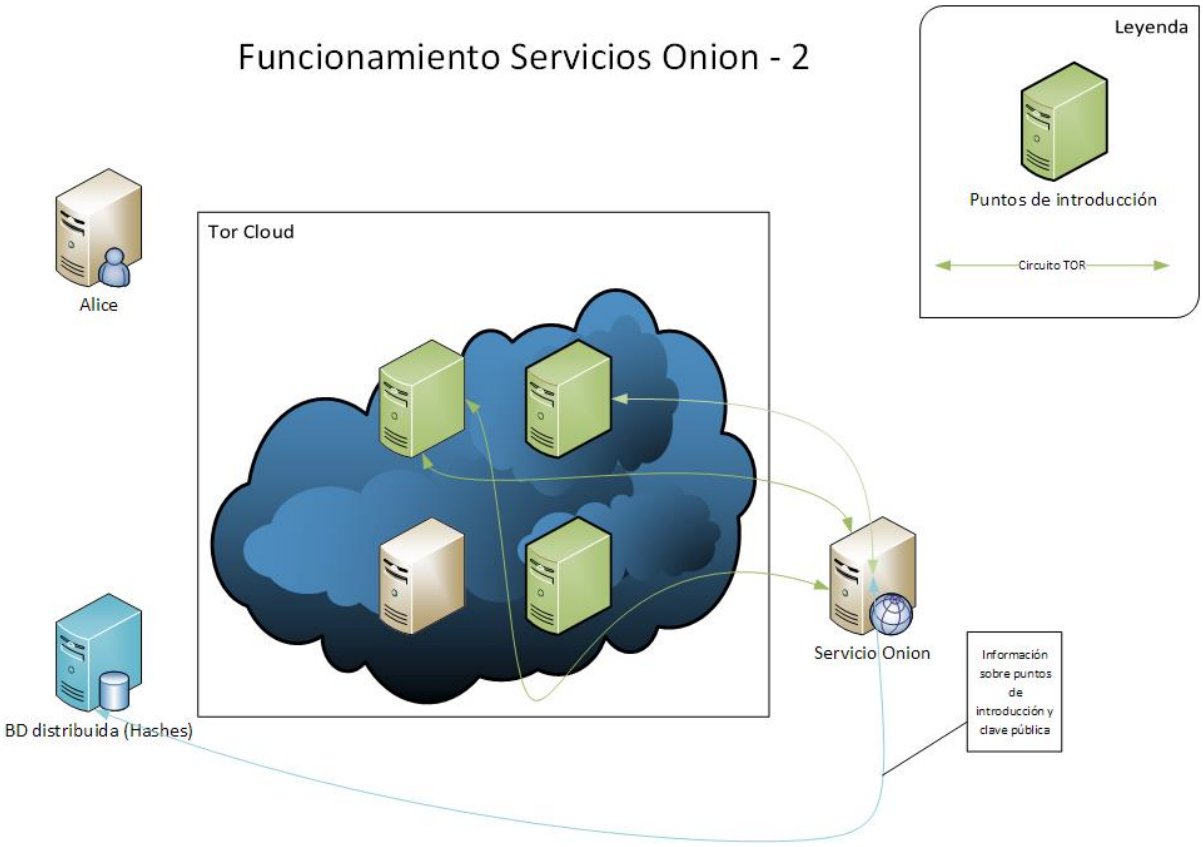

Ilustración 13 - Comunicación con servicios onion - 2

Cuando un cliente, como por ejemplo Tor Browser, quiere contactar con un servicio onion, primero debe de saber su dirección ".onion". Después el cliente puede iniciar una conexión, descargando el descriptor del servicio, de la base de datos de hashes distribuida. Si esa dirección se encuentra, el cliente, mediante el descriptor, conoce los puntos de introducción, así como la clave pública del servicio.

En este momento, el cliente crea un circuito aleatorio hasta un nodo, y le pide que actúe como **punto de encuentro**, indicándole un valor secreto de un solo uso.

Cuando se tiene descriptor y el punto de encuentro está listo, el cliente construye un mensaje de presentación, encriptado con la clave pública del servicio de onion, que incluye la dirección del punto de encuentro y el secreto.

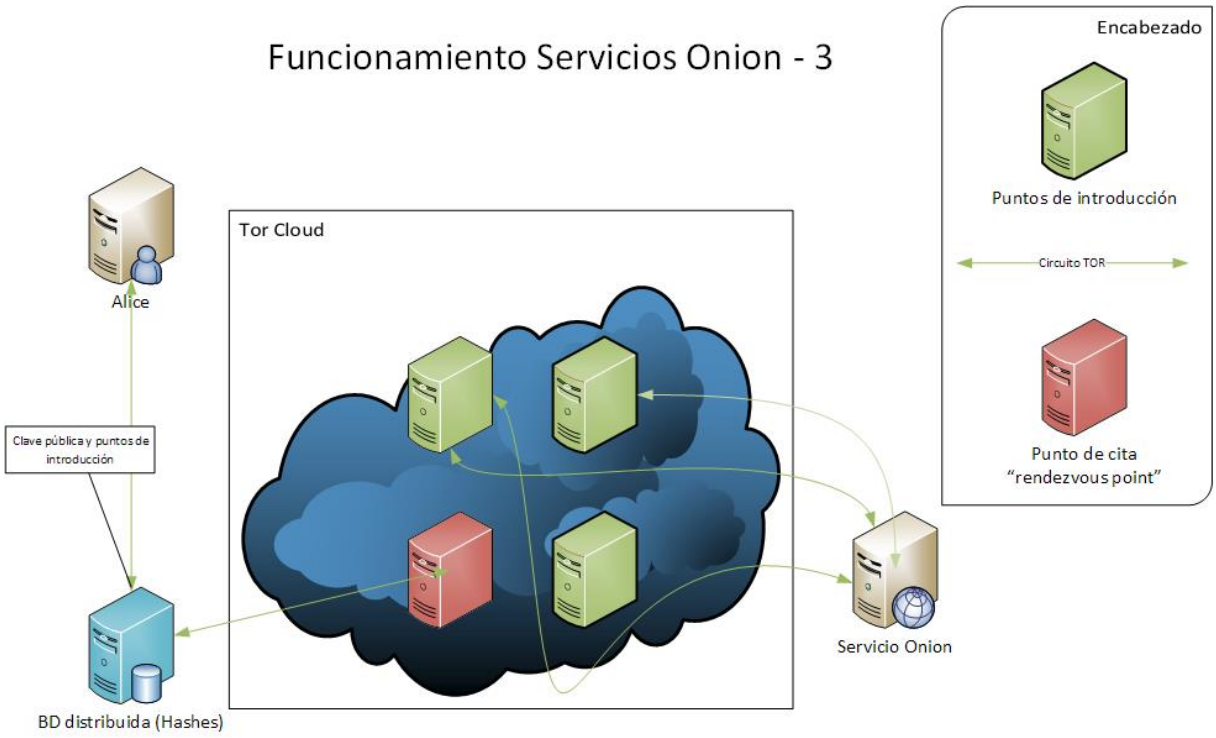

Ilustración 14 - Comunicación con servicios onion - 3

El cliente envía este mensaje a uno de los puntos de introducción, solicitando que se entregue al servicio de cebolla. Una vez más, la comunicación se realiza a través de un circuito Tor, por lo que nadie puede relacionar el envío del mensaje de introducción a la dirección IP del cliente, garantizando su anonimato.

En este punto, el servicio onion descifra el mensaje de presentación del cliente, encuentra la dirección del "punto de encuentro" y el valor secreto enviado. Ahora es el servicio el que crea un circuito hasta el punto de encuentro elegido, y envía de vuelta el secreto de un solo uso, en un mensaje de encuentro.

Este punto es crítico para la seguridad, aquí el servicio debería mantener los nodos guarda o entrada usados, cuando cree nuevos circuitos. De lo contrario, un atacante podría lanzar su propio nodo y forzar a un servicio onion a crear un número arbitrario de circuitos con la esperanza de que el relé "controlado" sea elegido como nodo de entrada o guarda y descubra la dirección IP del servidor de onion a través del análisis de tiempos (Lasse & Syverson, 2006).

En el último paso, el punto de encuentro notificará al cliente que la conexión se ha establecido correctamente. A partir de este momento, el cliente y el servicio onion, se comunicarán entre

sí, usando sus circuitos hasta el punto de encuentro. El punto de encuentro simplemente retrasmitirá los mensajes en ambas direcciones, encriptados extremo-a-extremo.

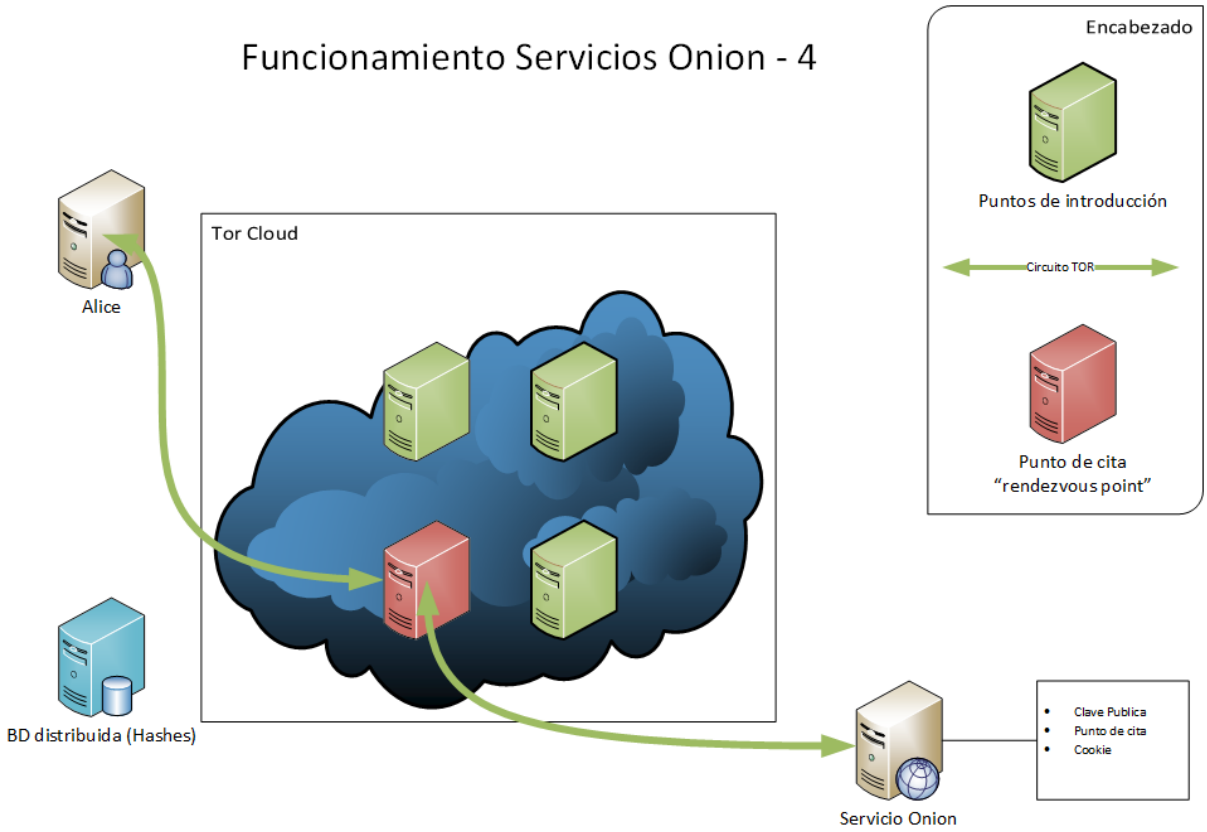

Ilustración 15 - Comunicación con servicios onion - 4

En general, la conexión completa entre el cliente y el servicio onion consiste en 6 nodos o relays, por un lado, 3 de ellos fueron seleccionados por el cliente con el tercero como punto de encuentro y los otros 3 fueron seleccionados por el servicio onion.

2.5 Mercados en la red TOR.

2.5.1 ¿Qué es un darknet market?

Un mercado de darknet o cryptomarket, como se suelen llamar, debido a que los pagos se realizan con cripto-moneda, es un sitio web comercial que opera a través de darknets como Tor o I2P. Por lo tanto, para acceder a ellos, necesitan herramientas como el navegador Tor.

Su funcionamiento es como el de los mercados negros, vendiendo o negociando transacciones que involucran drogas, armas cibernéticas, armas, moneda falsificada, datos de tarjetas de crédito robadas, documentos falsificados, productos farmacéuticos sin licencia, esteroides, servicios de hacking y otros productos ilícitos, aunque también se venden productos legales. En diciembre de 2014, un estudio de Gareth Owen de la Universidad de Portsmouth sugirió que los segundos sitios más populares en Tor eran los mercados de darknet.

Este tipo de mercados se caracterizan fundamentalmente por el su acceso anónimo, típicamente usando la red Tor, realizando pagos Bitcoin, Monero, etc. y uso de servicios de scrow o custodia.

Uno de los mercados pioneros fue ‘The Silk Road’, fundada por Ross Ulbricht, bajo el seudónimo “Dread pirate Roberts” en el año 2011. Este mercado, que vendía drogas fundamentalmente, ha sido uno de los referentes de los darkmarkets y fue cerrado por el FBI en octubre de 2013 después de una larga investigación (Reuters, 2013). Después de esto han aparecido multitud de nuevos mercados, algunos de los cuales ya han sido cerrados en operaciones policiales, en cuanto se cierra uno automáticamente aparecen nuevos en la Darkweb.

2.5.2 Características de los “dark markets”

La mayoría de estos mercados usan el inglés, pero algunos han empezado a operar en chino, ruso o ucraniano.

Los usuarios, una vez que descubren una dirección de uno de estos mercados, se deben de registrar, a veces esto solo puede hacerse mediante invitación. A partir de ese momento, ya puede ver que productos se venden.

No todos los mercados comparten el mismo nivel de seguridad, disponibilidad de productos, facilidad de uso, credibilidad, transparencia u otras características. De hecho, existen muchos mercados que realmente son fraudes que solo buscan hacerse con el dinero (cripto-monedas) de los usuarios.

Las transacciones en estos mercados se suelen hacer mediante Bitcoins, o monedas con mayor capacidad de anonimato como Monero (Monero, 2019).

Algunos mercados utilizan sistemas de scrow o intermediarios de confianza para hacer más seguras las transacciones. Otros deciden realizar las transacciones a través de fondos en el propio sitio, al realizar una compra, el comprador debe transferir la criptomoneda a un depósito en garantía del mercado, después de lo cual el vendedor envía sus productos y luego reclama el pago al mercado.

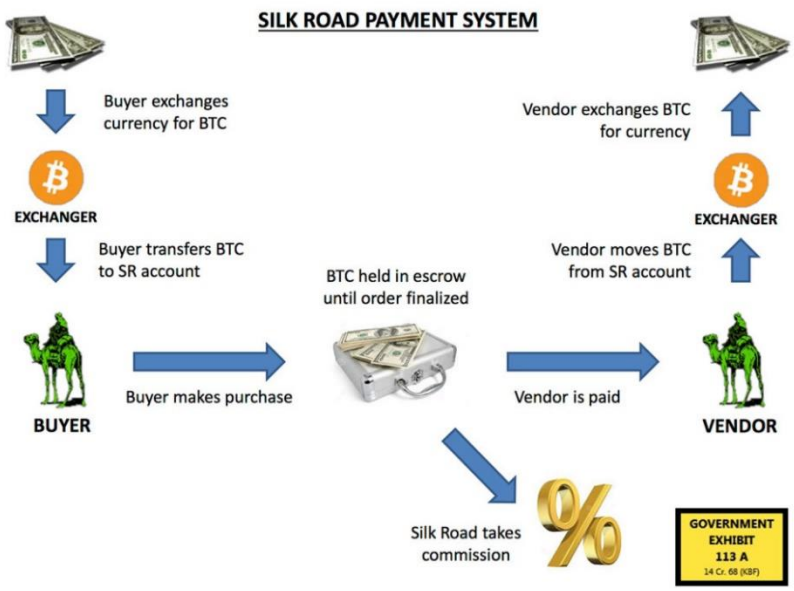

Ilustración 16 - Modelo de pago usando en Silk Road

Estos mercados, suelen permitir que los usuarios pueden dejar comentarios que valoran la transacción y dan o no confianza sobre ese vendedor concreto, generando un sistema de reputación que puede ayudar a decidir o no la compra según de que vendedor se trate.

Otra característica importante, es se utiliza PGP para garantizar el anonimato de las comunicaciones (Pontiroli, 2013) entre los compradores y los vendedores.

Muchos mercados, prohíben expresamente, la venta de ciertos productos o servicios que consideran inadecuados, como venenos, pornografía infantil, armas de destrucción masiva, terrorismo, contratos de asesinato, y en general productos que se puedan utilizar para “dañar” a personas.

3.Objetivos y Metodología

3.1 Objetivo General

El objetivo principal de este trabajo es crear y desplegar en la red Tor un servicio onion, que será un sitio web de tipo mercado, donde compradores y vendedores pueden interactuar con garantías de anonimato. Las funcionalidades con las que debe contar este servicio deben ser realistas con respecto lo que se espera en la Deep Web de estos servicios. También se analizará y capturará el tráfico de red. Esto ayudara al conocimiento del funcionamiento de la Deep Web y estos tipos de servicio.

3.2 Objetivos Específicos

Crear un mercado en la red Tor con las funcionalidades reales que se esperan de estos servicios en la red Tor.

Implementar ese mercado como un servicio onion v3, será una aplicación web y se desplegará en una infraestructura basada en máquinas virtuales, con intención de garantizar y proteger el anonimato de los operadores del sitio, y proteger el servicio.

Implementar un sistema de comunicación entre los usuarios de la aplicación web que garantice el anonimato de sus comunicaciones. Para ello se usará PGP con criptografía asimétrica, para cifrar sus correos electrónicos. Estos correos solo podrán enviarse y recibirse dentro de la red Tor y descifrados por sus receptores.

Permitir el pago de los bienes adquiridos mediante Bitcoins, pero de forma que se pueda mantener el anonimato de estas transacciones.

Ofrecer a los usuarios del sitio, la posibilidad de seguir y comprobar sus transacciones mediante bitcoins.

Conocer en profundidad el funcionamiento de la red Tor, y sus componentes.

Capturar el tráfico del nodo, donde reside el servicio onion, una vez desplegado en la red y analizar el tráfico capturado comprobando que es lo que podemos obtener del mismo. El tráfico de la red Tor usa varias capas de cifrado, se pretender ver cómo protegen el anonimato en el uso de estos servicios.

3.3 Metodología

El pilar principal de este trabajo corresponde al diseño, desarrollo y despliegue de un sitio web en la red Tor dedicado a la venta de mercancías ofertadas por vendedores y compradas por otros usuarios de la aplicación.

Otro pilar importante, es que debe de implementarse de forma que garantice el anonimato de los operadores del sitio, así como de los usuarios de este, tanto compradores como vendedores, protegiendo tanto en sus comunicaciones como sus intercambios económicos.
Esta fase implica dos vertientes. Por un lado, la arquitectura del sistema, haciendo énfasis en la estructura de seguridad adecuada y de despliegue, y por otro lado, los propios requisitos funcionales y de seguridad necesarios durante el desarrollo de la solución.

Por último, es importante también, observar si se puede obtener información de la captura del tráfico de red, de cara a comprobar si se puede obtener alguna información sobre las transacciones o accesos de los usuarios del sitio.

Se propone una metodología de trabajo, que permita encaminarnos a obtener todos los objetivos. Podemos descomponer esta metodología en una serie de pasos. Estos pasos serán interactivos, de forma que ante objetivos específicos se pueden usar de la misma forma.
Los pasos de la metodología son:

1. Definición y análisis de los requisitos.
2. Investigación y estudio de las posibles soluciones para resolver el problema.
3. Diseño, implementación, prueba y despliegue de la solución.
4. Análisis de los resultados

Estos pasos son interactivos, que se aplican a los objetivos globales del proyecto a alto nivel, es decir al desarrollo completo del mercado, su despliegue y análisis de los datos obtenidos de las capturas de red pudiéndose volver a iterar de nuevo las veces necesarias. Pero, además, se usa la misma metodología para los objetivos específicos. Es decir, pongamos los

casos de la comunicación privada a los vendedores y compradores, o el caso de la gestión de pagos mediante bitcoins. Ambos casos son subproblemas a los que se aplicarán los mismos 4 pasos indicados con la posibilidad igualmente de interacción.

Ilustración 17 - Metodología

4. Construcción de MarkeTOR

Cuando escuchamos el término Deep Web, Dark Web, cripto-mercados y términos similares, nos viene a la mente algo que relacionamos por ejemplo con drogas, armas, terrorismo, contratación de asesinos, pornografía, y todo tipo de monstruosidades. Aunque esto es así en muchos casos, estas redes y los servicios que residen en ellos no tiene por qué ser así.

En Internet, actualmente, no existe prácticamente ningún tipo de privacidad, cuando navegamos por ella, múltiples datos sobre las personas y el uso que hacen de la misma se almacenan y analiza por terceras partes que ni siquiera nos imaginamos. Aunque esto último no tiene por qué ser intrínsecamente malo, pueden existir personas que no se encuentren cómodas con un sistema así. En países donde la libertad de expresión está controlada y censurada o el conocimiento no es de libre acceso, por restricciones religiosas o políticas, es donde más claro se ve la necesidad de ciertos niveles de privacidad. Es aquí donde se puede apreciar la utilidad de la red TOR y de los servicios alojados en esta red, los servicios onion. Por otro lado, es cierto que estas garantías de anonimato pueden ser aprovechadas para realizar actividades ilegales o crear servicios ilícitos.

Este proyecto se aborda desde el lado “rojo” de la línea, nos ponemos en la piel de alguien que desea desarrollar un servicio en la red TOR con la intención de ofrecerlo con unas altas garantías de anonimato. Posteriormente se analizará el tráfico obtenido del uso de este sitio, para comprobar si se consiguen o no las medidas de anonimato requeridas.

MarkeTOR es el nombre que se le ha dado a este proyecto, que consiste en el diseño, implementación y despliegue un mercado como servicio onion en la red TOR.

Este sitio web, permite a vendedores ofrecer productos, que pueden ser adquiridos por compradores con garantías de anonimato a ambas partes, así como a los operadores de este. Las transacciones económicas se realizarán mediante criptomoneda, en concreto en este proyecto se usará Bitcoin.

También se analizará el tráfico que se capture durante el uso de la aplicación, en distintas situaciones, con el fin de comprobar si es posible obtener alguna información sobre las transacciones que realizan los usuarios. Se espera que esto sea realmente complicado, debido a las características de anonimato, mediante cifrado que ofrece la red TOR.

Los requisitos de funcionamiento que debe tener un mercado de este tipo se analizarán desde una perspectiva real y viable. Identificando los retos de seguridad, confidencialidad y anonimato requeridos en el mundo real para una aplicación así. Detectado estos requisitos y ofreciendo una solución viable al mismo.

4.1 Requisitos de MarkeTOR

De los objetivos principales y específicos podemos obtener los requisitos funcionales de la aplicación web. Enumeraremos y detallaremos estos requisitos.

- La aplicación será una aplicación web desplegada en la red TOR como un servicio onion versión 3.

- La arquitectura de despliegue debería proteger la localización y acceso al servidor donde se ejecuta el servicio onion.

- El desarrollo y despliegue del sitio debe de garantizar en lo posible el anonimato de los operadores, compradores y vendedores.

- Las transacciones económicas de los productos se realizarán mediante la criptomoneda Bitcoin y estas transacciones podrán seguirse para comprobar si se han completado o no.

- Existirán los roles de comprador, vendedor y administrador o operador del sitio.

- Las comunicaciones entre los compradores, vendedores y operadores del sitio serán completamente privadas y anónimas, usando cifrado basado en infraestructura de clave pública.

- Los usuarios no podrán ver los productos hasta que se hayan dado de alta en el portal.

- El sitio web no debe exponer los inicios de sesión de usuario en ningún momento a ninguno de los otros usuarios.

- Cuando un usuario compra productos de varios compradores, los compradores no deben de saber que otros productos se han comprado a los restantes compradores.

- Tanto compradores como vendedores deberán tener un medio para comprobar el estado que las transacciones económicas.

4.2 MarkeTOR: Diseño, implementación y despliegue

Se expondrán a continuación las distintas partes del desarrollo de forma en las que un usuario las fuera encontrando al usar la aplicación. En cada parte o funcionalidad se explicará cómo se ha diseñado y construido esa parte, indicando qué problema resuelve y como se ha implementado.

4.2.1 Tecnologías utilizadas en MarkeTOR

Lenguaje de programación usado: Python.

En el desarrollo de la aplicación MarkeTOR, se ha optado por la utilización de Python[17] en su versión 3.7, como lenguaje de implementación. La elección de este lenguaje de programación para la implementación se ha basado en los siguientes criterios:

- **Conocimiento del autor**. El autor del proyecto tiene conocimientos de programación usando este lenguaje, lo que facilita si construcción.

- **Frameworks web disponibles**: Python permite elegir entre una amplia variedad de ellos, con características importantes de cara a abordar este proyecto, como son seguridad y robustez, por ejemplo. Se ha elegido Django en este proyecto.

- **Seguridad**. Python es considerado un lenguaje con garantías de seguridad en cuanto a tipos y gestión de memoria.

[17] https://www.python.org/

- **Existencia de librerías**. La existencia de librerías para casi cualquier cosa en Python, garantiza el progreso rápido en el desarrollo de la aplicación.

Django: Framework Web

Se ha elegido Django como framework de desarrollo web. Django[18] está basado en Python, y fomenta el desarrollo rápido y un diseño limpio y pragmático. Es una herramienta que se toma en serio la seguridad y ayuda a evitar muchos errores de seguridad comunes. La versión de Django utilizada en este desarrollo es la versión 2.2 y se ha elegido esta versión ya que es una versión con soporte de larga duración, lo que garantiza la solución de bugs y fallos de seguridad durante un periodo de tiempo mayor. Esto permitiría evitar las actualizaciones de versión durante más tiempo.

Vagrant: **Gestión de máquinas virtuales en desarrollo.**

Vagrant[19] es una herramienta que ayuda a crear y manejar máquinas virtuales definidas mediante un archivo de configuración. Permite definir los servicios a instalar, así como también sus configuraciones, lo que permite tener un entorno de desarrollo lo más parecido al de producción. En este caso se ha usado una máquina basada en Debian 10, ya que el despliegue se realiza sobre Whonix versión 15, que es una distribución basada en Debian 10. A la hora de desplegar en producción todas las dependencias se pueden obtener ya instaladas o instalarlas de forma sencilla. Permite desarrollar en local como si estuviésemos trabajando en el entorno final de producción prácticamente.

Whonix 15: Sistema Operativo en producción.

Whonix[20] es una distribución Linux, que se centra en la seguridad, la privacidad y el anonimato. Está basado en Debian 10 y en la red TOR. Whonix consta de dos partes Whonix-Gateway, máquina virtual que ejecuta a TOR y redirige todo el tráfico a través de TOR y Whonix-Workstation, máquina virtual completamente aislada, que solo puede enviar y recibir

[18] https://www.djangoproject.com/
[19] https://www.vagrantup.com/
[20] https://www.whonix.org/

tráfico a través Whonix-Gateway. El lugar ideal para ejecutar MarkeTOR como servicio onion, es en Whonix-Workstation.

4.2.2 Arquitectura del sistema

Uno de los requisitos más importantes, como se ha indicado, es la seguridad, la privacidad y anonimato de la aplicación y de los usuarios. Un punto muy importante para garantizar estas metas es la arquitectura sobre la que se despliegue la solución.

Como ya se ha indicado previamente, la arquitecta estará basada en máquinas virtuales ejecutando Whonix como sistema operativo.

Se ha elegido Oracle VM VirtualBox[21] como hipervisor, ya que es un producto Open Source, que se puede ejecutar bajo cualquier sistema operativo y que nos permite mucha flexibilidad. Permite además aplicar cifrado a las máquinas virtuales con AES-256.

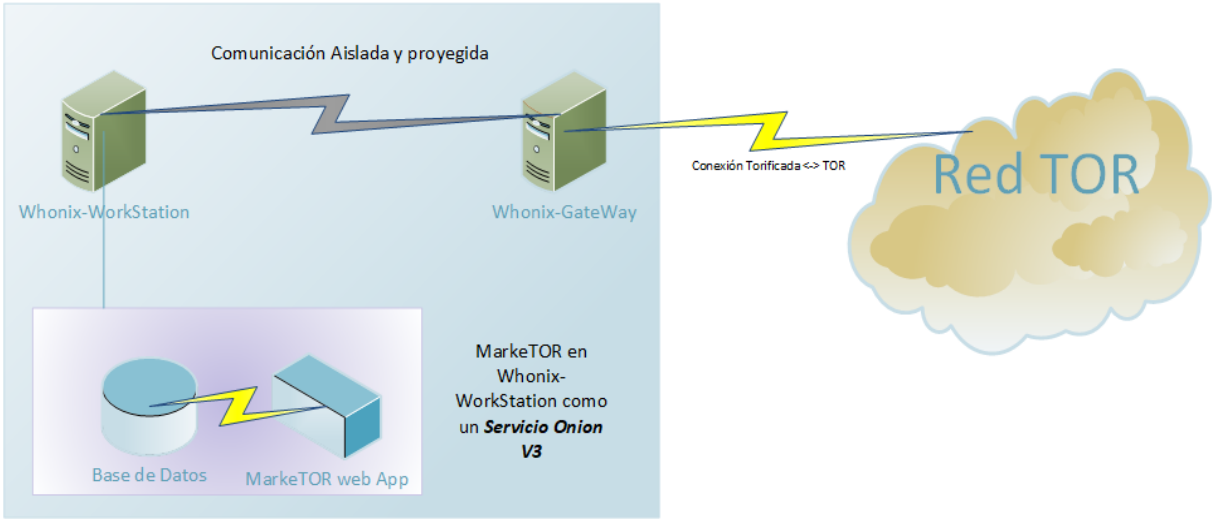

Ilustración 18 - Arquitectura básica de despliegue

En el diagrama se pueden apreciar los distintos componentes de la arquitectura usada para aislar la aplicación y garantizar la seguridad y anonimato.

[21] https://www.virtualbox.org/

La aplicación Django, junto con la base de datos, residen en la máquina virtual **Whonix-WorkStation**, esta máquina virtual ejecuta la aplicación como un **servicio onion v3** y está configurada como tal.

Esta máquina está completamente aislada y protegida mediante firewall, no puede establecer comunicaciones de ninguna forma si no es pasando por la máquina **Whonix-Gateway**, ya que tanto la configuración de red como de firewall solamente permiten ese tipo de comunicación.

La máquina **Whonix-Gateway** se comunica solamente con TOR, actúa como un **proxy hacia la red TOR**.

La implicación de esta arquitectura es que solamente está expuesta en la red TOR la máquina Whonix-Gateway, el servicio onion que corre en Whonix-Workstation está protegido, detrás de este proxy y completamente aislado.

Un adversario que quisiera tomar el control del servicio toparía con la pasarela, añadiendo, si no pasa inadvertido una capa más de seguridad y complejidad. Otra ventaja de esta arquitectura es que un software dañino que hipotéticamente llegase a afectar al servicio o a la máquina donde reside, estaría aislado y tendría complicado la posibilidad de comunicación con el exterior.

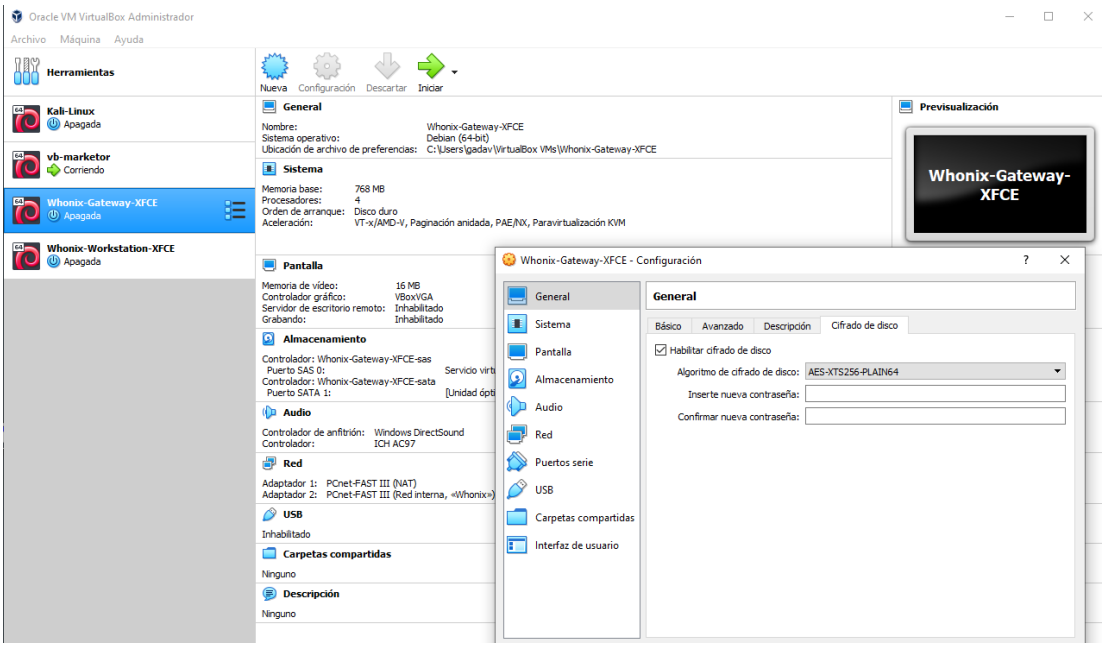

Ilustración 19 - Máquinas virtuales sobre Oracle VM VirtualBox

4.2.3 Arquitectura de la aplicación MarkeTOR

Ya se ha presentado la arquitectura sobre la que se desplegará la aplicación, pero igual de importante es la arquitectura propia de la aplicación en sí, es decir los componentes que la forman.

Django tiene una forma de trabajo y estructura propia ya definida que se debe seguir en la construcción de aplicaciones. MarkeTOR sigue fielmente esa estructura. La siguiente imagen muestra los componentes de la aplicación que describiremos a continuación.

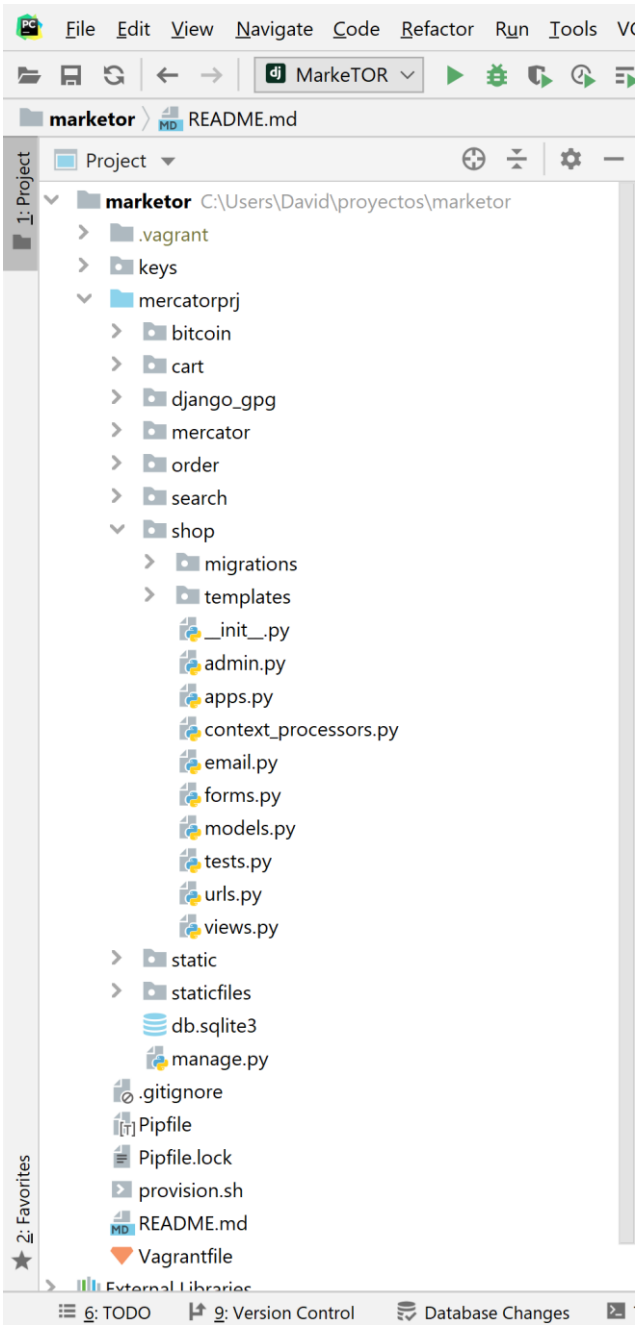

Ilustración 20 - Vista de la arquitectura de la aplicación

El proyecto Django de la aplicación de llama como se aprecia en la imagen '**mercatorprj'**, este proyecto contiene los distintos componentes de los que está formado, en Django estos componentes de primer nivel se llaman aplicaciones y residen en carpetas que cuelgan directamente del proyecto.

Estas aplicaciones son contenedores de funcionalidad relacionada, por ejemplo, se puede ver en la imagen, aplicaciones como son **bitcoin**, **cart**, **order** etc. La aplicación 'bitcoin', por ejemplo, contiene toda la funcionalidad relacionada con el pago mediante bitcoin, la gestión de transacciones, direcciones bitcoin etc. Por otro lado, tendríamos '**order'** donde toda la funcionalidad sobre las órdenes de compra se aglutina. Así con el resto de las funcionalidades que explicaremos en detalle.

Django también tiene una forma muy concreta de organizar el código dentro de estas aplicaciones. Django fue diseñado con la intención de ofrecer el acoplamiento débil y una separación estricta entre las partes de una aplicación. Si se sigue esa filosofía, es fácil hacer cambios en un lugar particular de la aplicación sin afectar otras partes.

El patrón de diseño que utiliza Django es una variación de MVC[22] (Model-View-Controller) se conoce como MVT (Model-View-Template), donde:

- **M -"Model" (Modelo).** La capa de acceso a la base de datos. Esta capa contiene toda la información sobre los datos de la aplicación, cómo acceder a estos, validación, su comportamiento y las relaciones entre los datos.

- **V - "View" (Vista)**, La capa de la lógica. Esta capa contiene la lógica que accede al modelo y la delega en la plantilla indicada.

- **T - "Template" (Plantilla),** la capa de presentación. Suelen ser las páginas HTML, pero no necesariamente, pueden ser cualquier tipo de presentación, XML, JSON, TXT, PDF, etc.

La siguiente representación gráfica, puede ayudar a entender esta arquitectura, y a comprender las partes que lo componen y como encajan las piezas juntas.

[22] https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html

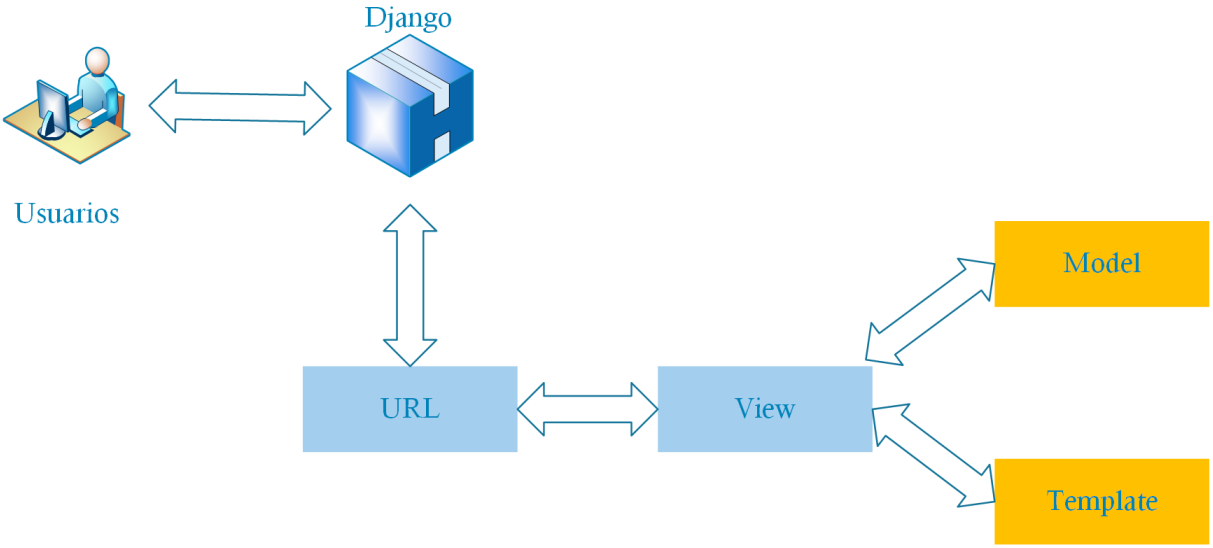

Ilustración 21 - Arquitectura MVT en Django

Si tomamos como ejemplo cualquiera de las aplicaciones que forman parte del proyecto, se puede ver como los módulos python reflejan dicha arquitectura.

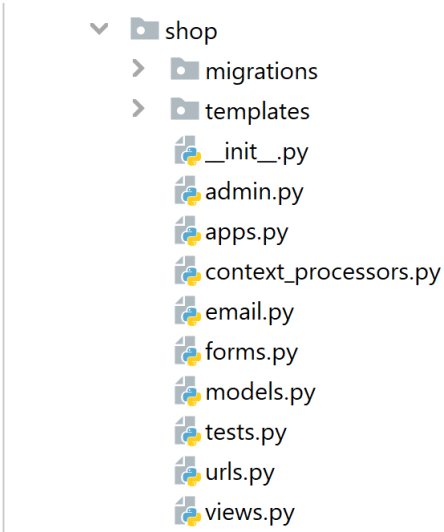

Ilustración 22 - Módulos python dentro de una aplicación Django

Se puede apreciar que la aplicación 'shop', que es la que representa la tienda a nivel general, está compuesto por módulos Python, que siguen la arquitectura, tenemos los módulos:

- **models.py**: Donde se definen los modelos u objetos de la base de datos específicos de esta aplicación.

- **urls.py**: Se definen las URLs a las que responde esta aplicación y donde se especifica que vistas serán llamadas en casa caso. En definitiva, es un mecanismo de rutas en el que se relacionan URLS con funciones Python.

- **Views.py**: Este módulo contiene las vistas del patrón MVT, son funciones Python que usan o acceden al modelo y seleccionan la plantilla adecuada en cada caso.

- **Otros módulos**: Pueden existir múltiples módulos Python, que se usan por Django para distintas funcionalidades, como los formularios en **forms.py**, o simplemente capas de servicio que añadimos como desarrolladores para separar la funcionalidad en la aplicación, como **email.py**, que implementa funcionalidad de envío de correo electrónico.

Se describirá brevemente las aplicaciones Django de las que consta el proyecto MarkeTOR, su funcionalidad se irá estudiando en detalle en puntos posteriores de este trabajo.

- **bitcoin**: Esta aplicación Django aglutina toda la funcionalidad del sitio web relacionada con el pago mediante bitcoins y la gestión de transacciones. En concreto permite:

    - La creación y gestión de monederos bitcoin y sus direcciones.
    - Realizar transacciones y subirlas a la blockchain bitcoin.
    - Comprobar los fondos que contiene una cartera.
    - Comprobar el estado de una transición en la red bitcoin.
    - Comprobar la validez de una dirección bitcoin.
    - Comprobar los fondos procedentes de una dirección bitcoin hacia un monedero.

- **cart**: La aplicación cart, es el carrito de la compra de la aplicación MarkeTOR, asocia los productos que se desean comprar por un usuario y los agrupa por vendedor

concreto y los prepara para el pago. Esta aplicación contiene los modelos que se almacena en la base de datos necesarios para el correcto funcionamiento del carrito y su persistencia entre sesiones. Las vistas necesarias para añadir, eliminar y modificar los productos añadidos al carrito. Las plantillas para mostrar la interfaz de usuario del carrito. Cualquier otra funcionalidad necesaria para el funcionamiento del carrito.

- **django_gpg[23]:** Aplicación para la gestión de claves PGP/GnuPG[24] públicas asociadas a los perfiles de usuario, y en cifrado de mensajes mediante clave infraestructura de clave pública. Este módulo es de terceros, obtenido desde GitHub, escrito inicialmente por Alex Reckter (Recker, n.d.). este módulo se ha modificado, ya que la aplicación estaba desarrollada para Django 1.x y Python 2.x. MarkeTOR es una aplicación Django 2.2.x que usa Python 3.7, **se ha modificado el código original, para hacerlo compatible con Python 3 y Django 2**.

- **mercator**: Aplicación principal donde residen los archivos de configuración de la aplicación y el módulo WSGI de arranque, también es donde se definen las rutas de alto nivel y donde se delegan a las demás aplicaciones.

- **order**: Gestión de órdenes de compra. Esta aplicación aglutina todo lo relacionado con las órdenes de compra y la preparación para el pago. Sus modelos para almacenar en la base de datos, las vistas para ver los detalles de una orden o el listado de órdenes de compra de un usuario, así como sus plantillas HTML se encuentran en esta aplicación. Trabaja estrechamente con las aplicaciones bitcoin, cart y shop.

- **search**: Aplicación de búsqueda. Permite buscar en el mercado posibles productos en MarkeTOR, en base a su descripción y su nombre. Esta aplicación solamente implementa vistas.

- **shop**: Aplicación central del mercado. Sus responsabilidades van desde la gestión de usuarios y sus perfiles, la gestión de categorías y de productos. Las vistas de esta aplicación son las que conforman el mercado en sí, permite navegar por los productos y categorías. Sus modelos almacenar en la base de datos los datos referidos a productos, categorías y perfiles de usuario.

[23] https://github.com/arecker/django-gpg
[24] https://gnupg.org/

4.2.4 Artefactos de desarrollo: Vagrant y Pipenv

Vagrant. Gestión de máquinas virtuales.

Para poder desarrollar la aplicación se ha utilizado, Vagrant como sistema de gestión de máquinas virtuales en desarrollo. El sistema operativo de despliegue elegido ha sido **Whonix 15**, el cual está basado en **Debian 10**, pero las maquinas que se han utilizado para el desarrollo de la aplicación han sido varias y con sistemas operativos variados. Se ha utilizado como sistemas operativos en desarrollo Windows 10, OpenSUSE Tumbleweed y Fedora Workstation 30. Como se puede observar son sistemas muy variados y que tienen que ver poco con el sistema final usado en producción.

El uso de Vagrant permite desarrollar, ejecutar, depurar y probar la aplicación en la máquina virtual deseada, en este cado Debian 10 (Buster), desde cualquier otro sistema operativo, lo que garantiza el correcto funcionamiento en producción, ya que el entorno de desarrollo y producción se asemejan completamente de esta forma. También permite la destrucción y regeneración del entorno de trabajo de forma muy rápida, con lo que pruebas y evoluciones en el desarrollo que se han desechado, no dejan ningún rastro y el estado de la máquina se controla completamente desde el archivo de configuración **Vagrantfile**.

```
Vagrant.configure("2") do |config|

  config.vm.box = "debian/contrib-buster64"
  config.vm.network :forwarded_port, guest: 8000, host: 8000
  config.vm.network :forwarded_port, guest: 80, host: 80

  config.vm.provider "virtualbox" do |vb|
   vb.gui = false

   vb.name = "vb-marketor"
   vb.memory = "2024"
   vb.cpus = 2
  end

  config.ssh.shell = "bash -c 'BASH_ENV=/etc/profile exec bash'"

  config.vm.provision :shell, :path => "provision.sh"
end
```

Ilustración 23 - Archivo Vagrantfile usado en el desarrollo de MarkeTOR

En el archivo *Vagrantfile* se puede observar el tipo de imagen usado "debian/contrib-buster64", los puertos expuestos por esa máquina virtual, en este caso el 80 y el 8000 que es el que usa Django en su servidor de desarrollo. En el archivo también se define las características de la máquina, en este caso definimos que se usarán 2GB de RAM, 2 CPUs virtuales y que no se usará entorno visual en la máquina.

Se aprecia también la llamada al script de aprovisionamiento, que se ejecutará cada vez que se crea o aprovisiona la máquina de nuevo. Permite llevar la máquina a un estado concreto cuando se construye. El script se llama '*provision.sh'* y se expone a continuación sus líneas principales.

Fundamentalmente preparan la máquina usar Python mediante entorno virtual instalando las dependencias necesarias.

```
#!/usr/bin/env bash

export DEBIAN_FRONTEND=noninteractive

# update del sistema
sudo apt-get update -q
sudo apt-get upgrade -y
sudo apt-get autoremove -y

# Utilidades de configuración y dependencias de desarrollo
sudo apt-get install debconf-utils -y

sudo apt-get install python3-pip -y
sudo apt-get install python3-dev libmysqlclient-dev libpq-dev -y

# base de datos
sudo apt-get install postgresql -y

sudo apt-get install python3-venv -y

# TOR
sudo apt install tor -y

#Pipenv
sudo -H pip3 install pipenv
```

Ilustración 24 – Shell script de aprovisionamiento provosion.sh para vagrant

Pipenv.

La mayoría de los sistemas GNU/Linux, y Whonix lo es, utilizan intensivamente Python y viene instalado por defecto junto con muchos paquetes concretos. Si realizamos un desarrollo con Python lo ideal es hacerlo en un 'entorno virtual' completamente aislado de la instalación del sistema, eligiendo la versión concreta de Python a utilizar e instalando los paquetes necesarios para el desarrollo sin que interfieran con los instalados por defecto en el sistema GNU/Linux usado. Aquí es donde interviene *pipenv*[25].

Pipenv es una herramienta que permite crear y gestionar entornos virtuales Python y actuar como un gestor de paquetes y dependencias que permite realizar construcciones de proyectos Python deterministas, es decir repetibles.

Una de las funcionalidades más interesantes desde el punto de vista de la **seguridad**, es que permite comprobar si se han detectado vulnerabilidades en las dependencias utilizadas por el proyecto y actualizarlas en caso afirmativo. Esto último ayuda a las prácticas de seguridad sobre la fase de desarrollo y producción del proyecto, permitiendo usar paquetes y dependencias sin vulnerabilidades detectadas de una forma rápida y sencilla.

```
[[source]]
name = "pypi"
url = "https://pypi.org/simple"
verify_ssl = true

[dev-packages]
django-debug-toolbar = "*"

[packages]
django = "~=2.2"
pillow = "*"
django-crispy-forms = "*"
bit = "*"
requests = "*"
python-gnupg = "*"
gunicorn = "*"

[requires]
python_version = "3.7"
```

Ilustración 25 - Pipfile con las dependencias del proyecto MarkeTOR

[25] https://pipenv.readthedocs.io/en/latest/

4.2.5 Acceso al servicio onion: MarkeTOR

Para poder acceder al sitio desplegado en TOR, debemos utilizar TOR Browser[26], si usamos la dirección del sitio en la barra de navegación llegaremos a la página inicial del sitio web. Cualquier otra página visitada sin haber iniciado sesión previamente, redirige a esta misma página.

La dirección del servicio onion es:

```
http://mktor3q5k22c24n6y5izk73iw2y5l5kpp2buejg5wbgupjnw4n6q2tqd.onion
```

Ilustración 26 - Dirección del servicio onion v3

En la dirección de acceso a la dirección, se pueden observar varias cosas:

- Se puede apreciar que es una dirección '.onion', por lo que reside en la red TOR y necesitamos TOR Browser para llegar a él, como ya hemos mencionado.

- La longitud de la dirección es de 56 caracteres, derivados de la clave pública de tipo curva elíptica **ed25519**. Esto indica que es un servicio onion versión 3.

- El comienzo de la dirección se parece al nombre de la aplicación, empieza por *mktor*, indica que es una *dirección personalizada y generada mediante fuerza bruta*. Más adelante en este trabajo si indicará cómo se ha generado esta dirección.

Si describimos la aplicación desde el punto de vista de su uso, es decir, cómo la usan los usuarios, el primer paso sería poder navegar por el mercado y ver sus productos. Esto no es posible en MarkeTOR, debido a que uno de los requisitos primordiales es la privacidad, no se puede acceder a ver los productos, sus categorías, búsquedas y demás funcionalidad hasta que un usuario se da de alta, entra en sesión y puede navegar. Vamos a seguir ese criterio para describir la aplicación y como se ha diseñado.

[26] https://www.torproject.org/download/

4.2.6 Gestión de usuarios

Se describirán a continuación los procesos básicos de la gestión de usuarios en el sitio, fundamentalmente la creación de usuarios, perfiles y la autenticación y control de acceso.

Como ya hemos indicado, solo se puede acceder a la aplicación MarkeTOR, si previamente nos hemos dado de alta e ingresamos con nuestras credenciales en la aplicación. Por lo tanto, cuando accedemos al sitio solo tenemos dos opciones o iniciar sesión o crear un usuario nuevo.

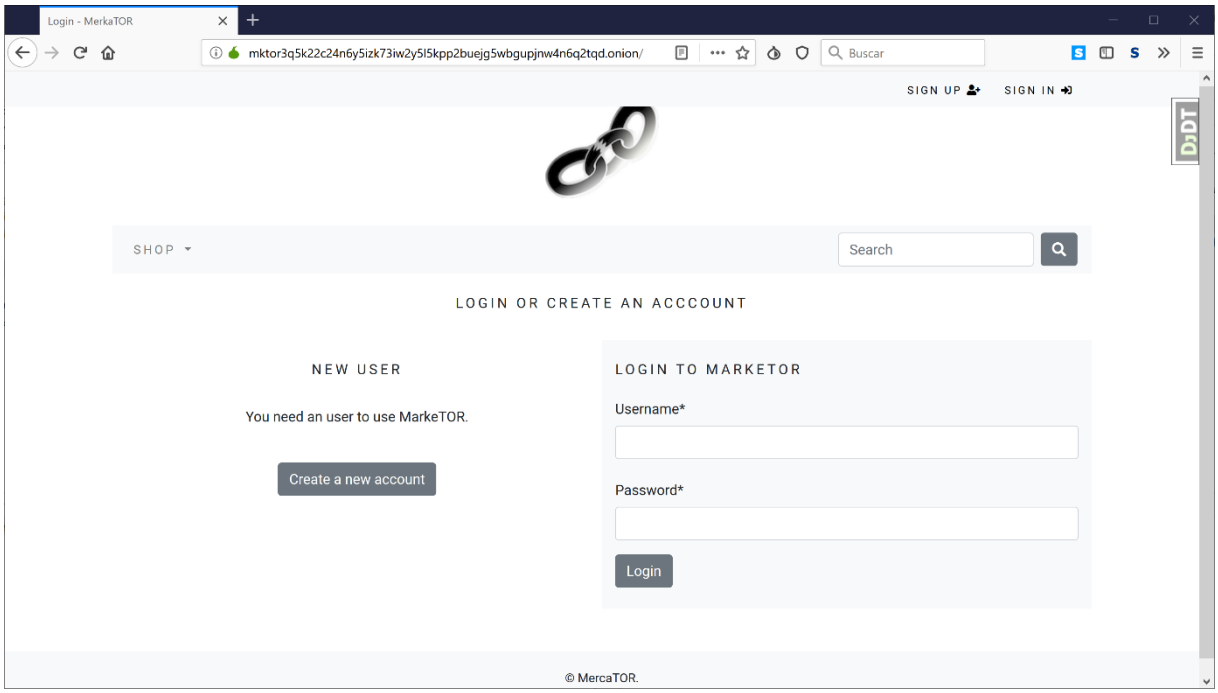

Ilustración 27 - MarkeTOR página de inicio

Se puede observar que la aplicación está enteramente en inglés, esto también es una decisión de diseño, por lado este tipo de sitios podrían atraer a más usuarios que i estuvieran en un idioma menos internacional, y por otro lado desde el punto de vista de los requisitos, donde se pide garantizar la privacidad, usando el inglés, se pretender crear un sitio más aséptico, no desvelando la posible información de origen del sitio.

La creación de un usuario del portal nos pide los datos básicos necesarios para el modelo *User*, que nos permitirá como mínimo acceder al sitio.

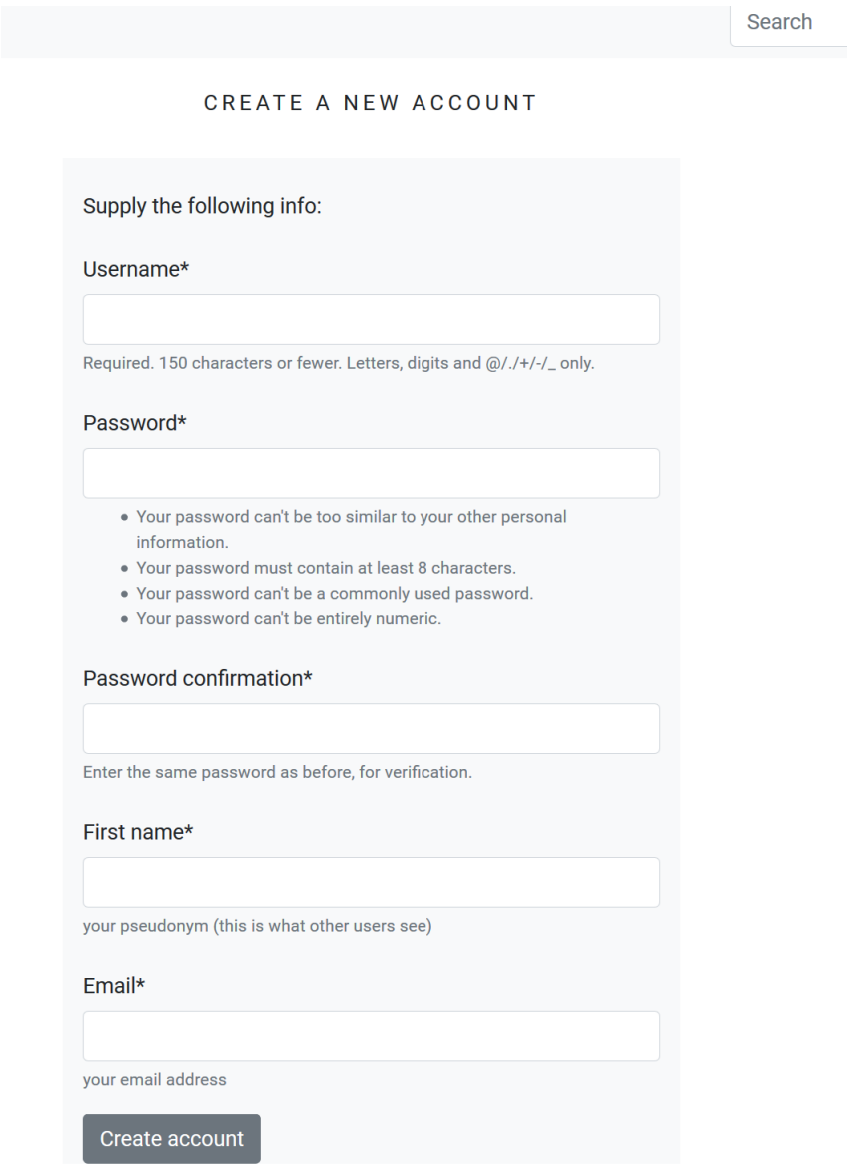

Ilustración 28 - Formulario de creación de usuario nuevo

Los datos básicos del usuario son los que se muestran en el formulario. Es importante mencionar que el campo '*username'* es el que se usará en los inicios de sesión, y no se mostrará nunca en las visitas al sitio por el resto de los usuarios, para eso se usa el campo *'first_name'.*

Se aprecian también las características mínimas que debe de tener una contraseña para ser válida en el sistema. La dirección de correo electrónico también es importante, ya que solo puede ser una dirección tipo '. *onion'*, es decir solamente accesible desde la red TOR.

Una parte fundamental de la gestión de usuarios es el perfil, aquí es donde está la información más interesante, ya que es la que permite realmente interactuar de forma completa con el sitio.

La siguiente imagen muestra un perfil de usuario, de un usuario existente en el sitio, con perfil de vendedor. Se puede apreciar como detalles interesantes, la clave pública GPG, que permite cifrar información de cualquier tipo como por ejemplo los correos que se le envían para maximizar la privacidad. Otro campo interesante es la dirección bitcoin en formato segwit[27] , es a esta dirección del monedero bitcoin a la que se harán los pagos de los productos vendidos por este vendedor. Estos datos se suman a los del alta del usuario, vistos anteriormente.

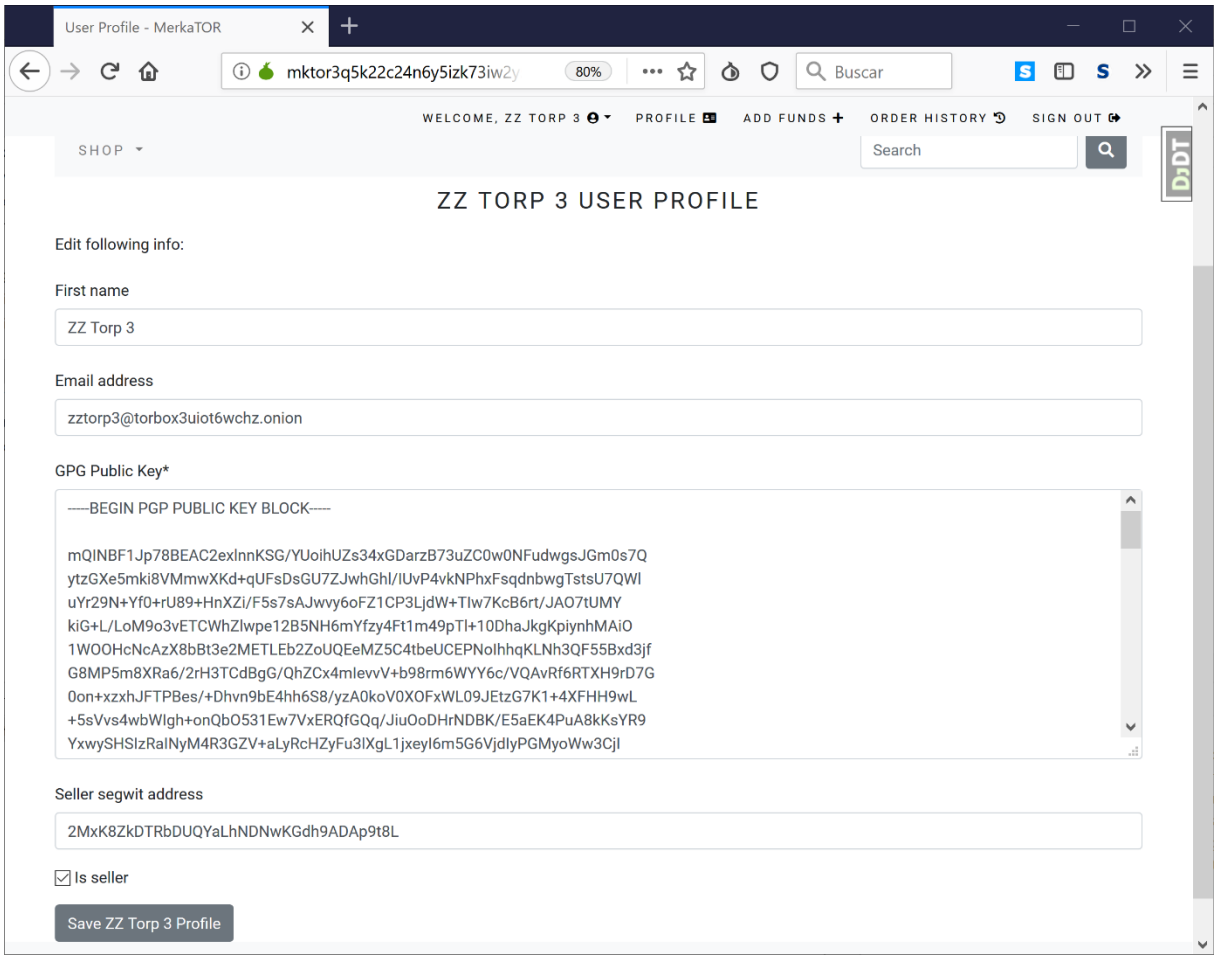

Ilustración 29 - Perfil de usuario en MarkeTOR

[27] https://en.bitcoin.it/wiki/Address

Toda la funcionalidad que implementa la gestión de usuarios, perfiles, autenticación y control de acceso, se implementa en la aplicación **'shop'**, junto con el sistema de autenticación de Django.

Se comentará seguidamente las partes de código más interesantes.

Modelos.

La gestión de usuarios se implementa basándose en el modelo **User** proporcionado por Django, que permite las tareas de autenticación y control de acceso a los recursos, en las vistas y plantillas. A este modelo estándar de Django se añade un perfil de usuario, como una relación 1 a 1 que es necesaria para la gestión del mercado. Funcionalidades como el cifrado mediante infraestructura de clave pública, los monederos para los pagos mediante bitcoin o si se trata de un vendedor, se almacenan en el perfil de usuario, tal como se aprecia el este extracto de los modelos de la aplicación '**shop**'.

```
class UserProfile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
     btc_deposit = models.DecimalField(max_digits=16, decimal_places=8, blank=True,
null=True)
    seller_segwit_address = models.CharField(max_length=100, blank=True, null=True)
    is_seller = models.BooleanField(default=False)

    def __str__(self):
        return '{}'.format(self.user.username)

    def __repr__(self):
        self.__str__()


@receiver(post_save, sender=User)
def create_user_profile(sender, instance, created, **kwargs):
    if created:
        UserProfile.objects.create(user=instance)

@receiver(post_save, sender=User)
def save_user_profile(sender, instance, **kwargs):
    instance.userprofile.save()
```

Ilustración 30 - Modelo UserProfile en shop/models.py

La clave GPG en cambio se integra en otro modelo, también como elación 1 a 1 con '**User**', pero en este caso se implementa en la aplicación '**django_gpg**'.

```
from django.db import models
from django.contrib.auth import get_user_model
from django.dispatch import receiver

from .settings import PUBLIC_KEY_REQUIRED
from .fields import PublicKeyField
from . import gpg


User = get_user_model()

class GpgProfileQueryset(models.QuerySet):
  def encrypt(self, message, ignore_empty=False):
    keys = filter(None, self.values_list('public_key', flat=True))

    if not ignore_empty and self.count() != len(keys):
      raise ValueError('Not every User has a key')

    return gpg.encrypt(message, recipient_keys=keys)


class GpgProfile(models.Model):

  objects = GpgProfileQueryset.as_manager()

  user = models.OneToOneField(
    User,
    primary_key=True,
    on_delete=models.CASCADE
  )

  public_key = PublicKeyField(
    blank=(not PUBLIC_KEY_REQUIRED),
    null=(not PUBLIC_KEY_REQUIRED)
  )


  def encrypt(self, message=''):
    return gpg.encrypt(message, recipient_keys=[self.public_key])

  class Meta:
    verbose_name = 'GPG Profile'


@receiver(models.signals.post_save, sender=User)
def save_gpg_profile(instance=None, created=False, **kwargs):
  if created:
    GpgProfile.objects.create(user=instance)
  else:
    instance.gpgprofile.save()
```

Ilustración 31 - Parte GPG del perfil de usuario

Estos fragmentos de código aglutinan la funcionalidad de los perfiles de usuario de la aplicación.

Vistas.

Las vistas son donde reside la funcionalidad o el control de la autenticación, aquí es donde se controla el *inicio de sesión, cierre de sesión*, alta de usuario etc. Se realiza la acción como puede ser iniciar sesión, usando el modelo adecuado, en este caso, el usuario y sus perfiles, y se redirige la salida a la plantilla adecuada.

Es importante indicar que el control de acceso básico se puede hacer en las vistas (también en las plantillas, aunque es menos recomendable). Por ejemplo, usando el decorador ***@login_required***, en ese caso el código de las vistas no se ejecuta si no se ha iniciado sesión en la aplicación previamente, redirigiendo al usuario a la página de inicio de sesión.

Como ejemplo de vistas en la gestión de usuarios y autenticación el siguiente fragmento de código muestra las vistas de inicio y fin de sesión.

```
def signin_view(request: HttpRequest):
    if request.method == 'POST':
        form = AuthenticationForm(data=request.POST)
        if form.is_valid():
            username = request.POST['username']
            password = request.POST['password']
            user = authenticate(username=username, password=password)
            if user is not None:
                login(request, user)
                return redirect('shop:allProdsCat')
            else:
                return redirect('signup')
    else:
        form = AuthenticationForm()

    return render(request, 'accounts/signin.html', {'form': form})


@login_required
def signout_view(request: HttpRequest):
    logout(request)
    return redirect('signin')
```

Ilustración 32- Vistas de inicio y cierre de sesión

Plantillas.

Las plantillas son el código HTML y CSS de las páginas finales de MarkeTOR, hay poco código relevante en este caso, más allá de HTML y código de plantillas Django, como mucho algún fragmento en el que en función el rol del usuario o si se ha iniciado sesión, se renderiza una parte de la página u otra.

Un ejemplo del uso del control de acceso y la personalización en las plantillas se puede apreciar en la parte superior donde aparecen las opciones destinadas a los usuarios registrados, esto se construye en base a los roles del usuario en el sistema.

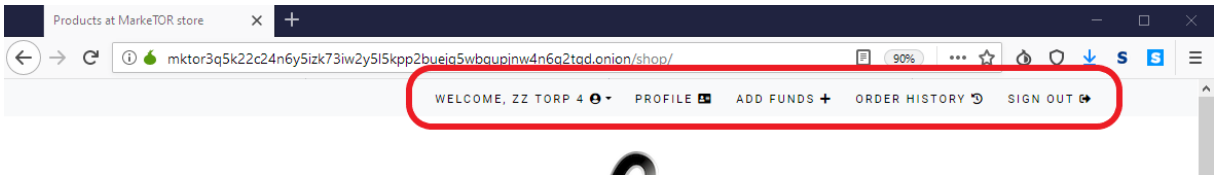

Ilustración 33 - Opciones del perfil de usuario

4.2.7 Comprando en MarkeTOR: Productos, categorías y carrito de la compra.

MarkeTOR es un mercado en la red TOR por lo que la función principal que tiene es la de permitir la compra y venta de productos ofertados por vendedores.

La aplicación permite navegar por las distintas categorías de productos, que se pueden acceder desde el menú de la tienda. Las categorías, aglutinan productos de cierto tipo para simplificar la tarea de buscar lo que necesitan los compradores. Las categorías solo pueden ser creadas por los administradores del sitio, de forma que sólo se admiten los productos que pertenezcan a esas categorías permitidas. Muchos mercados son muy estrictos con el tipo de mercancías que se pueden o no vender.

Inicialmente la página muestra paginados todos los productos de todas las categorías de forma paginada, pero se puede refinar las búsquedas filtrando por *categorías,* o mediante el buscador, situado en la parte superior derecha. Este buscador busca en las descripciones y

en los nombres de los productos. Al navegar por los productos y las categorías se va generando una '*miga de pan*' para facilitar la navegación y saber en todo momento donde se encuentra el comprador, estas facilidades se pueden ver en la siguiente imagen.

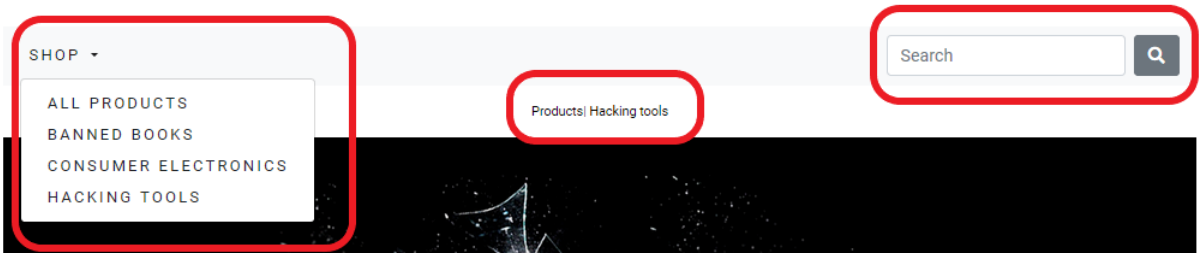

Ilustración 34 - Mecanismos de navegación en MarkeTOR

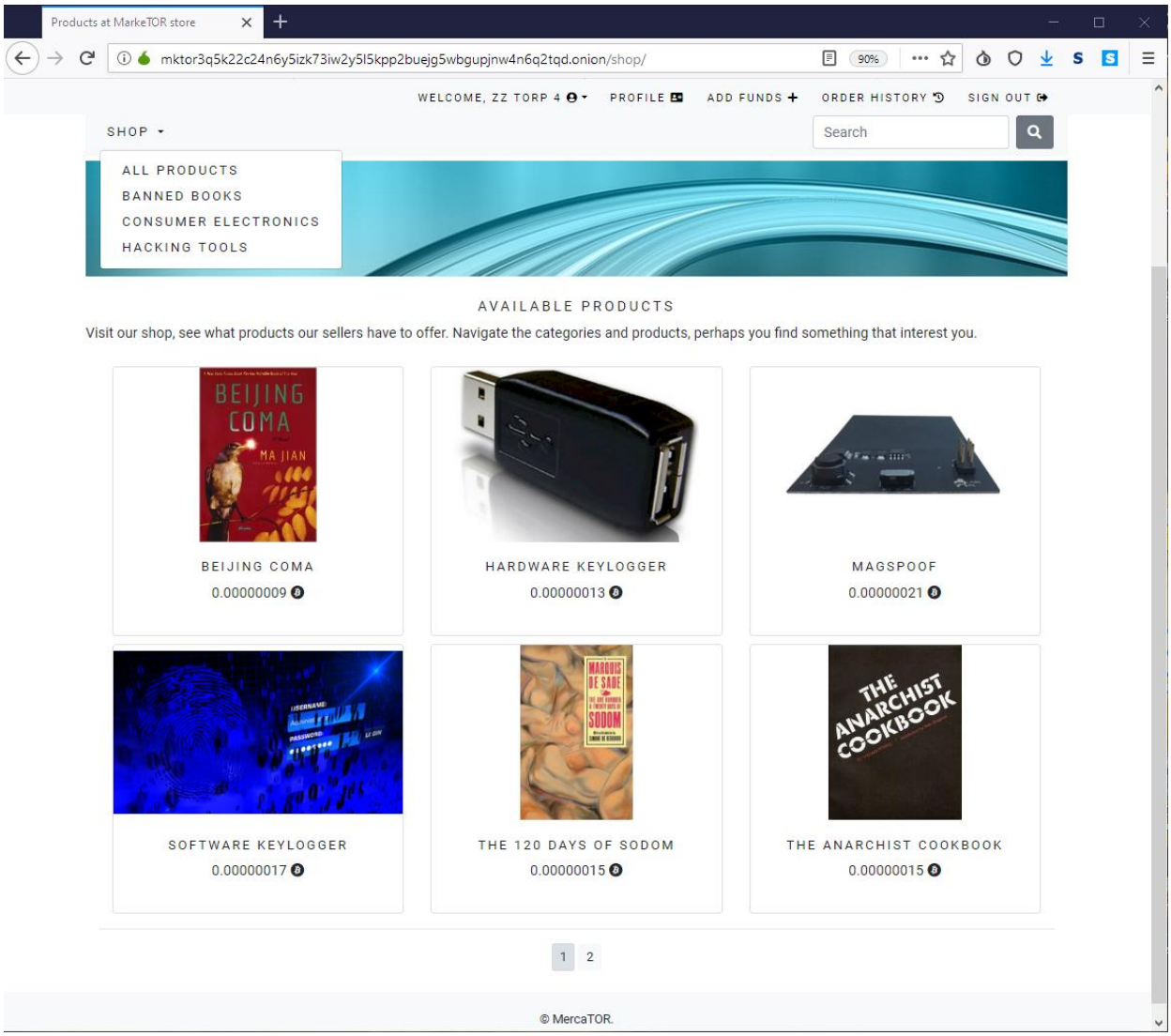

Ilustración 35 - MarkeTOR - Navegando por los productos disponibles

Si el comprador encuentra lo que está buscando puede pulsar sobre el producto para ver más detalles.

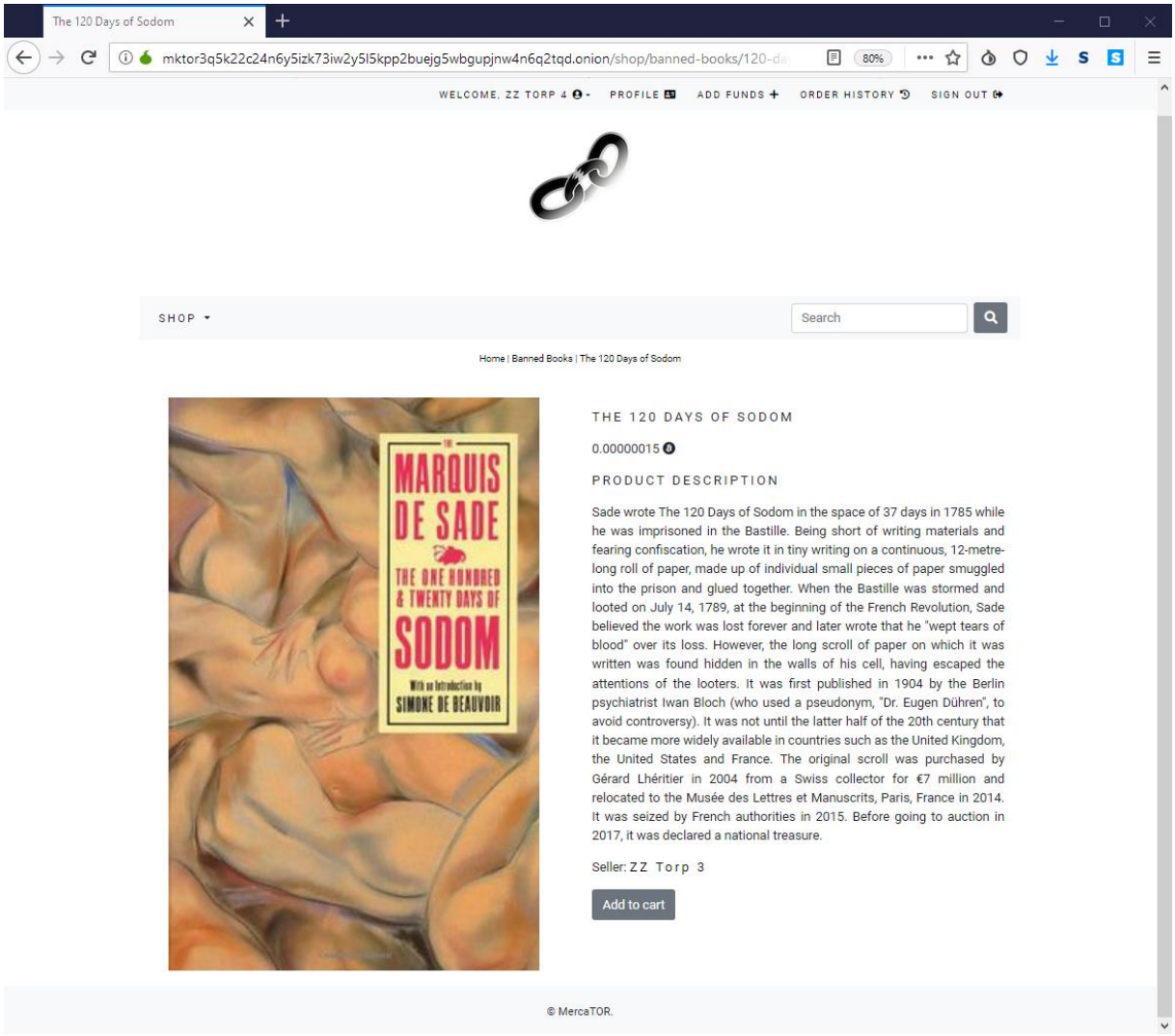

Ilustración 36 - Vista de un producto de la tienda

En la página de detalle de un producto, muestra una imagen del producto, su nombre y descripción, el precio en bitcoins y quien lo vende. Desde aquí podemos añadirlo al carrito de la compra.

El carrito, permite mantener los productos que estamos interesados en comprar y prepara la compra de forma que cada vendedor solo reciba la orden de compra de los productos que él vende. Toda esa lógica se encuentra en las aplicaciones **cart, order y bitcoin.**

Se pueden ir añadiendo productos de distintos vendedores al carrito, que podemos comprobar en cualquier momento.

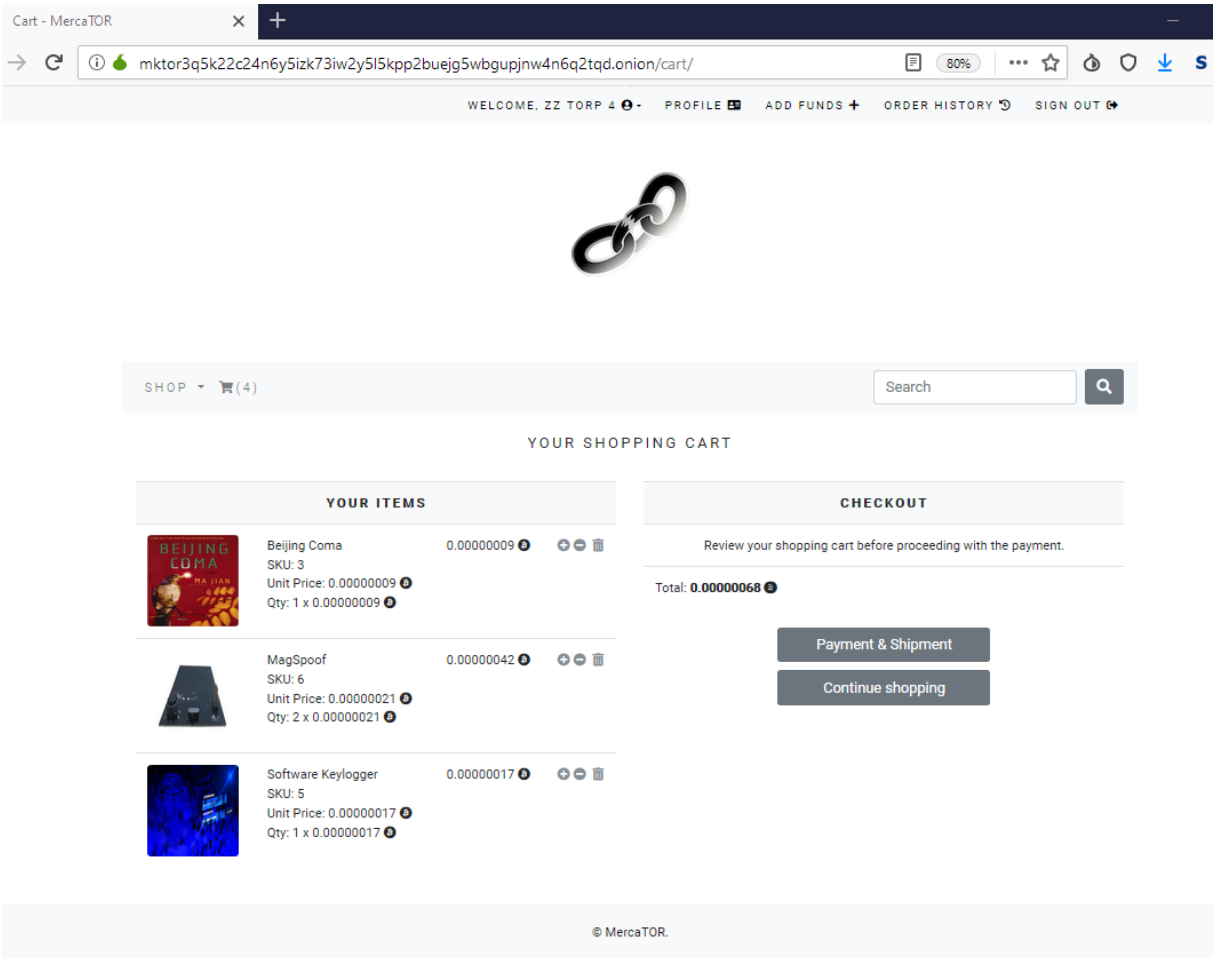

Ilustración 37 - Carrito con varios productos

Cuando tengamos en el carrito todos los productos que deseemos, podemos proceder al pago e indicar la información de envío.

La información sobre los productos comprados y la dirección de envío, se comunicará de manera cifrada, usando GnuPG[28] con infraestructura de clave pública.

Para poder pagar deberemos tener fondos en el mercado es decir tendríamos que haber subido fondos. Se comprueba si los fondos que tenemos son suficientes para hacer el pago,

[28] https://gnupg.org/

si no fuera así, un mensaje de error lo indica y se tendrían que retirar productos del carrito o añadir más fondos a MarkeTOR.

Si todo ha ido bien, en este momento se puede proceder a introducir la información de envío.

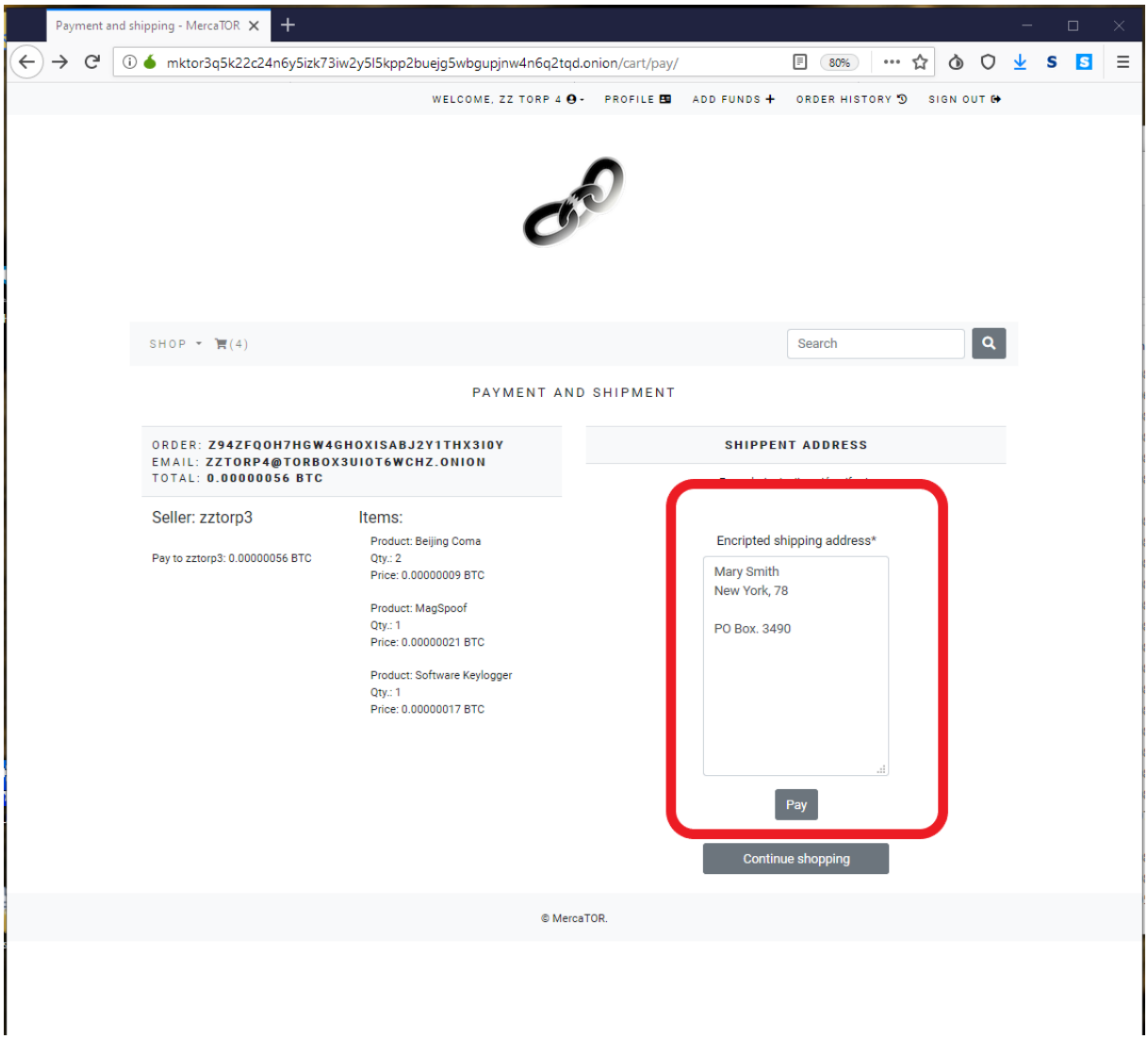

Ilustración 38 - Cumplimentando la dirección de envío y pagando

Una vez que se ha completado el pago, se generarán varios correos electrónicos cifrados:

1. Al *Comprador* con su clave pública se cifra en mensaje indicando qué ha comprado a cada *vendedor*, cuanto ha pagado a cada uno, todos los identificadores de transacciones bitcoin, una por cada vendedor.

2. A cada *vendedor* se le envía un correo cifrado con su clave pública solo con la información de sus productos y la transacción bitcoin.

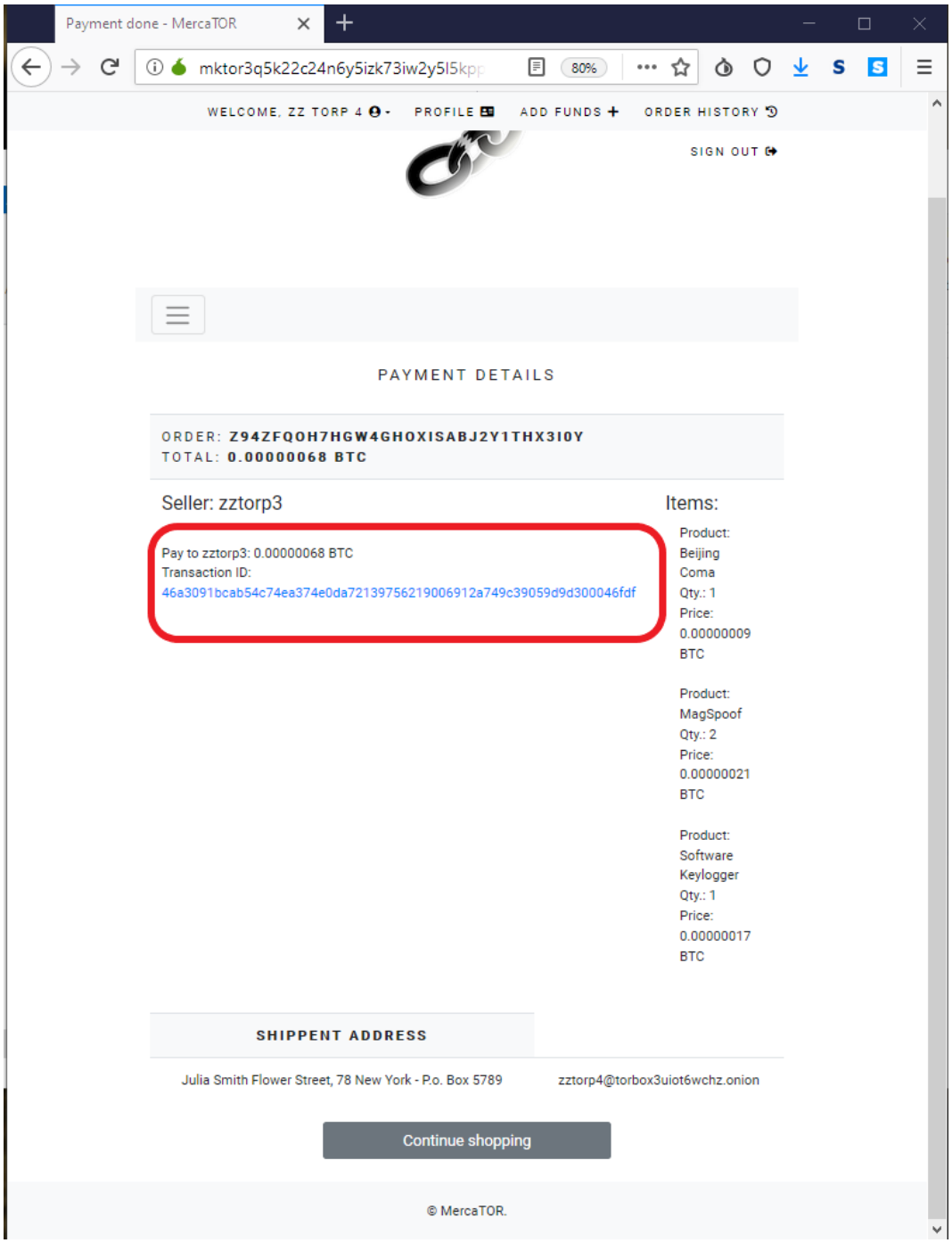

Ilustración 39 - Confirmación de Compra e información de transición Bitcoin

La siguiente imagen muestra la confirmación de compra. El punto más interesante aquí es la información sobre la transacción, que permite tanto a compradores como a vendedores comprobar si se ha realizado el pago.

Usando la URL suministrada, que también se envía en los correos cifrados a compradores y vendedores, se puede comprobar si se han completado con éxito las transacciones en la red blockchain de bitcoin.

Se puede observar la información que proporciona este servicio, cuya API[29] se utiliza internamente en el proyecto.

Las imágenes siguientes muestran una de las transacciones realizadas en la blockchain de 'test' de Bitcoin.

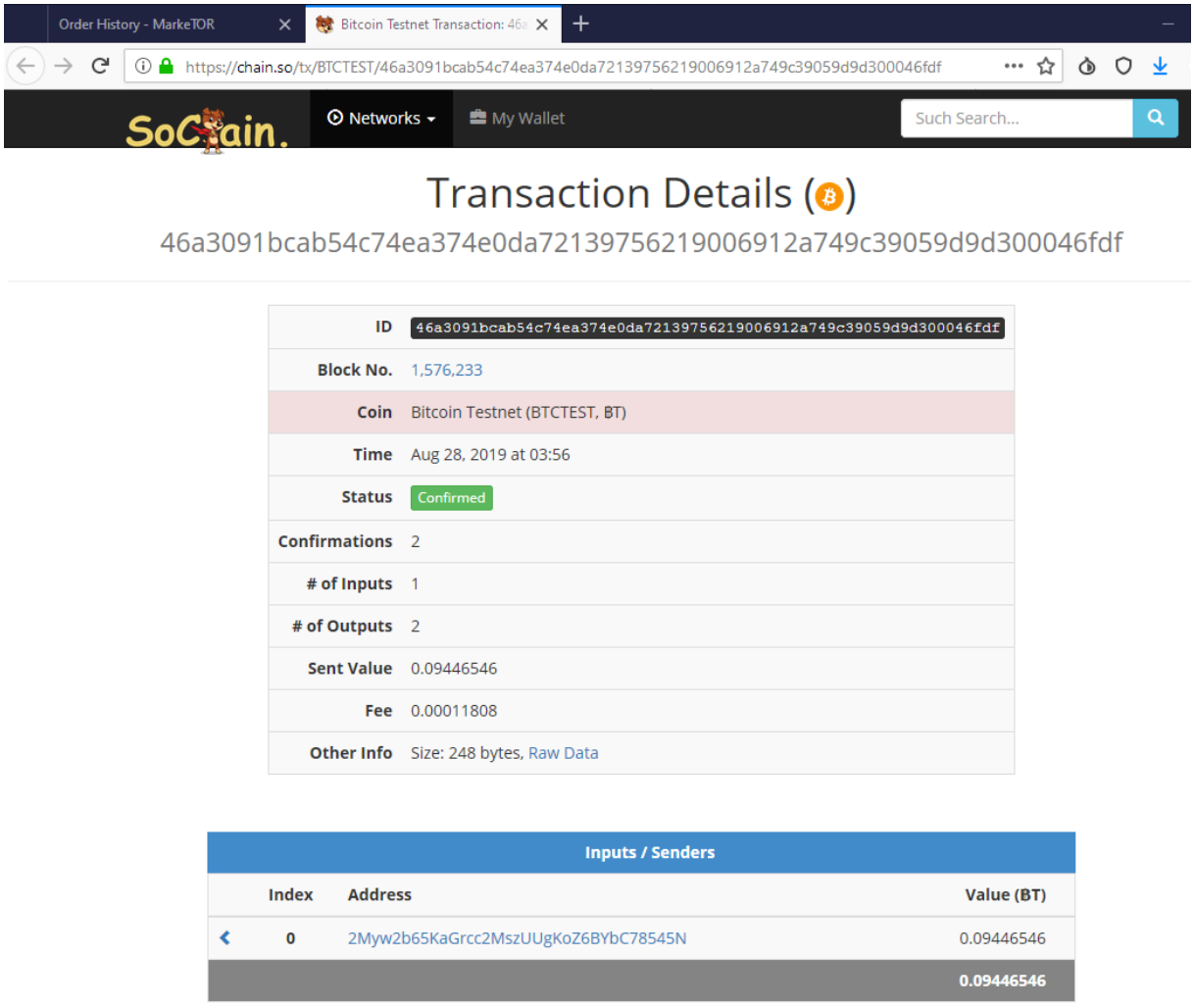

Ilustración 40 - Detalles transacción Blockchain bitcoin en TESTNET - 1

[29] https://chain.so/btc

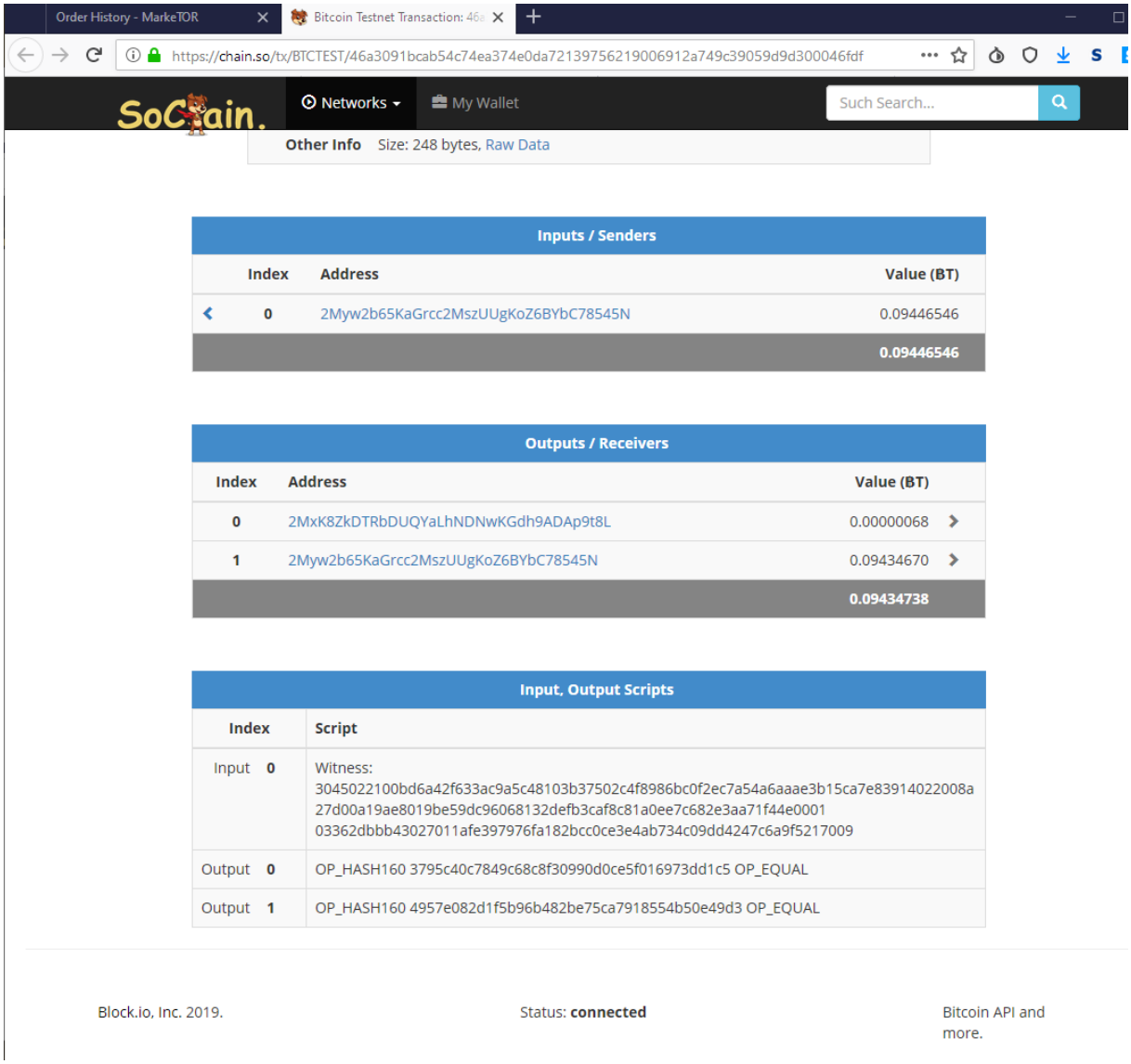

Ilustración 41 - Detalles transacción Blockchain bitcoin en TESTNET - 2

Un usuario siempre puede acceder a sus órdenes de compra una vez ha iniciado sesión, y comprobar si se han completado las transacciones en la red bitcoin, así como acceder a los detalles de la orden.

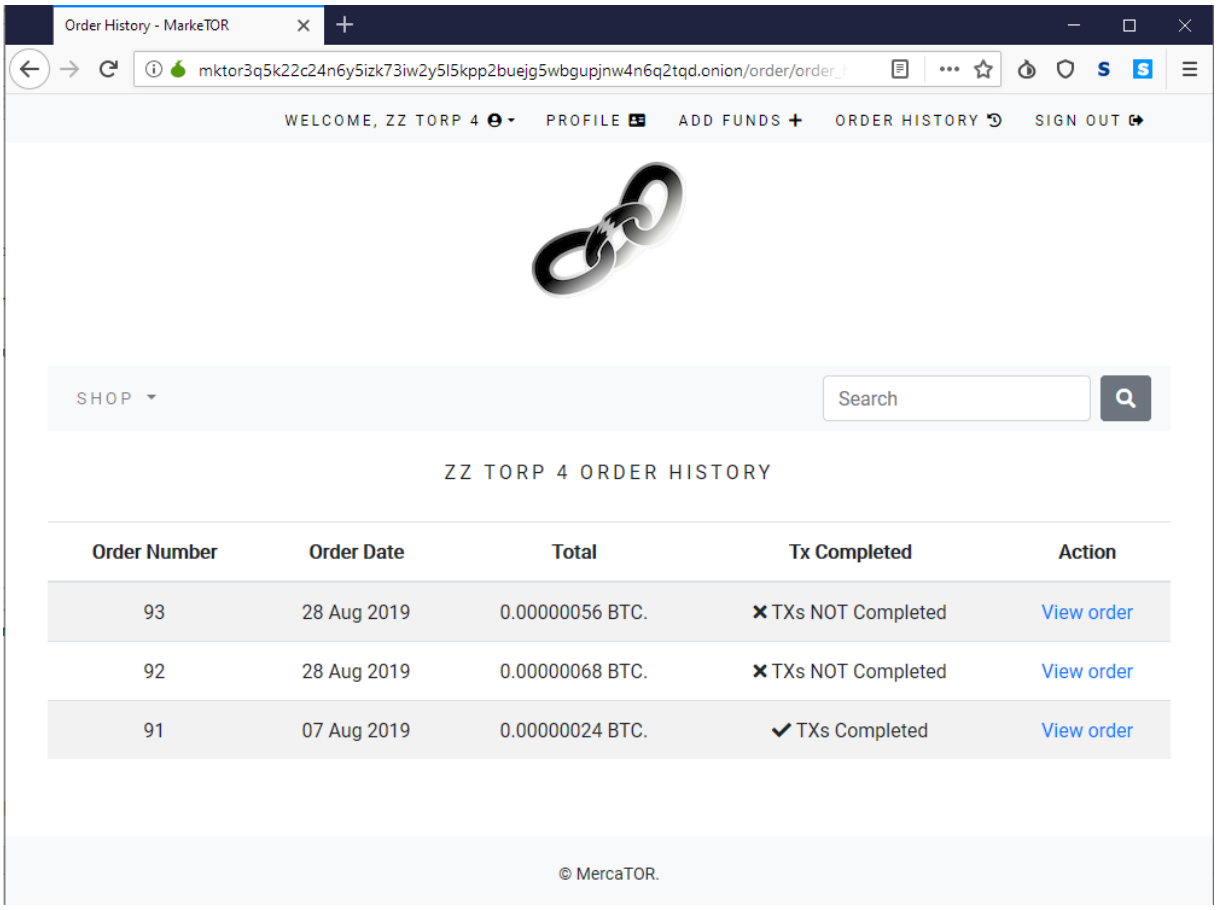

Ilustración 42 - Histórico de compras del usuario

Modelos.

En la arquitectura de la aplicación, esta parte que forma el núcleo de lo que es un mercado online, lo proporcionan varias aplicaciones Django, como son **shop, cart, order y search.** Que a su vez se apoyan en otras como **bitcoin** y **django_gpg,** para realizar todas las funcionalidades del mercado. Se revisa en este punto las primeras, dejando para puntos más específicos el resto.

Los modelos de los productos y categorías muestran la información requerida y almacenada en la base de datos para todo lo referente a lo que es posible comprar y vender en la aplicación.
El siguiente código muestra las partes más importantes de los modelos usados en las aplicaciones **shop, cart, order y search.** Este código reside en los archivos ***model.py*** de esas aplicaciones.

```
class Category(models.Model):
    name = models.CharField(max_length=150, unique=True)
    slug = models.SlugField(max_length=150, unique=True)
    image = models.ImageField(upload_to='category_photo', blank=True)
    description = models.TextField(blank=True)


class Product(models.Model):
    name = models.CharField(max_length=200, unique=True)
    slug = models.SlugField(max_length=200, unique=True)
    description = models.TextField(blank=True)
    image = models.ImageField(upload_to='product_photo', blank=True)
    category = models.ForeignKey(Category, on_delete=models.CASCADE)
    price = models.DecimalField(max_digits=16, decimal_places=8)
    stock = models.IntegerField()
    available = models.BooleanField(default=True)
    seller = models.ForeignKey(User, on_delete=models.CASCADE,
                               limit_choices_to=Q(userprofile__is_seller=True),
                               null=True)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)


class Cart(models.Model):
    cart_id = models.CharField(max_length=250, blank=True)
    date_added = models.DateField(auto_now_add=True)


class CartItem(models.Model):
    product = models.ForeignKey(Product, on_delete=models.CASCADE, related_name='products')
    cart = models.ForeignKey(Cart, on_delete=models.CASCADE)
    quantity = models.IntegerField()
    active = models.BooleanField(default=True)
```

Ilustración 43 - Extracto modelos de shop y cart

```
class Order(models.Model):
    token = models.CharField(max_length=250, blank=True)
    total = models.DecimalField(max_digits=16, decimal_places=8, verbose_name='BTC Order total',
default=0)
    email = models.EmailField(max_length=250, blank=True, verbose_name='Email Address')
    encripted_shipping_address = models.TextField(null=True)
    created = models.DateTimeField(auto_now_add=True)
    user = models.ForeignKey(User, on_delete=models.CASCADE, null=True)

    def is_paid(self):
        # Solo si todas las Subordenes están pagadas.
        return all(map(lambda x: x.is_tx_completed() is True, self.suborders.all()))


class SubOrderSeller(models.Model):
    order = models.ForeignKey(Order, on_delete=models.CASCADE, related_name='suborders')
    seller = models.ForeignKey(User, on_delete=models.CASCADE, related_name='seller')
    subtotal = models.DecimalField(max_digits=16, decimal_places=8, verbose_name='BTC
subtotal')
    created = models.DateTimeField(auto_now_add=True)
    btc_tx_id = models.CharField(max_length=64, null=True, default=None)
    btc_tx_complete = models.BooleanField(default=False)


class SubOrderItem(models.Model):
    product = models.ForeignKey(Product, on_delete=models.CASCADE,
related_name='products_suborder')
    quantity = models.IntegerField()
    price = models.DecimalField(max_digits=16, decimal_places=8, verbose_name='BTC price')
    sub_order = models.ForeignKey(SubOrderSeller, on_delete=models.CASCADE)
```

Ilustración 44 - Extracto modelos de gestión de órdenes

4.2.8 Pagos y transacciones mediante Bitcoin.

Una vez que se navega por el mercado, el usuario elegirá que productos desea adquirir. En los mercados en la red TOR son de uso común las criptomonedas. MarkeTOR utiliza como moneda para realizar pagos Bitcoin.

MarkeTOR usa en concreto la red TESTNET de Bitcoin, donde se pueden realizar las pruebas como que fuera la red principal de Bitcoin, pero aquí los Bitcoins no tiene valor real. Se usa en el desarrollo de aplicaciones que utilizan esta blockchain. El paso a la red principal es meramente una tarea de configuración básica.

No todas las criptomonedas son iguales, bitcoin es la más usada, pero si se buscan altas garantías de privacidad y anonimato por defecto (Juhász, Stéger, Kondor, & Vattay, 2018)

(Goldfeder, Kalodner, Reisman, & Narayanan, 2018), existen otras criptomonedas pensadas desde un inicio para la privacidad son por ejemplo Monero[30] (XMR), Dash[31] o Bitcoin Private[32] (CTCP) entre otras.

MarkeTOR utiliza para gestionar los pagos con Bitcoins y los monederos las herramientas:

1. El paquete Python '**Bit**' (Lev, 2019), que es una librería para el uso de bitcoins y otros 25 tipos de criptomonedas que permite, entre otras, operaciones como:

    - Gestión de monederos y claves
    - Firmas digitales (RFC 6979),
    - Acceso a la blockchain,
    - Uso de cuotas (fee),
    - Soporte de direcciones Segwit, y clásicas
    - Representaciones de claves privadas mediante

2. **Bitcoin SmartBit**[33]. Un explorador de la blockchain de Bitcoin, se usa en MarkeTOR para seguir las transacciones y comprobar su estado.

3. El '**API Sochain'**[34], que permite acceder a la blockchain de varias criptomonedas, entre ellas Bitcoin. Permite interrogar la blockchain correspondiente para obtener información sobre transacciones con criptomonedas, información de la red, gestión de direcciones etc. En el proyecto se usa para generar enlaces a la información asociada a una transacción de bitcoins.

Funcionamiento de los pagos en MarkeTOR.

MarkeTOR cuanta con un monedero, desde el que se hacen todas las transferencias monetarias. De cara a lo que se pretende en el proyecto sirve perfectamente, pero en

[30] https://www.investopedia.com/terms/m/monero.asp
[31] https://www.investopedia.com/terms/d/dash.asp
[32] https://www.investopedia.com/terms/b/bitcoin-private-btcp.asp
[33] https://www.smartbit.com.au/api
[34] https://my.dogechain.info/api

situaciones de máxima privacidad se podrían usar múltiples monederos y haber usado técnicas de Bitcoin tumbling o mixing[35], pero queda fuera del alcance de este trabajo.

El funcionamiento propiamente dicho es el siguiente. Los compradores, cuando se dan de alta, para poder comprar necesitan tener unos fondos (bitcoins) disponibles. La forma de tener disponibles bitcoins para su gasto en el mercado es mediante la subida de fondos. Para ello se utiliza un sistema sencillo.

Se debe de hacer una transacción con los fondos en bitcoins deseados a la dirección del monedero del mercado y comunicar a los administradores el identificador de la transacción mediante un correo cifrado con la clave pública del correo de los operadores de MarkeTOR, cuya dirección es (*marketor at torbox3uiot6wchz.onion*). Dirección que solo puede ser accesible desde la red TOR.

Los operadores comprobarán la validez de la transacción y si se ha completado con éxito, en ese momento añadirán al perfil esa cantidad como disponible para gastar en MarkeTOR. Se puede observar como este valor se almacena en el modelo del perfil de usuario.

Cuando el usuario decide comprar productos, se restarán de los sus fondos. Si el usuario compra varios productos a varios vendedores, se realizarán tantas transacciones de bitcoins, como vendedores existan en la orden de compra. De esta forma, los vendedores reciben transacciones independientes.

Se envía un correo cifrado a todos los implicados, comprador y vendedores, usando sus claves públicas e informándoles de los identificadores de transacciones que deban ver y como consultarla, de forma pueden saber si se ha completado la transición y cuando se ha completado con éxito.

Creación de monederos para los usuarios de prueba de MarkeTOR.

MarkeTOR, necesita un monedero de bitcoin para recibir los fondos de los compradores y realizar el pago a los vendedores. Se podría haber optado por un monedero creado de forma

[35] Proceso de usar un servicio de terceros para romper la conexión entre una dirección de Bitcoin que envía monedas y las direcciones a las que se envían.

online, usando los múltiples servicios disponibles tanto en la *Clearnet* como en la *Deep Web/Dark Web*.

Una de las premisas de este proyecto, es la privacidad de todos los actores implicados. Esta cartera es usada por los operadores del mercado de forma automática y para añadir una capa más de privacidad y seguridad se decidió crear en monedero en local. De forma que la clave privada que genera este monedero no sale del equipo en el que se monta el sistema, pero a su vez puede operar sin problemas de forma online. Los sistemas que crean monederos en la nube podrían tener realmente acceso a la clave privada del monedero, por lo que no se optó por esta opción.

Los usuarios de la aplicación pueden usar el monedero Bitcoin que consideren oportuno, en este proyecto los usuarios que se han usado de prueba han seguido el mismo proceso que para la generación del monedero principal de MarkeTOR.

Todas las operaciones en MarkeTOR se han realizado en **satoshi[36],** para evitar problemas de redondeo usando Bitcoin (BTC) directamente. Un Satoshi es la unidad mínima de medida que se puede utilizar en el sistema Bitcoin, es decir la fracción más pequeña en la que se puede dividir un Bitcoin. Esta fracción mínima sería 0,00000001 BTC y recibe el nombre de **satoshi** en honor al seudónimo usado por el creador de Bitcoin Satoshi Nakamoto.
Por lo tanto, las equivalencias entre las unidades de medida satoshi y bitcoin

- 1 satoshi = 0,00000001 bitcoins (BTC)
- 1 bitcoin (BTC) = 100,000,000 satoshi

La creación monederos se ha realizado mediante la librería Python bit, como ya se ha indicado. Los siguientes fragmentos de código, limpios de código de control y gestión de errores, muestran algunas de las operaciones básicas y sus salidas.

[36] https://academy.bit2me.com/que-es-un-satoshi/

```
from bit import PrivateKeyTestnet, SUPPORTED_CURRENCIES
from os.path import exists


def write_wif_to_file(file, key: PrivateKeyTestnet):

    if not exists(file):
        with open(file, 'w') as f:
            f.write(key.to_wif())
            return True
    else:
        return False



    # Crear Wallet (menedero)
    # Monedero en TESTNEt,m para la Blockchain real de Bitcoin usar PrivateKey()
    k2 = PrivateKeyTestnet()

    print("k2 WIF: {}".format(k2.to_wif()))
    print("k2 address: {}".format(k2.address))
    print("k2 segwit_address: {}".format(k2.segwit_address))
    print("k2 balance inicial: 0 satoshi")
    print("k2 balance: {}".format(k2.get_balance()))

    write_wif_to_file('k2t_wif', k2)
```

Ilustración 45 - Código ejemplo creación de monedero Bitcoin

El resultado de la ejecución de ese código sería:

```
k2 WIF: [redacted]
k2 address: n11zbPN69Ufy2n7Aji9cVMd7iz6fGGTHu7
k2 segwit_address: 2Myw2b65KaGrcc2MszUUgKoZ6BYbC78545N
k2 balance inicial: 9557701 satoshi
k2 balance: 9430396
Desde Archivo
```

Ilustración 46 - Creación de Monedero BTC desde WIF

Pagos y transacciones Bitcoin en MarkeTOR.

Se ha creado en MarkeTOR una clase llamada **BTCWallet** que se encarga de realizar los pagos y mantener los datos de los posibles monederos de la aplicación. Reside en la aplicación Django **'bitcoin'.** En MarkeTOR, un usuario puede ver su historio de órdenes y puede observar si se han completado o no las transacciones bitcoin correspondientes, es decir si se ha asentado su transacción en la blockchain Bitcoin.

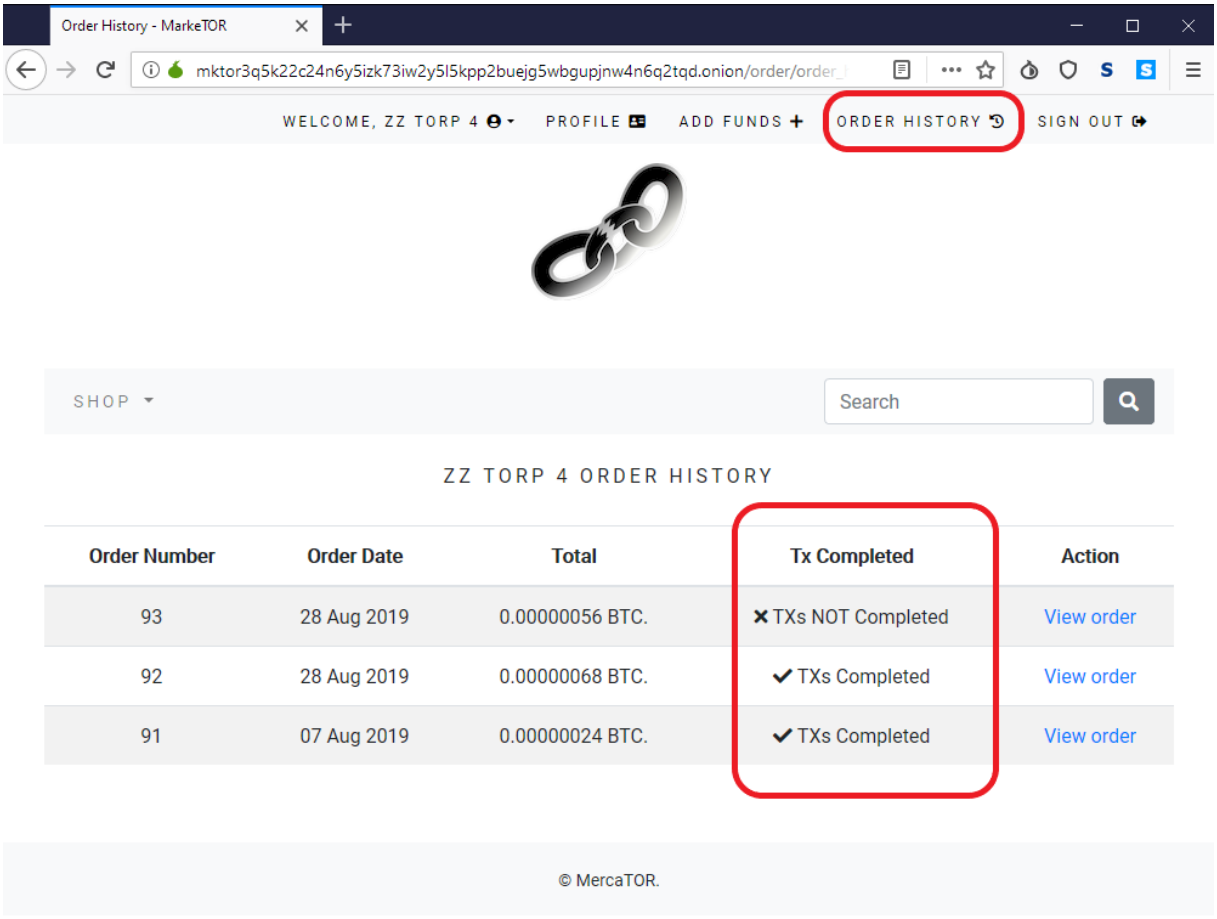

Ilustración 47 - Listado de órdenes de un usuario

MarkeTOR, cuando se tienen productos en el carrito, permite realizar la compra añadiendo a la blockchain tantas transacciones como vendedores intervienen.

El proceso se ha explicado anteriormente desde el punto de vista del usuario. Resumiendo, se agrupan las *órdenes* de compra en *subórdenes* por vendedor y se realiza un pago a cada uno. A su vez se envían correos electrónicos cifrados al vendedor con toda la información y a los distintos vendedores con la información de los subórdenes relevantes para ellos.

Este proceso lo realiza la aplicación Django '**cart'**, mediante la llamada a la función '**cart_payment'** del módulo '**cart.views',**

Un extracto del código desea función, donde se puede apreciar la funcionalidad comentada:

```
# pago, transacciones y mails... REVISAR
wallet: BTCWallet = BTCWallet(MARKET_WIF)

# Pagamos ....
for so in suborders:
    segwit = so.seller.userprofile.seller_segwit_address
    output = [(segwit, str(so.subtotal * (10**8)), 'satoshi'), ]  # 1 BTC = 10**8  satoshi
    so.btc_tx_id = wallet.send_bitcoin(output)
    # Bajar stock producto
    so_items = SubOrderItem.objects.filter(sub_order=so)
    for item in so_items:
        item.product.stock -= item.quantity
        item.save()
    so.save()
    send_encripted_mail(rol=UserTypeMail.seller,
                        order=order,
                        suborder=so)

send_encripted_mail(rol=UserTypeMail.buyer,
                    order=order,
                    context=context)
# Eliminar carrito
cart.delete()

return render(request, 'done.html', {'order': order, 'subordenes': context})
```

Ilustración 48 - Fragmento código pago con BTC de una compra en MarkeTOR

La comprobación de las transacciones en la blockchain puede hacerlas el usuario, tanto comprador como vendedor, con la información cifrada sobre las órdenes enviadas por correo.

El comprador, adicionalmente puede observar en el listado de órdenes y en el destalle de cada orden si se ha completado la transacción, además de tener acceso a un enlace que le permite seguir dicha transacción.

La siguiente imagen muestra un detalle de una orden de compra de un usuario, así como el identificador y enlace de la transacción.

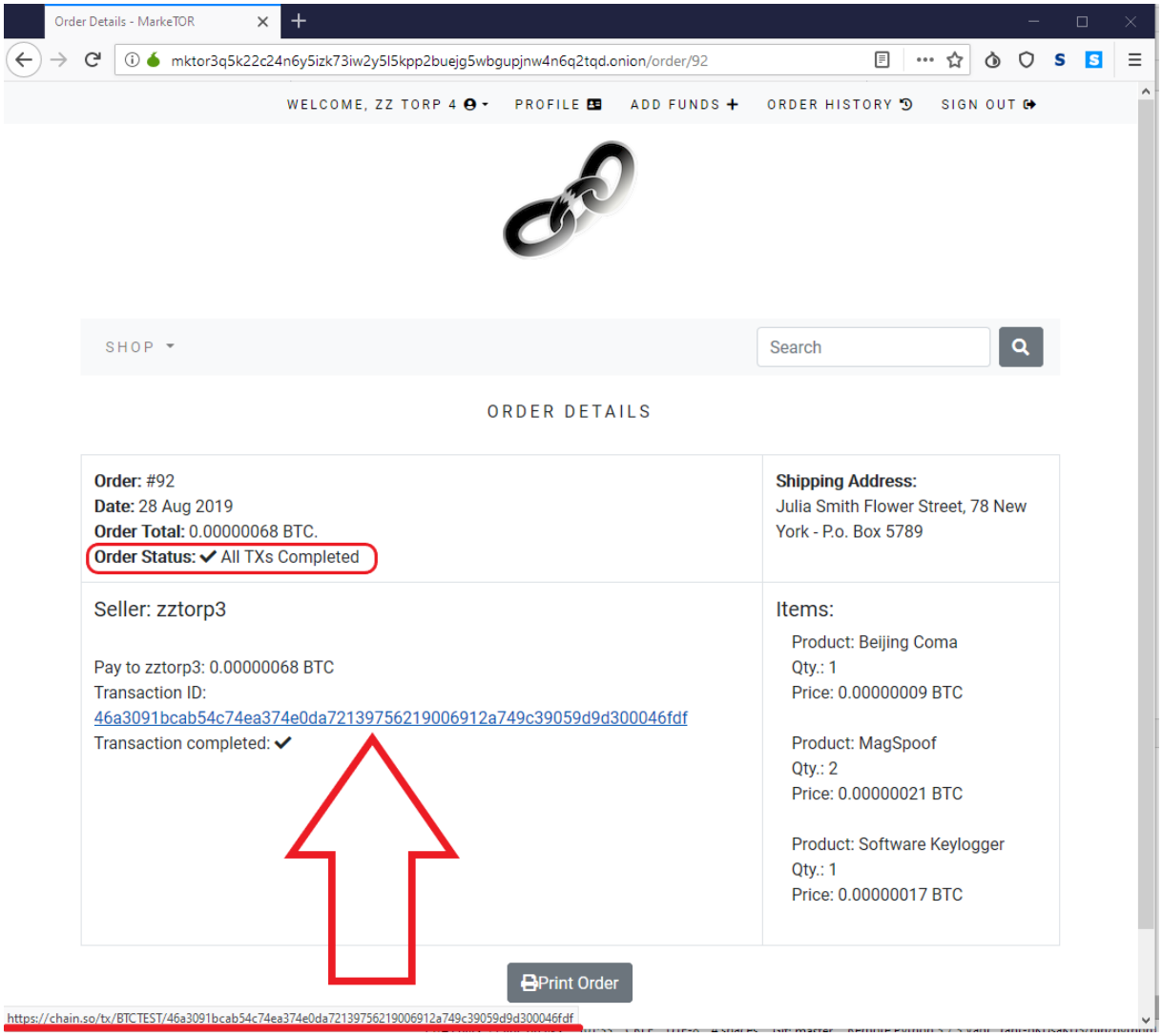

Ilustración 49 - Detalle orden, estado e ID de transacción Bitcoin

El ID de transacción se recoge en el pago y se almacena en el modelo de las *subórdenes*. Cuando se muestra el histórico de órdenes y el detalle, se comprueba también el estado de las transacciones. Esto se hace solamente si existe alguna transacción pendiente. Las transacciones ya completadas no se vuelven a comprobar.

Esta funcionalidad la realiza el modelo de **subórdenes**, que almacena el estado de las transacciones agrupadas como se ha indicado por vendedor. El siguiente fragmento de código que se encuentra en el módulo **models.py** de la aplicación Django **'order'**, contiene un método que es el encargado de comprobar las transacciones y actualizar su estado en la base de datos.

```
class SubOrderSeller(models.Model):
    order = models.ForeignKey(Order, on_delete=models.CASCADE,
related_name='suborders')
    seller = models.ForeignKey(User, on_delete=models.CASCADE, related_name='seller')
    subtotal = models.DecimalField(max_digits=16, decimal_places=8, verbose_name='BTC
subtotal')
    created = models.DateTimeField(auto_now_add=True)
    btc_tx_id = models.CharField(max_length=64, null=True, default=None)
    btc_tx_complete = models.BooleanField(default=False)

    def is_tx_completed(self):
        """Comprobamos si en la blockchain esta completada esta transacción """
        if self.btc_tx_complete is True:
            return True
        else:
            if self.btc_tx_id:
                # completed: bool = services.check_btc_tx(red='BTCTEST', tx=self.btc_tx_id)
                completed: bool = services.check_btc_tx_smartbit(tx=self.btc_tx_id)
                if completed:
                    self.btc_tx_complete = True
                    self.save()
                    return True
                else:
                    return False
            else:
                return False
```

Ilustración 50 - Modelo Subórdenes y comprobación de la transacción

La comprobación se delega al módulo **services.py** de la aplicación **bitcoin**, donde residen varias funciones para la gestión de los monederos y las transacciones.

```
@dataclass
class BTCWallet:
    wif: str
    _pk: field(init=False, repr=False) = None

    def __post_init__(self):
        self._pk = PrivateKeyTestnet(self.wif)

    @property
    def private_key(self) -> PrivateKeyTestnet:
        return self._pk
    @property
    def balance_in_satoshi(self):
        return self._pk.get_balance()
    @property
    def balance_in_bitcoin(self):
        return self._pk.get_balance(currency='btc')

    def send_bitcoin(self, outputs: Outputs) -> str:
        # Envía a la red  las ope
        return self.private_key.send(outputs)

def check_btc_tx_smartbit(tx: str) -> bool:
    url = f'{API_SMARTBIT}/{tx}'
    response = requests.get(url)
    if response.status_code == 200:
        json = response.json()
        return json['success']
    else:
        return False

def is_address_valid(red: str, addr: str) -> bool:
    """ Indica si la dirección es valida en la red especificada """
    url = f'{API_CHAIN_SO}is_address_valid/{red}/{addr}'
    response = requests.get(url)
    if response.status_code == 200:
        json = response.json()
        return parse_bool(json['data']['is_valid'])
    else:
        return False

def value_received_from_address(red: str, addr: str) -> Dict:
    """ Indica el total recibido por la dirección indicada en la red indicada """
    url = f'{API_CHAIN_SO}get_address_received/{red}/{addr}'
    response = requests.get(url)
    if response.status_code == 200:
        json = response.json()
        return dict(estado='OK',
                    confirmed_received_value=Decimal(json['data']['confirmed_received_value']),
                    unconfirmed_received_value=Decimal(json['data']['unconfirmed_received_value']),)
    else:
        return dict(estado='ERROR')
```

Ilustración 51 - Clases y funciones. bitcoin/services.py

4.2.9 Comunicaciones privadas en MarkeTOR: GnuPG.

Desde el principio de la concepción de este proyecto la privacidad es uno de los pilares fundamentales. Un mercado en el que se pueden comprar productos a diferentes vendedores requiere que la información fluya para comunicar información sobre qué se ha comprado y donde enviarlo, pero solamente se debe comunicar la información relevante. Si el comprador adquiere productos de tres vendedores, cada vendedor solo tiene interés en lo que se le ha comprado a él. No necesita saber nada sobre los productos que se han comprado a otros. El vendedor, sin embargo, necesita saber toda la información de lo que ha comprado.

En MarkeTOR, La comunicación de estas informaciones, que en definitiva son las órdenes y las subórdenes de compra y los productos adquiridos, se comunican usando correo electrónico cifrados con infraestructura de clave pública. Es decir, se envían correos desde una cuenta de correo perteneciente a MerkaTOR, cifrando el mensaje con las claves públicas de los destinatarios.

Los correos a los compradores con la información completa de la orden se cifran con la clave pública del comprador, que es uno de los campos del perfil de usuario.

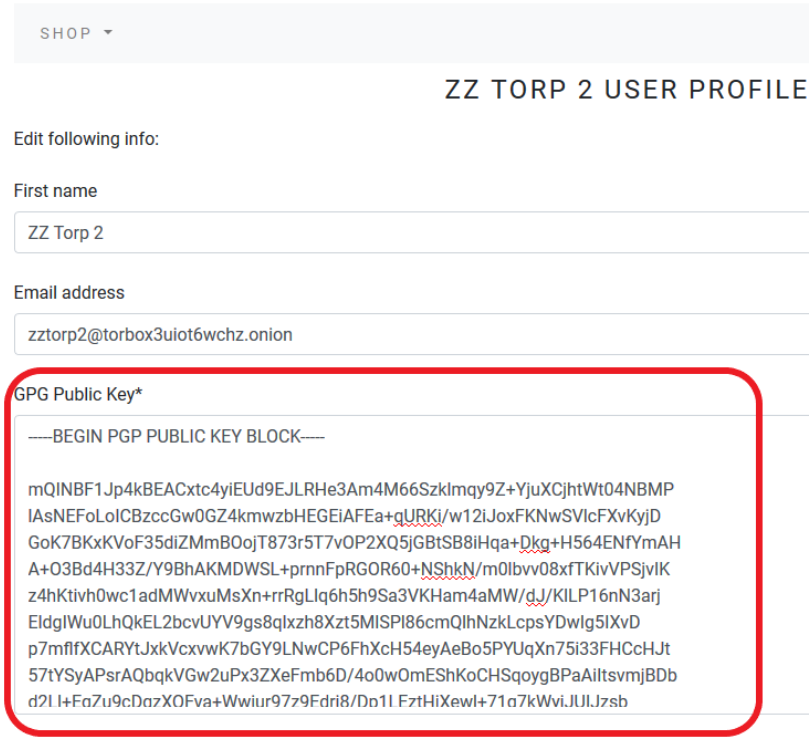

Ilustración 52 - Clave publica del usuario, usada en el cifrado de los correos.

De la misma forma a los vendedores, se les envía otro correo también cifrado con su clave pública.

Correo electrónico en TOR.

Se indico ya anteriormente, que MarkeTOR usaría solo correos dentro de la red TOR, para ello se debe de buscar un proveedor de correo que sea un servicio onion y permita el envío y recepción de correo solamente en TOR.

Un servicio que permite hacer esto es **TorBox** (http://torbox3uiot6wchz.onion/) , su sitio en TOR indica que es un servicio oculto, solamente accesible desde TOR y no tiene conexión entre la red TOR e Internet.

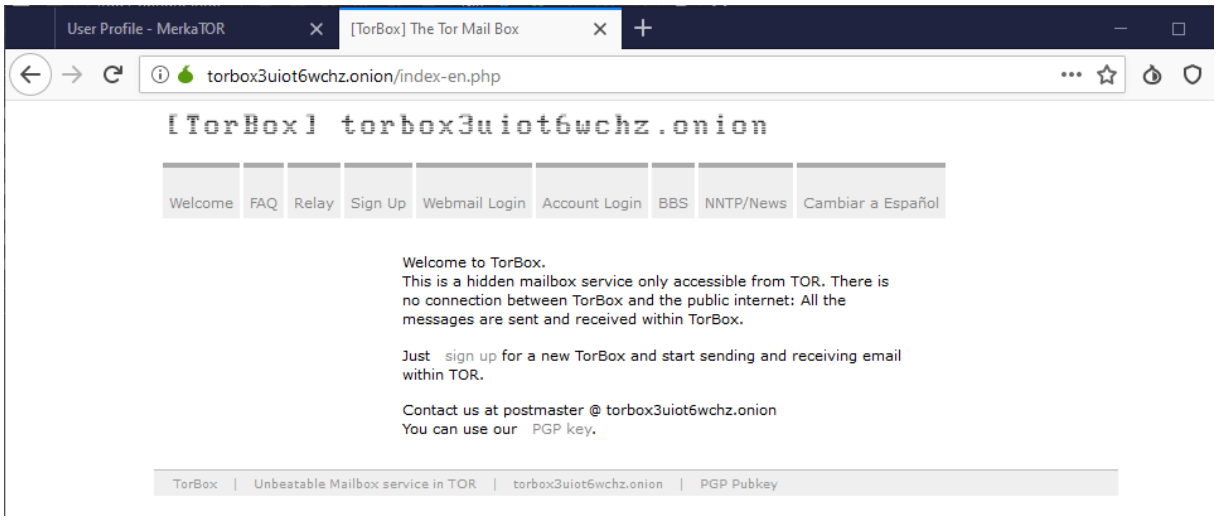

Ilustración 53 - Servicio TorBox. Correo electrónico en TOR

MarkeTOR utiliza una cuenta en TorBox, para enviar los correos electrónicos cifrados a los distintos usuarios. TorBox permite configurar de forma sencilla SMTP, de forma que la aplicación MarkeTOR, usando la funcionalidad de Django para envío de correo puede enviar correos de forma automática.

Crear una cuenta en este servicio es sumamente sencillo, pulsamos en '**Sign Up**', introducimos un nombre de usuario válido y una contraseña. A partir de ese momento ya tenemos una cuenta de correo en TOR, la siguiente imagen muestra cómo se realiza dicha operación:

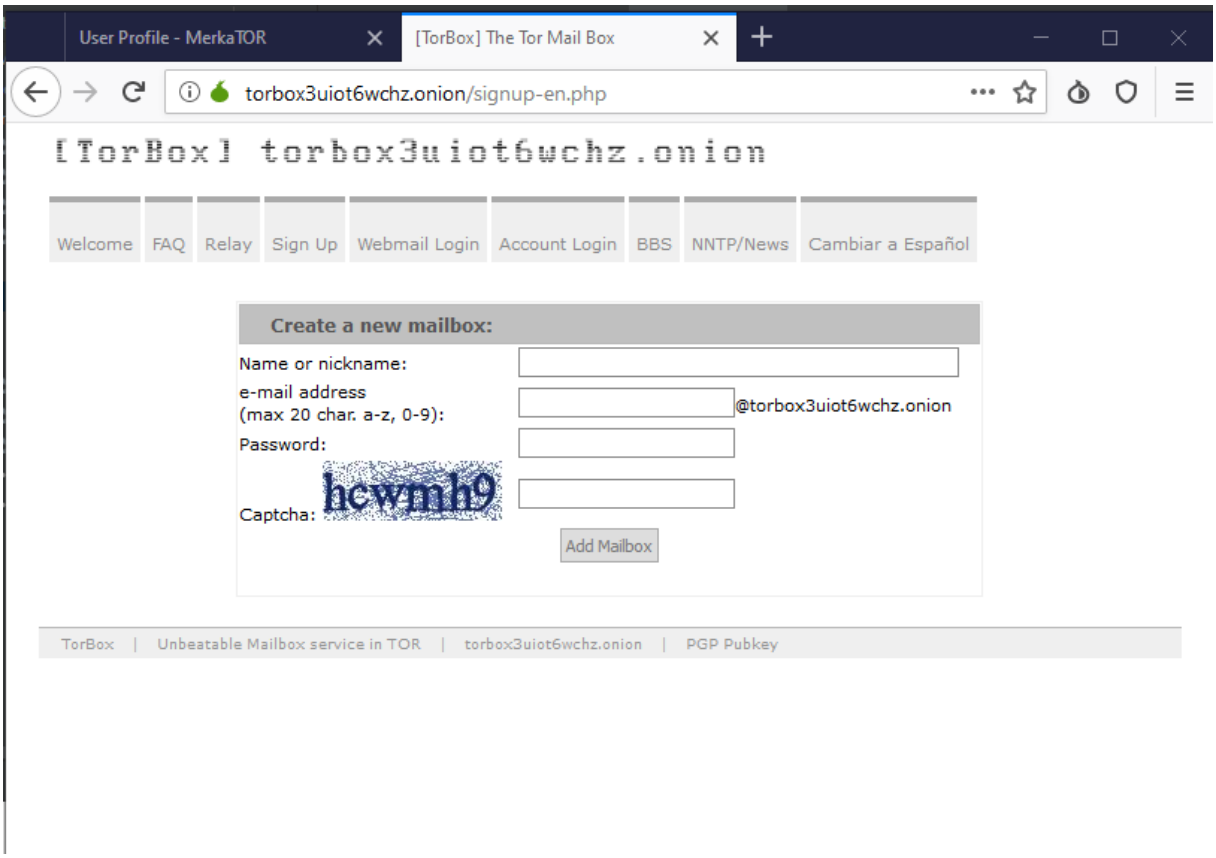

Ilustración 54 - Crear cuenta de correo en TorBox

El usuario "**ZZ Torp 4**", un usuario de MarkeTOR para realizar las pruebas de compra, venta y cifrado de mensajes, utiliza como cuanta de correo una en TorBox, si el usuario entra en su bandeja de entrada podrá ver los mensajes enviados por MarkeTOR con la información de las compras o ventas que ha realizado.

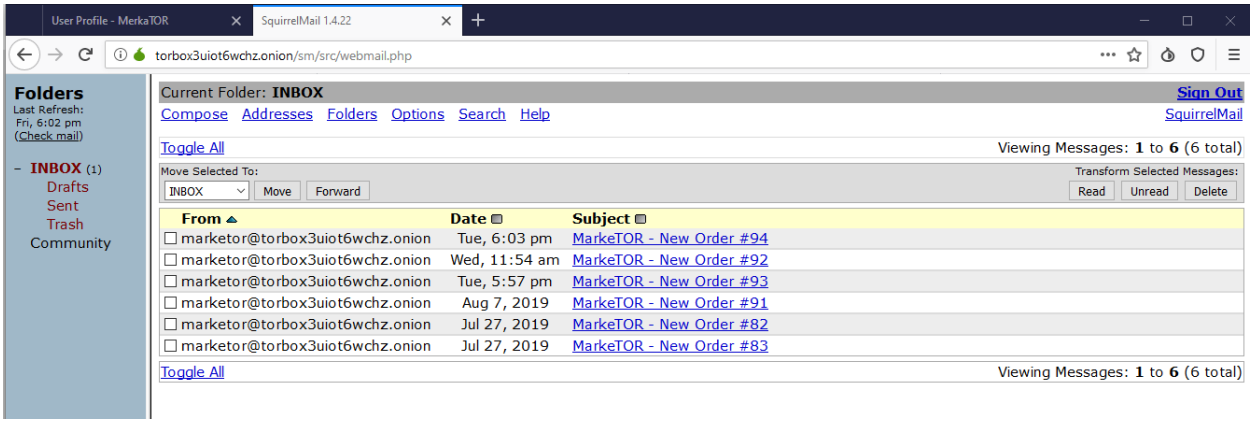

Ilustración 55 - Correos recibidos por el usuario ZZTorp 4 del mercado en TorBox

Se pueden observar las distintas órdenes de compra/venta que tiene en la bandeja de entrada. Esos correos electrónicos están cifrados y para poder leerlos tendrá que descargarlos y usando su clave privada los descifrará y leerá.

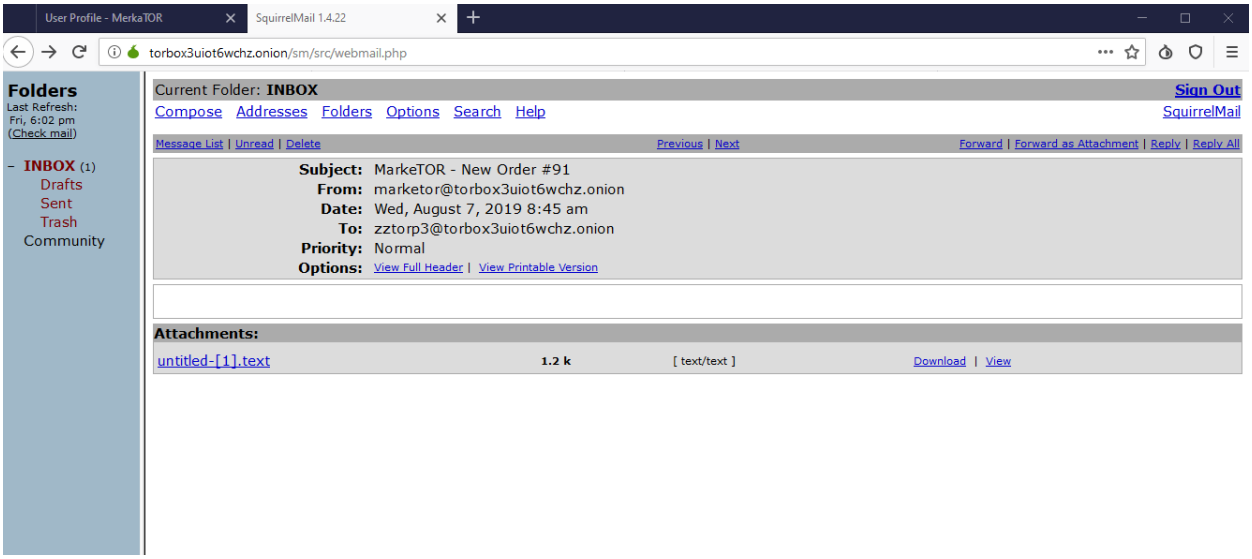

Ilustración 56 - Mensaje cifrado 1

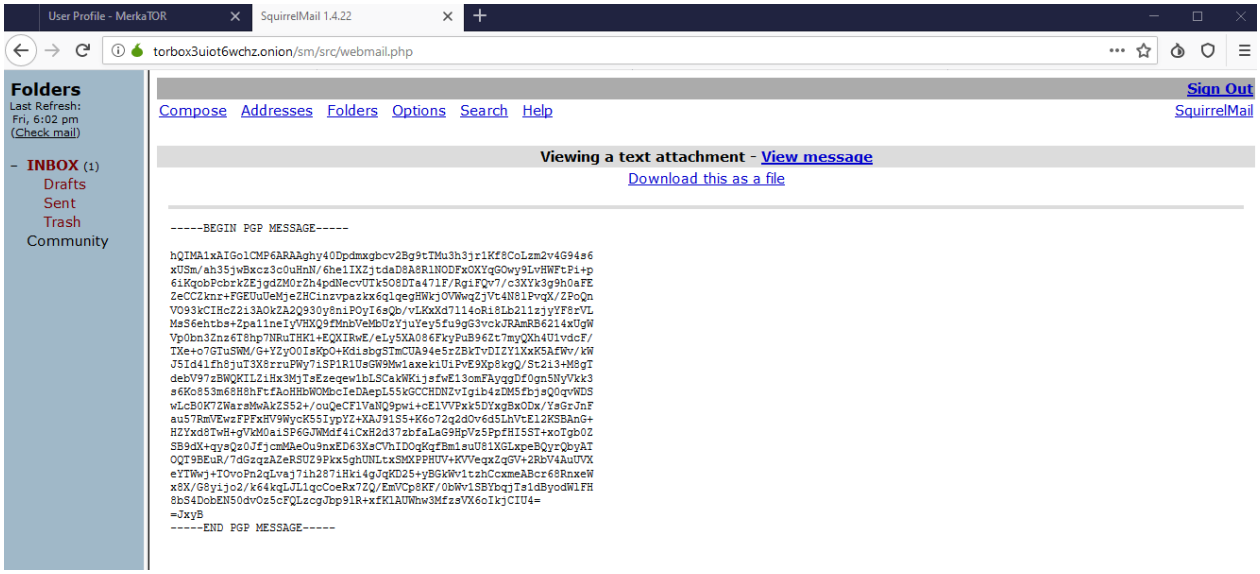

Ilustración 57 - Mensaje cifrado 2

Si el usuario o cualquier adversario que accediese a su correo quisiera leer el contenido de estos correos, necesitaría tener a su disposición, la clave privada del usuario '**ZZ Torp 4**' para poder acceder a su contenido.

Para descifrar dicho mensaje usaremos GnuPG, desde una máquina donde tenemos instalada la clave privada correspondiente a la clave pública con la que se cifró el mensaje.

Se puede observar en la siguiente imagen como accedemos al contenido descifrando el mensaje.

```
vagrant@contrib-buster:/vagrant/keys/mensajes$ gpg --decrypt 'MarkeTOR - New Order #91_buyer_zztorp4.text'
gpg: encrypted with 4096-bit RSA key, ID 14DD1397A8E9A990, created 2019-08-06
      "ZzTorp 4 <zztorp4@torbox3uiot6wchz.onion>"
New Order #91 - MarkeTOR

Thanks for shopping with us
This email is to confirm that you have placed an order on MarkeTOR.
Please make sure that all the details of your order are correct.

Order: e8tdnjmmcxxckwy9lf036vn1mttndlb2
Total: 0.00000024 BTC


  - Seller: zztorp3
  - Pay to Seller: 0.00000024 BTC
  - TX: 548f0cf48257bca21706586da45c6d74233dfb623a8d4b3ca11a6a814d0544f3 - https://chain.so/tx/BTCTEST/548f
0cf48257bca21706586da45c6d74233dfb623a8d4b3ca11a6a814d0544f3
  - Tx Completed: False
    Items:

        Product: Beijing Coma
        Qty.: 1
        Price: 0.00000009 BTC

        Product: The Anarchist Cookbook
        Qty.: 1
        Price: 0.00000015 BTC



Shipment Address:
ZZtorp 4
Area 51
```

Ilustración 58 - Descifrando el correo recibido con la clave privada correspondiente

La imagen muestra el uso de GnuPG, en concreto el comando ***gpg,*** para descifrar en mensaje recibido mediante correo electrónico. Se aprecia que se ha usado RSA de 4096 bit, y el texto descifrado:

- Los productos que compró el usuario
- Donde se deben de enviar
- El ID de transacción, su estado (en el momento del envío del correo) y el enlace para seguir as la transacción bitcoin.

El usuario, si accede a su historial de compra, puede ver que la información es la misma que se le envío por correo electrónico.

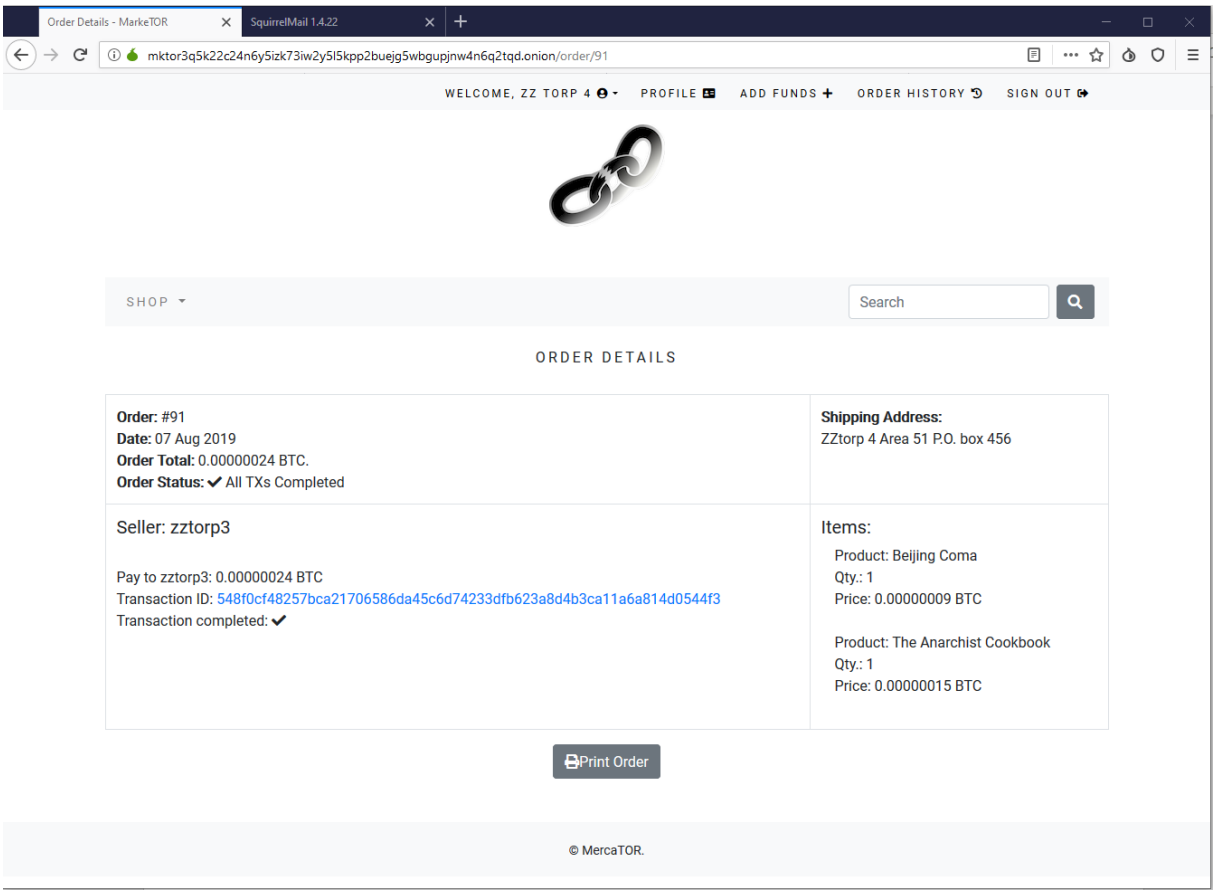

Ilustración 59 - Orden de compra desde MarkeTOR correspondiente al mensaje cifrado.

Creación de claves de cifrado con GnuPG.

Los usuarios pueden crearse sus propias claves de cifrado usando GnuPG. Prácticamente todos los sistemas operativos tienen soporte a través de aplicaciones de GnuPG, software que nos permite crear claves, cifrar y descifrar textos con esas claves. Todas las claves generadas para su uso con MarkeTOR, se han realizado desde GNU/Linux usando el comando ***gpg***.

Para crear un par de claves, se utiliza el comando:

```
$ gpg --full-generate-key
```

Con ese comando desde cualquier distribución Linux, podemos generar un par de claves pública y privada para utilizar en MarkeTOR.

Otros comandos importantes en el uso de las claves es la importación y exportación de estas, como la exportación de la clave pública es importante, para poder hacerla accesible a los usuarios y que la puedan usar para ponerse en contacto con nosotros. En el caso de MarkeTOR, la clave pública se espera en formato '**ASCII-armored**', es decir codificada en Base-64.

El siguiente comando, exporta la clave publica asociada a un archivo llamado *marketor_pub.key.asc* para su distribución:

```
$ gpg --output marketor_pub.key.asc --armor --export
marketor@torbox3uiot6wchz.onion
```

El archivo creado contiene la clave pública que se puede copiar y pegar en el campo '**GPG Public Key'** de MarkeTOR. A partir de ese momento la aplicación de puede comunicar de forma segura y privada con ese usuario.

Envío de correo cifrado en MarkeTOR.

Como ya se ha indicado, MarkeTOR cifra los mensajes que se enviaran a los usuarios mediante el uso de su clave pública. La clave pública que se usa en cada caso es aportada por los usuarios cuando crean su perfil de usuario. Los siguientes fragmentos de código indican cómo se usa dentro de la aplicación

```
clear_message = get_template(f'email/email_{rol.value}.txt').render(order_information)

if rol is UserTypeMail.seller:
    message = suborder.seller.gpgprofile.encrypt(clear_message)
else:
    message = order.user.gpgprofile.encrypt(clear_message)

msg = EmailMessage(subject, str(message, 'utf-8'), to=to, from_email=from_email)
msg.content_subtype = 'text'
msg.send()
```

Ilustración 60 - Fragmento del envío de correo cifrado

El cifrado se aplica directamente desde la cuenta de usuario, es decir en el perfil se almacena su clave pública y como comportamiento se añade la posibilidad de cifrado de cualquier texto.

Se utiliza la librería Python **gnupg**, librería que envuelve la funcionalidad aportada por GnuPG instalada en el sistema.

La funcionalidad principal se encuentra en la aplicación Django **django_pgp,** y unas dos funciones interesantes son la de *obtención y preparación del cliente, así como el cifrado*.

```
@contextlib.contextmanager
def client(import_keys=[]):
  tmp = tempfile.mkdtemp()
  try:
    g = gnupg.GPG(gnupghome=tmp, options=['--trust-model', 'always'])
    if import_keys:
      result = g.import_keys('\n'.join(import_keys))
      if 'IMPORT_OK 1' not in result.stderr:
        raise ValueError(result.stderr.split('\n')[0])
    yield g
  finally:
    shutil.rmtree(tmp)
```

Ilustración 61 - Creación del cliente GnuPG desde la aplicación MarkeTOR

El fragmento de código prepara una instancia de GnuPG, importa la calve pública del perfil del usuario y la marca como de confianza, después la devuelve para ser usada por la función de cifrado incorporándola en el contexto de la aplicación.

```
def encrypt(message, recipient_keys=[]):
  if not recipient_keys:
    raise ValueError('Need at least one public key to encrypt to')
  with client(import_keys=recipient_keys) as g:
    keyids = [key['keyid'] for key in g.list_keys()]
    result = g.encrypt(message, ','.join(keyids))
    if not result.ok:
      raise ValueError(result.stderr.split('\n')[0])
    return result.data
```

Ilustración 62 - Función de cifrado usando la clave pública y el cliente inyectado en el contexto de la aplicación

Esta es la función que se llama para cifrar un mensaje de texto, con la clave pública del usuario. Se llama desde el objeto que representa al perfil del usuario: **<user>,gpgprofile.encrypt(mensaje)**

4.2.10 Despliegue del Servicio Onion en TOR usando Whonix

La aplicación ya tiene toda la funcionalidad lista, es el momento de desplegarla a producción. Se procede a describir en este punto cómo se ha despliega en producción el servicio onion.

La arquitectura del sistema a grandes está formada por dos máquinas virtuales que corren en un host físico. Las máquinas virtuales están basadas en Whonix que es un sistema operativo GNU/Linux orientado a la privacidad y seguridad.

Consta de 2 máquinas virtuales, una hace de pasarela hacia TOR, es decir se encarga de la comunicación con la red TOR, hace que todo el tráfico que llegue a ella solo pueda salir mediante TOR. Esta primera máquina recibe el nombre de **Whonix-GateWay**.

La segunda máquina, llamada **Whonix-Workstation,** se conecta a la anterior y es donde reside el servicio onion, por lo que esta máquina queda aislada y solo pudiendo salir hacía la red TOR y además solo pasando por la máquina pasarela **Whonix-GateWay.**

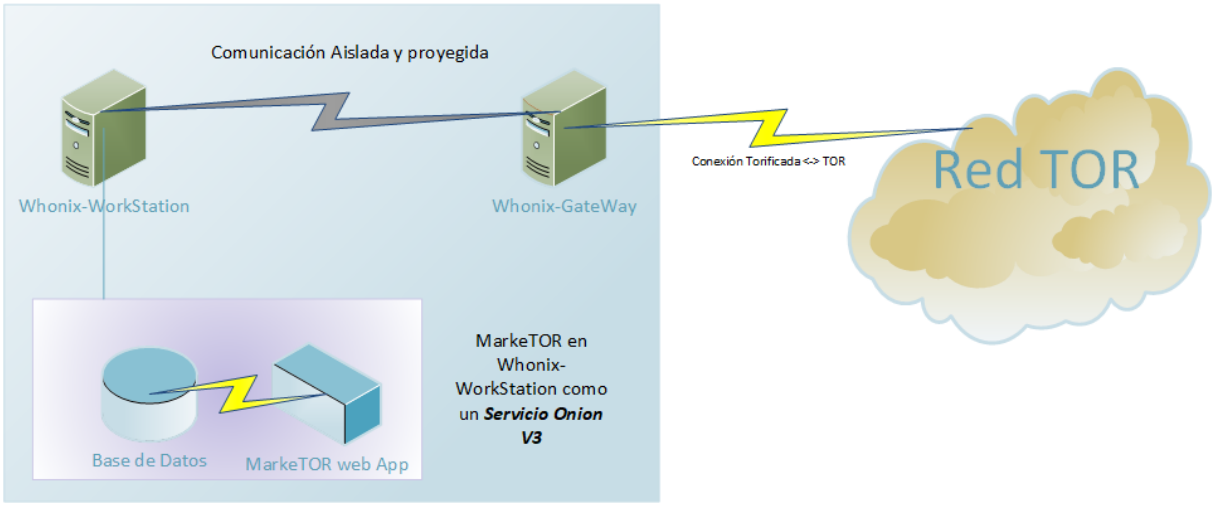

Ilustración 63 - Arquitectura de alto nivel de despliegue

4.2.11 Configurando Whonix-Gateway. Servicio Onion en TOR

Whonix-Gateway es la máquina virtual que se expone a TOR y que redirige todo el tráfico hacia esa red.

Es aquí donde se configura el acceso al servicio onion. Es la máquina que contiene el nombre del servicio y las claves pública y privada que se utilizan para generar ese nombre, que luego se sube a la base de datos distribuida de TOR, junto con la clave pública, para que los servicios onion sean encontrados en TOR.

En MarkeTOR, se han generado unas claves y un nombre de host, que permita tener las primeras letras de este reconocibles, haciendo que la dirección onion v3 empiece por '**mktor**'. El proceso de creación de direcciones onion personalizadas se abordará en el siguiente punto.

Configurando el servicio onion.

Se debe de configurar la información sobre los datos del servicio, es decir directorio donde se almacenan las claves y nombre de host, así como el número de puerto que se va a utilizar. También es importante indicar la versión de servicios onion que se va a utilizar. MarkeTOR utiliza la última versión, la 3, que incorpora ventajas de seguridad y privacidad.

El archivo de configuración que se debe de utilizar es:

```
/usr/local/etc/torrc.d/50 user.conf
```

Ilustración 64 - Archivo de configuración del servicio en Whonix-Gateway

Los valores para configurar en este archivo son los siguientes:

- `HiddenServiceDir:` Indica el directorio donde se almacena la configuración del servicio onion, es decir, donde se encuentran el nombre del host y la clave privada.

- `HiddenServicePort:` El Puerto virtual que se usará, así como la IP y el puerto de la máquina Whonix-Workstation donde corre el servicio.

- `HiddenServiceVersion:` Versión del servicio, solo puede ser 2 o 3. MarkeTOR utiliza la versión 3.

En el caso de MarkeTOR, el archivo `/usr/local/etc/torrc.d/50_user.conf` queda configurado de la siguiente forma:

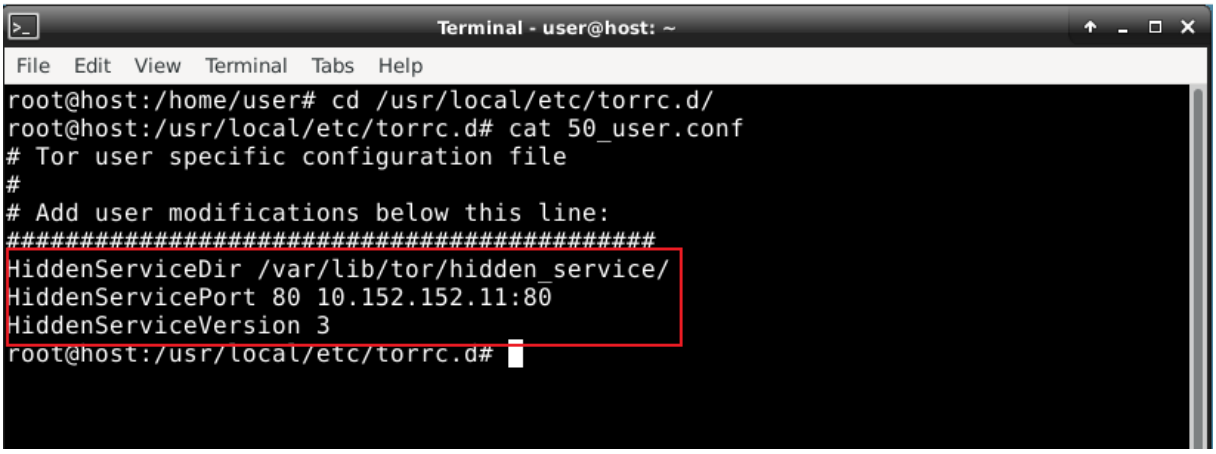

Ilustración 65 - Archivo de configuración 50_user_conf (Whonix-gateway) - Configuración servicio Onion

El directorio `/var/lib/tor/hidden_service` contiene los datos de configuración del servicio. El nombre del host, y las claves privadas y públicas, que en la versión 3 de los servios onion usan ed25519. Al configurar el servicio estos archivos se generan automáticamente, generando una dirección aleatoria. En MarkeTOR se han sustituido por una personalizada, cuyo procedimiento de generación explicaremos más adelante.

Es muy recomendable hacer una copia de seguridad de estos archivos, sobre todo de la clave privada. Si alguna vez se destruye la máquina, si no la clave privada, la dirección del servicio onion se perdería y se tendría que generar un nombre distinto con claves distintas.

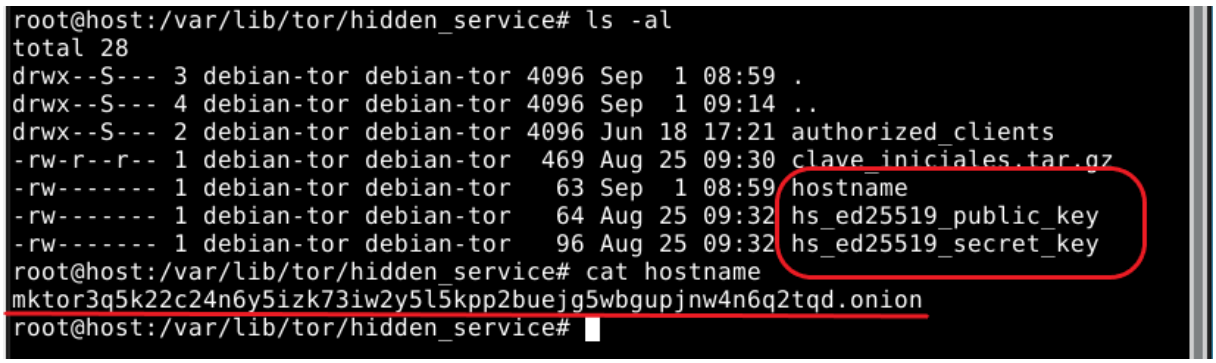

Ilustración 66 - Archivos de configuración: hostname y claves del servicio.

Esta sería la configuración por realizar en la máquina Gateway, aprovechando la instalación por defecto que hace Whonix de sus máquinas virtuales.

4.2.12 Creación de una dirección onion v3 personalizada.

Ya se ha visto, que al configurar en servicio onion v3, se generan unas claves ed25519 para firma digital y el nombre del host que se deriva de la clave pública generada. La versión 3, en el ámbito criptográfico, mejora el conjunto de algoritmos criptográficos usados, pasando de **SHA1/DH/RSA1024** a usar **SHA3/ed25519/curve25519**.

Aunque mantener las claves y nombre de host generados es totalmente válido, se generará una dirección personalizada, que permite a primera vista reconocer la dirección más fácilmente. Estas direcciones se conocen como ***TOR Onion vanity address.***

La dirección que se ha generado para el servicio onion MarkeTOR es:

mktor3q5k22c24n6y5izk73iw2y5l5kpp2buejg5wbgupjnw4n6q2tqd.onion

Ilustración 67 – TOR Onion v3 Vanity Address del servicio MarkeTOR.

Para ello hemos utilizado el proyecto **mkp224o** (cathugger, 2019) capaz de generar direcciones v3 personalizadas y permitir optimizaciones de compilación.

El proceso a grandes rasgos para generar una dirección onion v3 es este:

- Generación de claves con curve25519
- Obtener el hash SHA-3 de la clave publica generada
- Codificar en Base32 el inicio del hash

Las direcciones onion v3, tienen una longitud de 56 caracteres codificados en Base-32, por lo que se usan las letras minúsculas de la ‘a’ hasta la ‘z’ y los números del 2 al 7.

Para personalizar la dirección que tendrá el servicio onion, es necesario la clave privada relacionada con dicha dirección y las direcciones son hashes, se realiza mediante el uso de

técnicas de fuerza bruta para generarla. Cuantos más caracteres personalizados queramos, más capacidad de cómputo y tiempo será necesario.

Para MarkeTOR se ha optado por solamente personalizar los primeros 5 caracteres, de forma que en pocos segundos se pueden generar varias direcciones con el software mkp224o. estas direcciones tienen la forma:

mktorxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.onion

El hardware usado para realizar esta generación es el siguiente:

- **CPU**: Ryzen 7 1700 (8 núcleos – 16 hilos) – 3.7 GHz
- **RAM**: 16 GB
- **S.O.:** GNU/Linux OpenSUSE Tumbleweed (amd_64)

El proyecto mkp244o, está escrito en C y debemos compilarlo en la máquina cuyas características se han mostrado previamente. Es importante indicar que el proyecto ofrece algunas indicaciones de optimización para su compilación, que en función del hardware usado permitirá un rendimiento mucho mayor, que una compilación estándar.

Para poder compilar dicho programa usaremos el sistema operativo GNU/Linux OpenSUSE Tumbleewed, pero se puede utilizar cualquier sistema Linux, siempre que tengamos las herramientas de compilación instaladas y algunas dependencias específicas instaladas. En concreto nos referimos a la biblioteca **libsodium,** que debe estar instalada en su versión de desarrollo.

```
• Derivados Debian (deb):  libsodium-dev
• Derivados Red Hat (rpm): libsodium-devel
```

Ilustración 68 - Dependencia principal para la construcción de mkp224o

En el caso del hardware utilizado usaremos las optimizaciones de compilación**: --enable-amd64-64-24k --enable-intfilter -B**:

```
$ ./autogen.sh
$ ./configure --enable-amd64-64-24k --enable-intfilter -B
$ make
```

Ilustración 69 - Compilación de mkp244o.

La siguiente imagen muestra el resultado de la ejecución del programa para la generación de direcciones que empezaran por **mktor**.

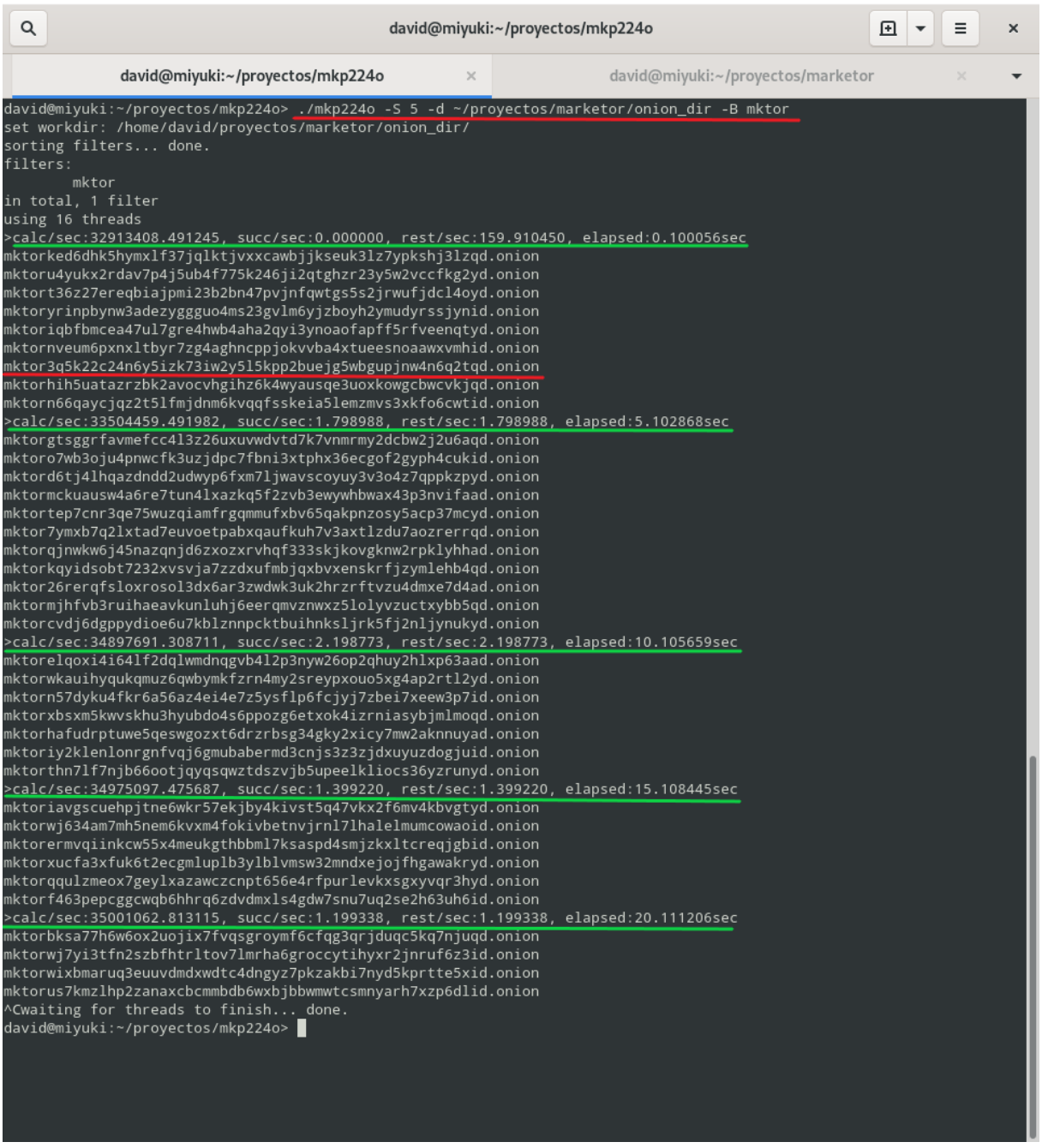

Ilustración 70 - Generación de dirección personalizada con mkp224o para MarkeTOR.

Se le pasan parámetros para indicar que use todos los núcleos del sistema y muestre información cada 5 segundos. También se le pasa el directorio donde queremos que se le guarden las claves generadas y nombres de host.

La imagen muestra que usando las optimizaciones de compilación en el hardware indicado y usan GCC 9, los tiempos de generación de nombres con 5 caracteres personalizados en muy baja.

Los tiempos de generación se amplían exponencialmente según se generan nombres personalizados con más caracteres elegidos. Una prueba de generación y extrapolado de resultados de generación realizado por Jamie Scaife (Scaife, 2018), usando un clúster de 5 nodos Raspberry Pi 2B arroja los siguientes resultados:

```
Vanity Characters : Approximate Generation Time
1  : <1 second
2  : <1 second
3  : 1 second
4  : 30 seconds
5  : 16 minutes
6  : 8.5 hours
7  : 11.5 days
8  : 1 year
9  : 32 years
10 : 1,024 years
11 : 32,768 years
12 : 1 million years
```

Ilustración 71 - Tiempos de cálculo de generación de dircciones onion - cluster 5 Raspberry Pi

Para los objetivos que se desean conseguir en este proyecto, se considera 5 caracteres personalizados como suficientes.

4.2.13 Configurando Whonix-Workstation y el Servicio Onion MarkeTOR

El servicio onion, es decir la aplicación MarkeTOR, se ejecuta en **Whonix-Workstation**. La aplicación para su despliegue utiliza dos componentes fundamentales:

- **Gunicorn**: Servidor WSGI HTTP de Python, expone la ampliación Django.
- **Nginx**. Servidor web que hace de proxy a la ampliación Django y sirve los recursos estáticos de la misma.

Los componentes de despliegue de la aplicación en la máquina Whonix-Workstation se pueden ver en la siguiente imagen:

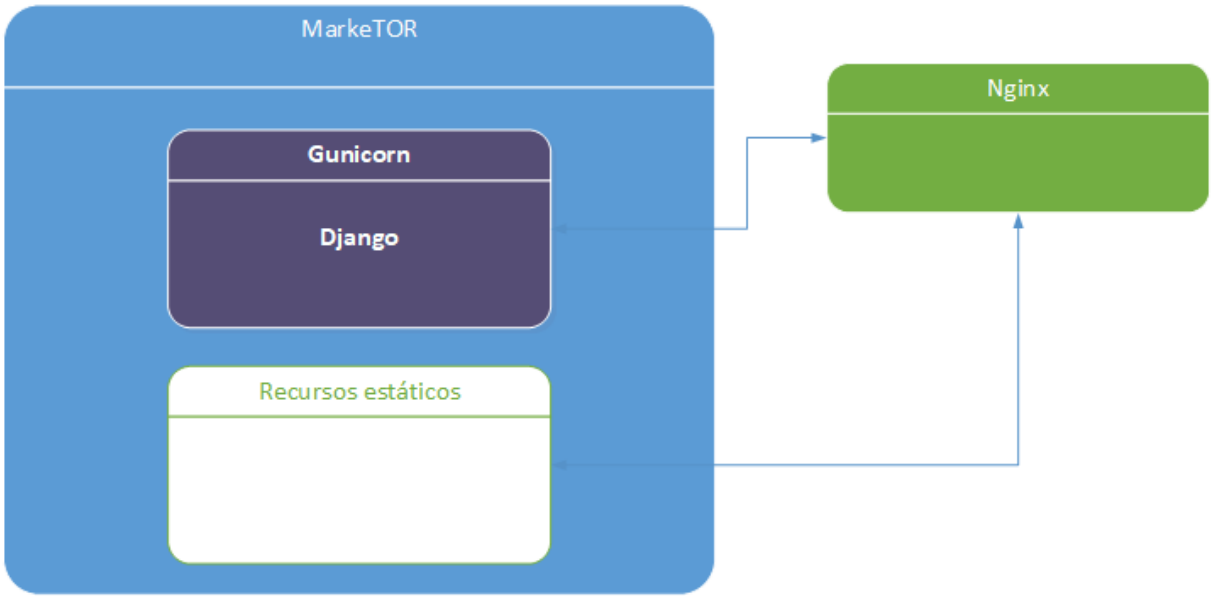

Ilustración 72 - Arquitectura de despliegue - MarkeTOR

Se van a describir a continuación cada uno de los componentes y su configuración.

Gunicorn[37].

Es un servidor de aplicaciones Python basado en WSGI. Su arquitectura se base en el modelo '*pre-fork*', lo que significa que tiene un **proceso central o maestro**, que gestiona un conjunto de procesos '**worker'** que son los que reciben las peticiones de los clientes. El proceso maestro escucha las peticiones y las envía a los procesos '**worker',** de forma asíncrona**,** según sea necesario, eliminando o creando nuevos si fuera necesario.

Nginx[38].

Es un servidor web y proxy inverso ligero de alto rendimiento. Es software libre y código abierto.

Se podría haber optado por otras alternativas como pueden ser Apache Web Server, pero se decidió utilizar **nginx** debido a su alto rendimiento, poco uso de memoria y la facilidad que ofrece para implementar características de seguridad. Otro de los motivos de usar **nginx** en vez de otras alternativas como Apache Web server, es por la recomendación que se hace en Whonix, específicamente sobre este tema. Se considera que de cara a la privacidad es más seguro utilizar **nginx**, ofreciendo menos posibilidades de dejar escapar información no deseada y ofreciendo una superficie de ataque menor.

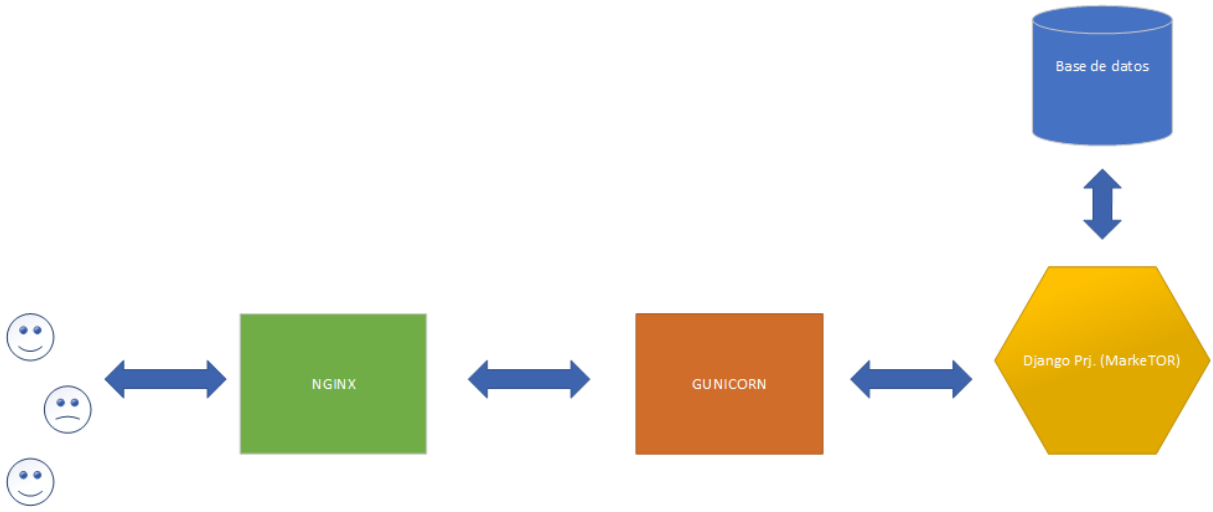

Ilustración 73 - Arquitectura básica: nginx - Gunicorn - Django

[37] https://gunicorn.org/
[38] https://www.nginx.com/

La máquina **Whonix-Workstation**, es la que ejecutará la siguiente combinación de software:

- Entorno virtual Python con pipenv
- Gunicorn
- Nginx

Entorno Virtual Python.

Se usa *pipenv como* sistema para gestionar el entorno virtual de ejecución de la aplicación y como gestor de dependencias.

Pipenv se instala en el sistema usando **pip**, que es el sistema de gestión de paquetes estándar de Python. Su instalación se ejecutará en la máquina **Whonix-Workstation.**

```
$ pip install pipenv
```

Una vez instalado ya se puede crear o activar el entorno virtual, instalar las dependencias del proyecto. Para ello se deben de ejecutar de forma seguidas los siguientes comandos en el directorio donde reside la aplicación MarkeTOR, en una consola de línea de comandos:

```
$ pipenv --python 3 # Crea un entorno virtual con Python 3 (SOLO se ejecutará la primera vez)

$ pipenv shell     # Activa el entorno virtual
$ pipenv install   # instala dependencias (usar update para actualizar las versiones de los paquetes.)
```

Tanto la creación del entorno virtual con Python 3, como la instalación o actualización de los paquetes Python requeridos, solo es necesario realizarlo la primera vez. La actualización de paquetes con *pipenv update* se puede realizar cuando se detecten vulnerabilidades (usando *pipenv check*) en los paquetes existentes y así lanzar una actualización de estos.

Una vez ejecutado el comando de activación del entorno virtual, el sistema ya está listo para lanzar el servidor de aplicaciones *Gunicorn*, que lanzará la aplicación Django y la servirá en el puerto 8000 de la máquina **Whonix-WorkStation.**

Gunicorn.

El servidor que Django trae para desarrollo no está optimizado para trabajar en producción. Durante el desarrollo se ha usado dadas sus capacidades de depuración, así como su facilidad de implementación, ya que viene preparado en cualquier proyecto Django para ser usado inmediatamente.

Gunicorn por otro lado es un servidor de producción más rápido y eficiente. Cuando se expone en producción en la red TOR la aplicación MarkeTOR, usamos este servidor en vez del de producción.

Gunicorn está especificado en el archivo **Pipfile** de la aplicación, por lo que al instalar el entorno virtual e instalar las dependencias del proyecto, ya estará instalado. Las dependencias del proyecto se mostraron en el punto **4.2.4 Artefactos de desarrollo: Vagrant y Pipenv**.

Lanzar la aplicación MarkeTOR con Gunicorn como servidor de aplicaciones, es muy sencillo. Con el entorno Python activado, se debe de invocar Gunicorn indicando la dirección y puerto de enlace, así como la aplicación WSGI que se desea ejecutar, en nuestro caso MarkeTOR. Django expone el objeto 'application' como un objeto ejecutable a nivel de modulo y en MarkeTOR corresponde al archivo **wsgi.py** de la carpeta **mercator** del proyecto, lugar donde se encuentra la configuración del proyecto (settings.py) y las urls principales (urls.py), por lo que el comando a usar será:

```
$ gunicorn --bind 0.0.0.0:8000 mercator.wsgi
```

La siguiente imagen muestra cómo se lanza la aplicación MarkeTOR con Gunicorn, usando el entorno virtual Python.

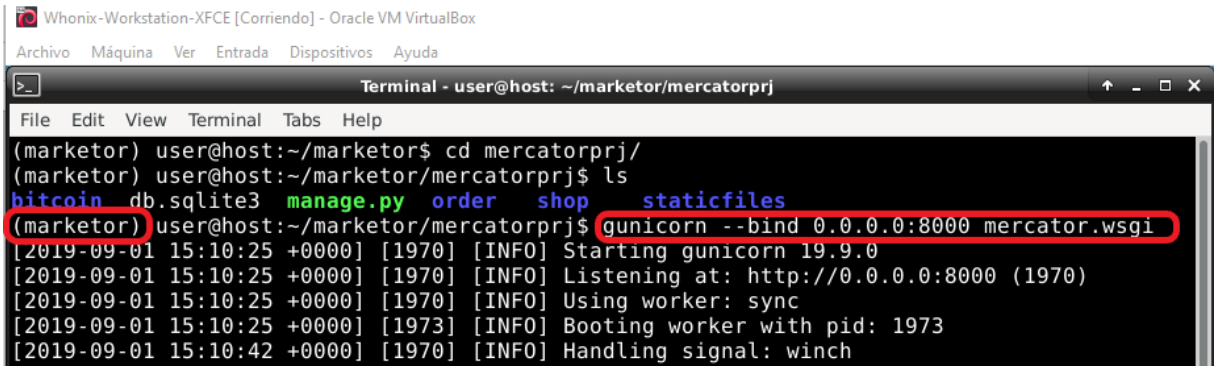

Ilustración 74 - Arrancando MarkeTOR con Gunicorn

Nginx.

La última pieza del puzle es **nginx**, es el servidor web que servirá los contenidos estáticos, imágenes, JavaScript, CSS etcétera y que además funciona como proxy inverso hacia el servidor de aplicaciones WSGI **Gunicorn**, pasando las peticiones que recibe este último para su procesado.

La instalación de **nginx** en **Whonix-Workstation** es muy sencilla, al ser un sistema basado en Debian se usa el gestor de paquetes APT. Nginx se encuentra en los paquetes de la distribución, por lo que su instalación se limita a ejecutar:

```
$ sudo apt install nginx
```

Una vez instalado, se debe de configurar para que se integre con Gunicorn y la aplicación Django MarkeTOR.

Configurando Nginx en MarkeTOR.

Whonix-Workstation, utiliza un firewall por defecto que bloquea todas las conexiones, por lo que sebe de configurar para permitir recibir las conexiones que le llegan e la red TOR, a través de **Whonix-Gateway**. Se debe abrir el puerto 80 en el firewall y reiniciar el mismo.

La configuración del firewall de **Whonix-Workstation,** se encuentra en el archivo `/etc/whonix_firewall.d/50_user.conf` y añadir el puerto 80 a la variable `EXTERNAL_OPEN_PORTS`.

Quedando dicho archivo con el siguiente contenido:

```
EXTERNAL_OPEN_PORTS+=" 80 "
```

Después se debe reiniciar el firewall ejecutando:

```
$ sudo whonix_firewall
```

Whonix-Workstation, utiliza al igual que Debian 10, ***Systemd*** como sistema para gestionar los servicios y otras configuraciones del sistema. Es recomendable arrancar e instalar en el arranque del sistema servicio **ngnix**. Ejecutaremos como administradores el siguiente comando que permite arrancar el servicio **ngnix** y hacer que arranque automáticamente en los inicios del sistema.

```
$ sudo systemctl start nginx && sudo systemctl enable nginx
```

En la siguiente imagen se observar el estado del servicio nginx, corriendo con normalidad.

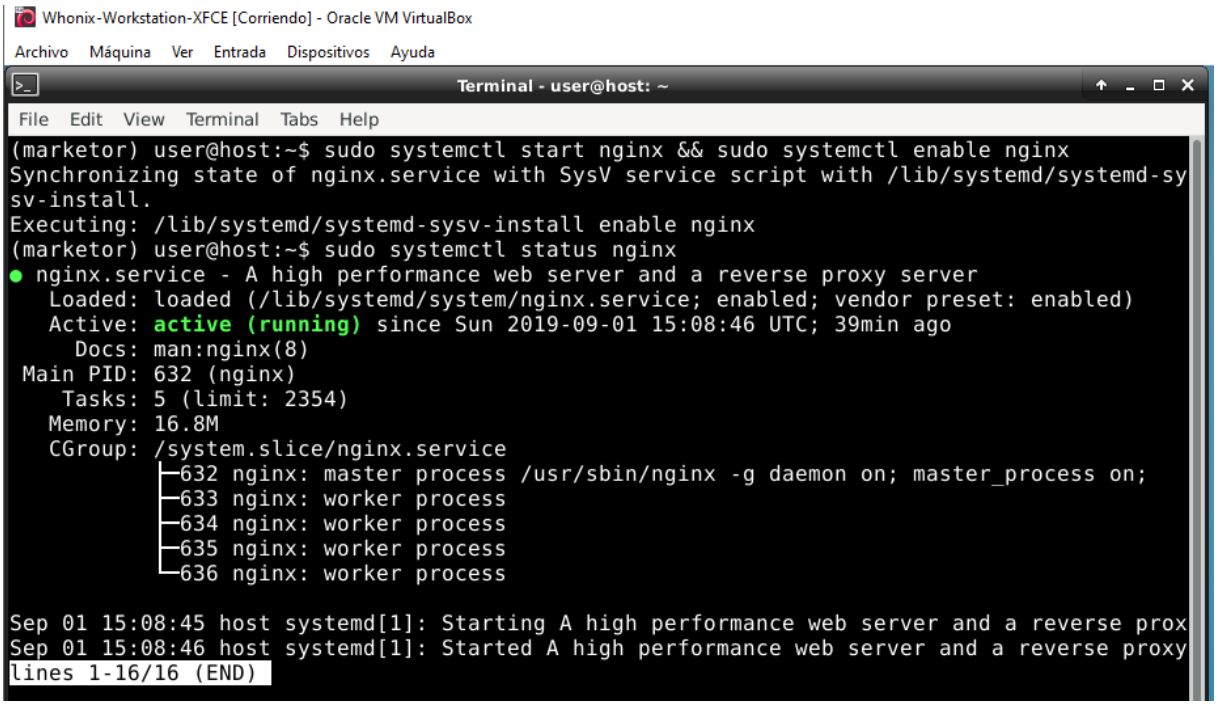

Ilustración 75 - Estado nginx, corriendo en producción.

Para que todo funcione con corrección, solamente falta el último paso que es la configuración de sitio MarkeTOR en nginx.

Nginx, contiene todos sus sitios disponibles en el directorio `/etc/nginx/sites-available/` por lo que se deberá crear ahí el archivo de configuración de la aplicación. Lo llamaremos **marketor**.

Este archivo debe de tener toda la configuración necesaria para poder escuchar en IP correcta, en este caso escuchamos en localhost, ya que **Whonix-Gateway** redirige las

peticiones realmente aquí. Otra información necesaria es el puerto sobre el que va a escuchar el servicio. Ya se ha visto que usaremos el puerto 80, tal y como se describió en la configuración del firewall.

Nginx recibirá las peticiones que le envía **Whonix-Gateway** procedentes de la red TOR. Si son contenidos estáticos los servirá el directamente, si es código de la aplicación Merkator, direcciones que gestiona Django, actuará como proxy inverso y le pasará a **Gunicorn** la petición.

Para lograr este comportamiento se debe de indicar en el archivo de configuración de nginx `/etc/nginx/sites-available/marketor`. Se puede ver dicho archivo en la imagen.

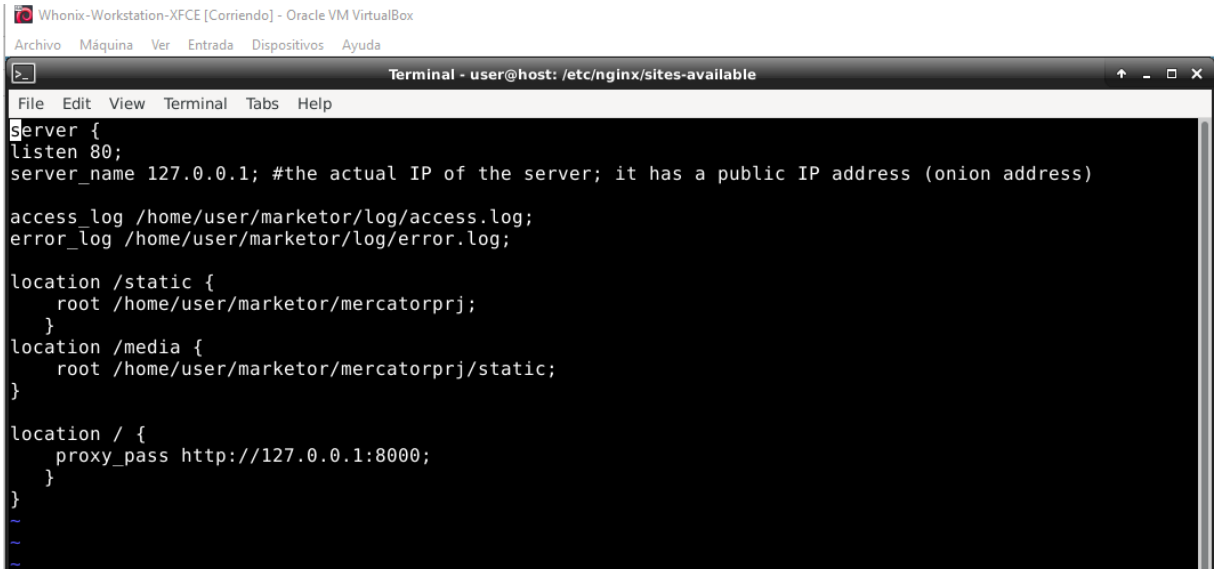

Ilustración 76 - Archivo de configuración nginx usado en producción

Son importante aquí las entradas *location* que indican como tratar los recurso bajo esas direcciones. Los URLS `/static` y `/media`, albergan contenido estático, en **static** están las CSS, imágenes fijas del sitio MarkeTOR, fuentes web y JavaScript del sitio. En **media** se almacenan las imágenes de los productos, que se suben cuando estos se dan de alta.

La tercera entrada de *location* se refiere a /, por lo que el resto de las peticiones toman este camino, que como se puede apreciar usa la directiva `proxy_pass`, es decir se pasan las peticiones a **Gunicorn** y este a la **aplicación Django** para ser procesadas.

Solamente queda decirle a nginx que use este archivo de configuración, para ello se realiza un enlace simbólico a este archivo en `/etc/nginx/sites-enabled/`.

Se pueden ejecutar las siguientes órdenes para conseguir ese efecto:

```
$ cd /etc/nginx/sites-enabled/
$ sudo ln -s /etc/nginx/sites-available/marketor marketor
```

Con este último paso la aplicación está desplegada en producción y funcionando en la red TOR, como un servicio onion versión 3, accesible mediante TOR Browser.

4.2.14 Evaluación de MarkeTOR

En la metodología utilizada durante el desarrollo de esta aplicación, se indicaba que es un trabajo cíclico, en que se iban construyendo artefactos, se testeaban y probaban para comprobar que cumplían los requisitos, funcionales y de rendimiento.

Con el fin de poder probar las funcionalidades se crearon perfiles de usuario para probar la aplicación, con la intención que fuesen usados por algunas personas y ofrecer un '*feedback*' del uso de esta, aportando ideas, encontrando errores y ofreciendo una opinión independiente.

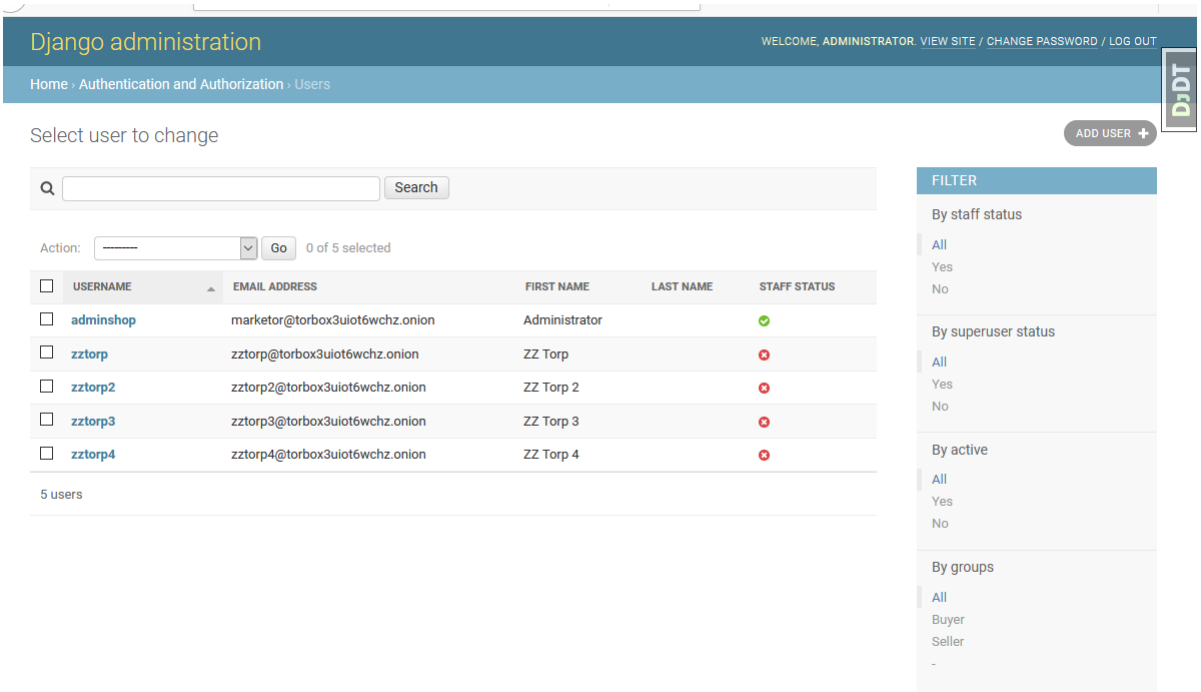

Ilustración 77 - Listado de usuarios utilizados para probar MarkeTOR

Se crearon cuatro usuarios, con distintos roles y sus perfiles. Los perfiles implican la creación de cuentas de correo en la red TOR, la creación de claves GnuPG para poder cifrar y descifrar el correo que se envía.

Se busco la ayuda de algunas personas, algunas con perfil más técnico y otras sin ese tipo de perfil, se les enseño brevemente a usar TOR Browser y como acceder al sitio. En cuanto a los usuarios no técnicos, no fueron capaces sin ayuda de poder descifrar los correos electrónicos que les llegaban de MarkeTOR. El resto de las funcionalidades, son las esperadas de un mercado en Internet por lo que no hubo problemas.

Durante estas pruebas se detectaron problemas y bugs, que se fueron solventando y haciendo que la aplicación funcionara de acuerdo con los requisitos necesarios de forma robusta.

Un caso concreto, fue la detección de problemas con las API de acceso a la blockchain de Bitcoin tanto en tiempo de respuesta como en disponibilidad.

Uno de los usuarios, detecto que a partir de un momento concreto dejo de funcionar la comprobación de estados de las transacciones, pero sí que podía usar el sitio web de constructor del API para comprobar su estado. Investigando el caso, se comprobó que devolvía errores HTTP 503, de forma indefinida. Tras investigar la situación no se pudo obtener información sobre si este API volvería a estar operativo en algún momento, por lo que se decidió implementar un mecanismo alternativo, que usa otro API para las funcionalidades que habían quedado sin funcionalidad.

El uso de la aplicación por este conjunto de usuario también ha aportado algo muy importante, ideas sobre las líneas de crecimiento que en un futuro podría tomar el proyecto para crecer. En el apartado de conclusiones y trabajos futuros se pueden leer algunas de estas aportaciones.

Esta fase ha sido muy importante de cara a la mejora de la solución propuesta, así como a ver perspectivas distintas de cómo debería de funcionar este tipo de servicios en la red TOR.

4.2.15 Captura y análisis del tráfico de MarkeTOR

El tráfico en la red TOR va completamente cifrado, como hemos visto cada nodo por el que pasa se aplica una capa de cifrado. El objetivo es garantizar el anonimato de sus usuarios y de los servicios alojados en la red TOR. Es, por lo tanto, es muy difícil analizar el tráfico o seguir el rastro de personas en la red TOR.

Para la captura de tráfico en la arquitectura montada en el despliegue de, MarkeTOR se ha definido un procedimiento simple, para observar si se puede obtener algo en claro.

La idea es obtener tráfico en tres puntos concretos de esta arquitectura y después analizarlo con alguna herramienta como son **tcpdump**. **Wireshark y NetworkMiner Free Edition**.

Los puntos elegidos para la captura de tráfico son:

1. Máquina usando TOR Browser para acceder a MarkeTOR
2. Whonix-Workstation
3. Whonix-Gateway

TOR configura un proxy SOCKS que escucha en el puerto TCP 9150 en la dirección 127.0.0.1 o *loopback*. Este proxy SOCKS se usa por el TOR Browser para cifrar el tráfico y enviarlo a la red TOR. La conexión entre TOR Browser y el proxy SOCKS es una conexión que no está cifrada, es un punto donde se podría obtener todo el tráfico que el cliente envía y recibe hacia y desde la red TOR.

Otros puntos interesantes son en las dos máquinas usadas para la publicación del servicio onion, es decir el otro extremo de la conexión, las dos máquinas Whonix.

Se pretende por lo tanta realizar las mismas tareas 3 veces y capturar y analizar el tráfico en esos tres puntos. Lo que se hará es:

- Crear un usuario nuevo en MarkeTOR
- Acceder con un usuario
- Navegar y realizar una compra, comprobando el estado de la transacción

- Comprobar el correo enviado en TorBox

Usaremos ***tcpdump*** para capturar el tráfico, ya que las tres máquinas usadas son GNU/Linux y es muy cómodo y simple de usar. Como alternativa se puede usar Wireshark, que tiene una interfaz gráfica potente y además permite visualizar las tramas obtenidas.

Máquina Cliente usando TOR Browser.

En una máquina ejecutando Fedora 30 Workstation, usamos TOR browser y realizamos las tareas indicadas, es decir, la creación de un usuario, inicio de sesión, navegar por MarkeTOR y comprar algún producto. También comprobaremos el correo enviado por el mercado y la transacción.

Se minimizará el uso de la red para interferir lo menos posible. Y se lanza el comando

```
$ sudo tcpdump -i lo tcp -w captura_cliente_tor_browser.pcap
```

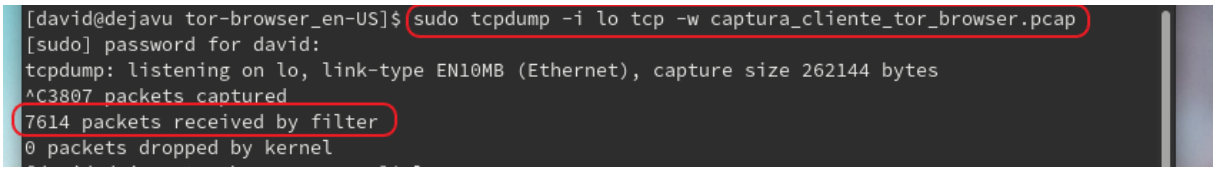

Ilustración 78 - captura de tráfico con tcpdump

Se puede ver en Tanto en el código como en la imagen que la captura se hace en la dirección de lookback (127.0.0.1) de tráfico TCP.

Una vez puesto a capturar paquetes de red, realizamos las tareas indicadas previamente, se captura el tráfico y cargamos el archivo PCAP en Wireshark y en NetworkMiner.

Abrimos el fichero de captura `captura_cliente_tor_browser.pcap`, con NetworkMiner y Wireshark.
En el caso de **Wireshark**, filtramos con la expresión **tcp.port==9150**, para filtrar el tráfico que va por SOCKS hacia ese puerto, antes de ser cifrado por TOR.

En las trazas obtenidas se puede ver todo el tráfico generado, de el podemos obtener que contenidos se han visitado, al obtener los GET de las peticiones. En los POST accedemos a las contraseñas y nombres de usuario, si buscamos los accesos a las páginas de inicio de sesión. Las siguientes imágenes muestran la captura de las contraseñas tanto del usuario creado como del inicio de sesión de otro usuario, usando **Wireshark**.

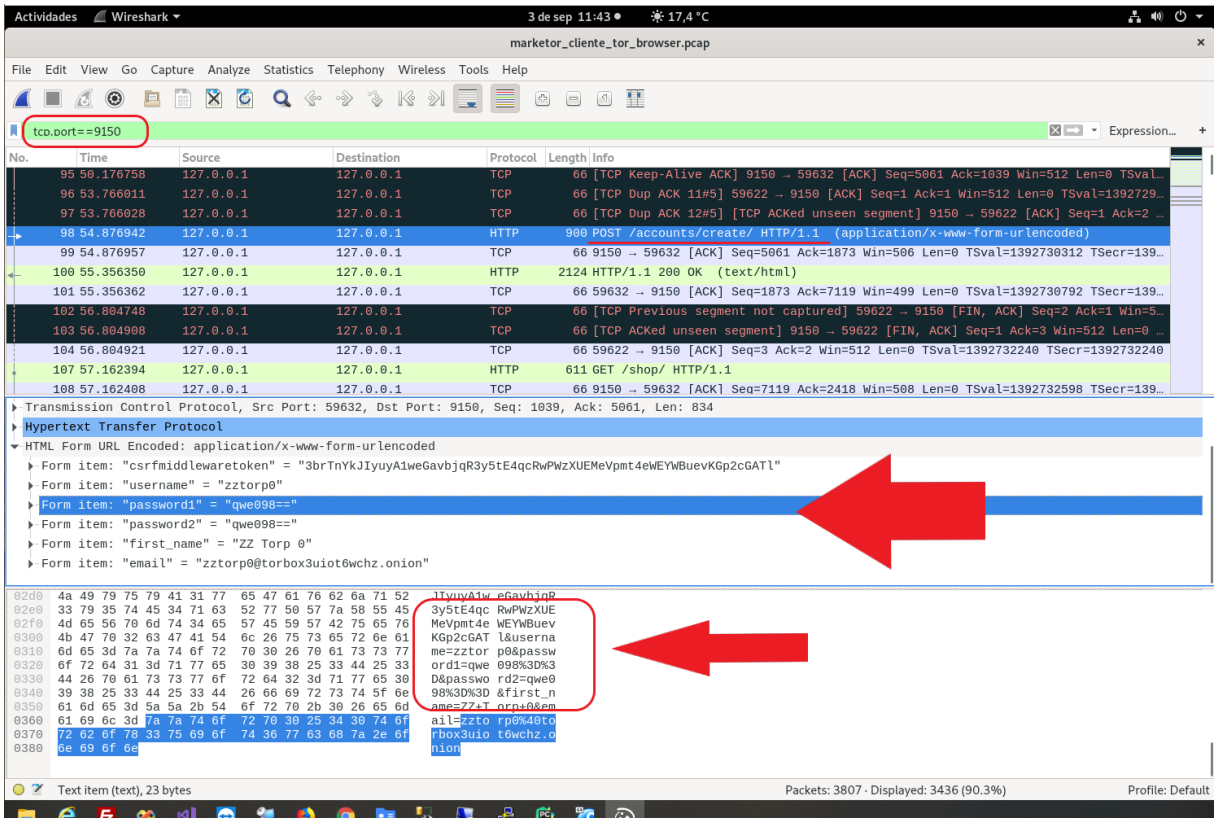

Ilustración 79 - Captura creación nuevo usuario

En la captura se puede apreciar como en el POST de /accounts/create tenemos la información del formulario, donde podemos ver que se ha obtenido completamente.

- Nombre de usuario creado: **zztorp0**
- Contraseña: **qwe098==**
- Token CSRF
- Correo del usuario
- Nombre en el portal

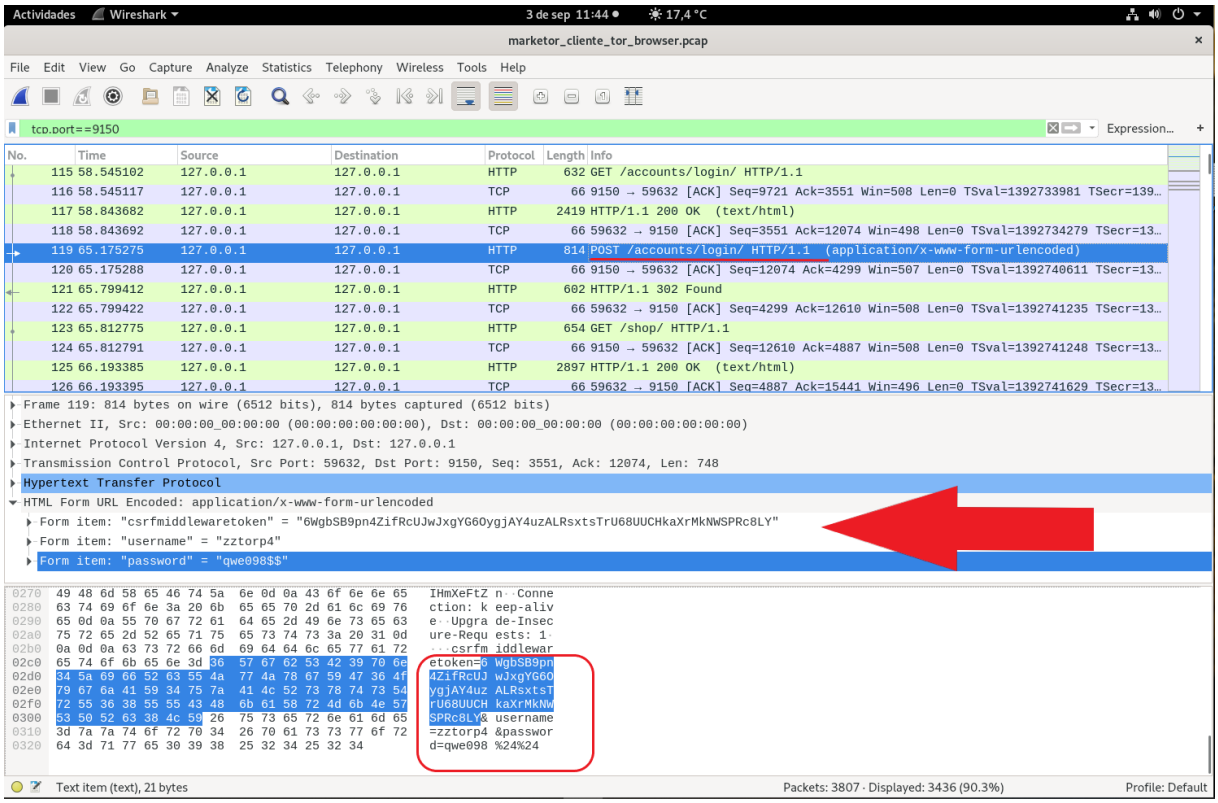

Ilustración 80 - Captura del inicio de sesión del usuario zztorp4

También se aprecia cómo se ha capturado también el inicio de sesión de otro usuario, **zztorp4**, con toda la información necesaria para suplantarle.

De la misma forma se puede obtener todas las URLs que ha visitado en el sitio y así observar que le interesa y que ha comprado y la dirección de envío.

Usando **NetworkMiner**, obtenemos también información muy interesante, como pueden ser las credenciales del correo electrónico en TorBox del usuario, cookies de sesión e imágenes de productos comprados y vistos. Las siguientes imágenes, muestran la información obtenida en la captura de paquetes.

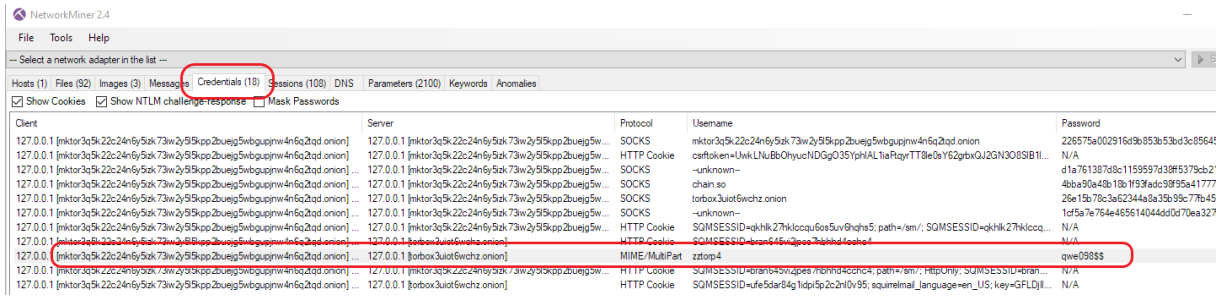

Ilustración 81 - Captura del inicio de sesión del correo en TorBox

NetworkMiner, también captura imágenes, en este caso el producto comprado por el usuario “zzTorp4”.

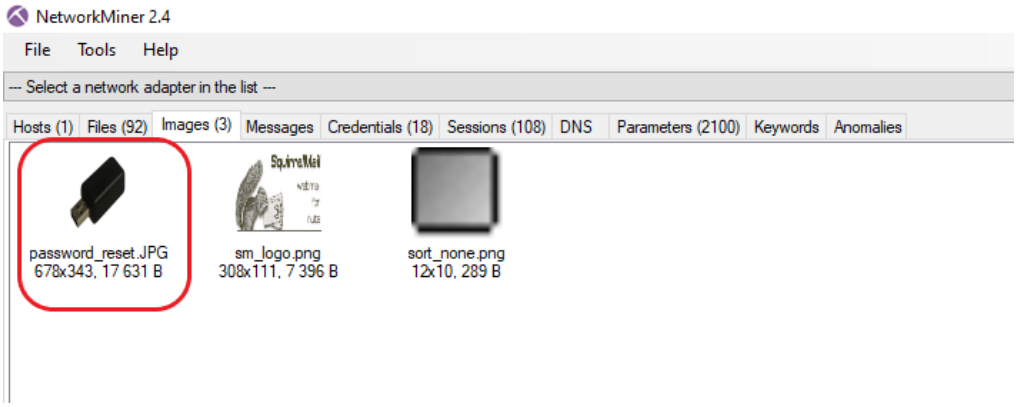

Ilustración 82 - Imágenes capturadas con NetworkMiner

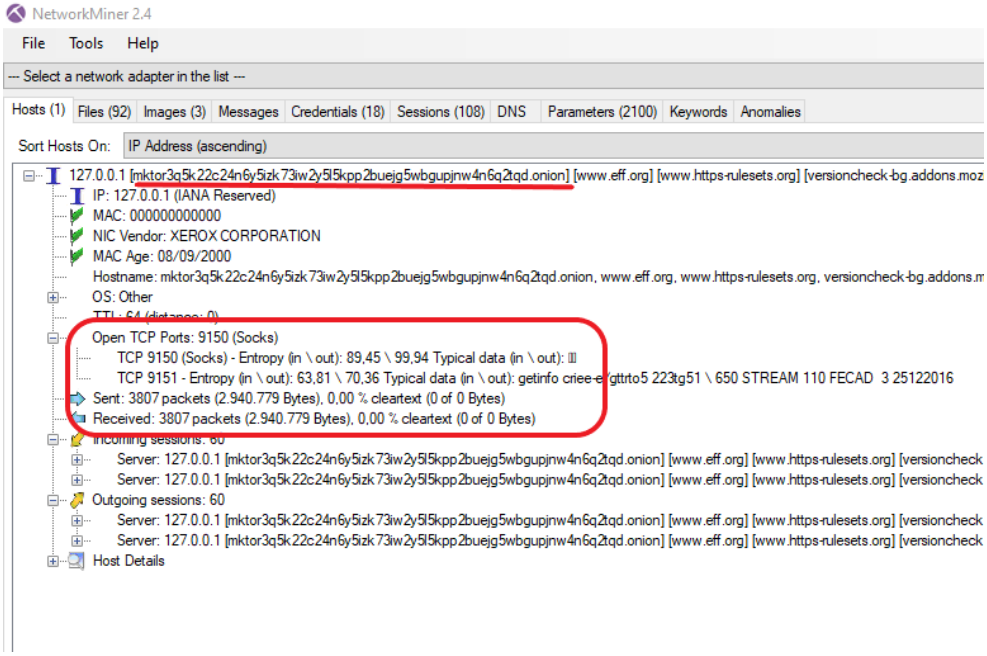

Ilustración 83 - Puertos usados en conexión a TOR por TOR Browser

La información obtenida justo en el cliente, como es lógico es accesible escuchando el contenido que pasa por el puerto TCP 9150. Ese contenido no se ha cifrado todavía y por lo tanto se puede capturar completamente. Es caso de un atacante, para obtener esta información debería tener control como root o administrador de la máquina que ejecuta el cliente TOR Browser. Esto podrá ser posible si estuviera infectado con algún tipo de malware

orientado a la captura de tráfico del equipo o si hubiera alguna vulnerabilidad en TOR browser que permitiera obtener esta información.

Máquina Whonix-Workstation – Servicio Onion.

Otro punto interesante, en el que se puede acceder a información, es en el servidor final, donde reside el servicio onion, la aplicación **MarkeTOR.**

Lógicamente si se captura tráfico en dicho host, en **Whonix-Workstation**, es porque está comprometido o tenemos las credenciales de root o administrador, por lo tendríamos acceso completo a la información enviada y recibida y el servicio al completo. Por lo tanto, es un punto crítico que proteger desde el punto de vista de la seguridad.

En la captura de tráfico con **tcpdump**, se ha obtenido prácticamente toda la información intercambiada entre el servicio onion en la máquina Whonix-Workstation y la máquina Whonix-Gateway, ya que es el único tráfico existente en este punto.

Como ejemplo se muestran información de la captura, donde se han obtenido, credenciales, mensajes de correo, imágenes alojadas en el servidor etc.

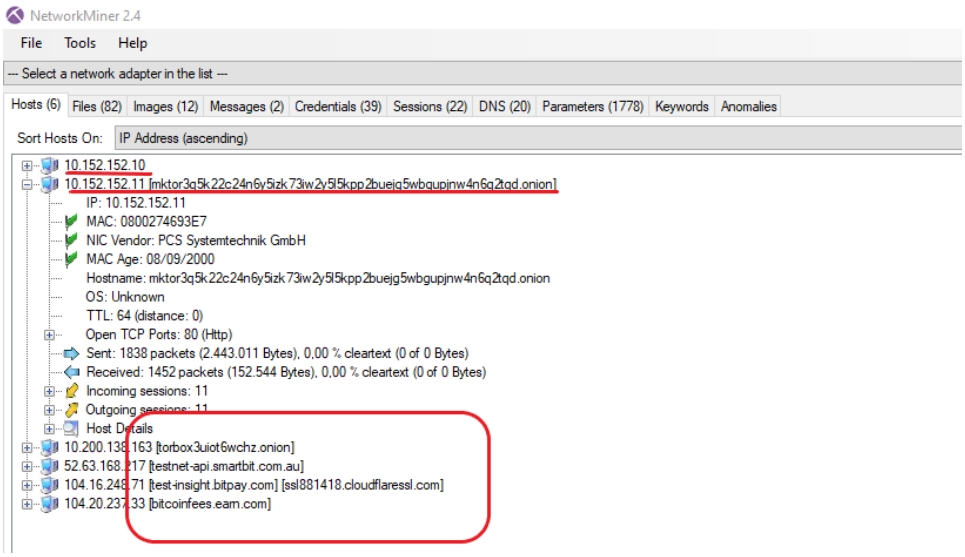

Ilustración 84 - Información capturada entre Whonix-Workstation y Whonix-Gateway. Y servicios externos accedidos. (APIs)

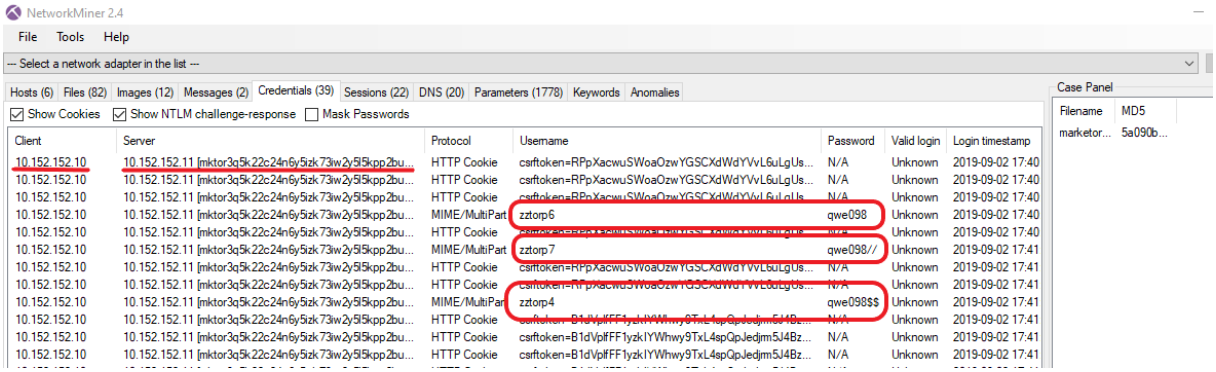

Ilustración 86 - Credenciales capturadas en Whonix-Workstation

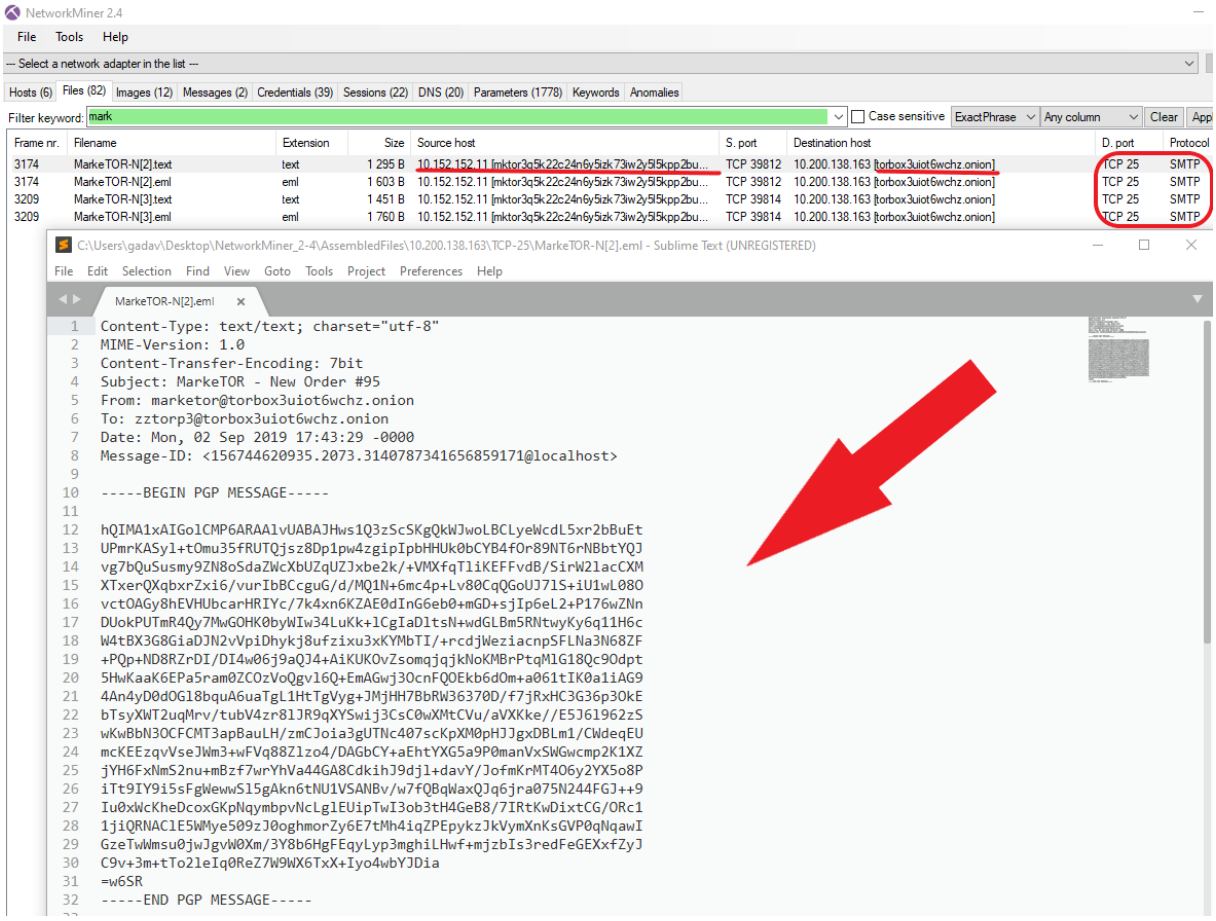

Ilustración 85 - Acceso a los correos intercambiados en MarkeTOR

En esta última imagen se puede observar que se ha capturado, los correos electrónicos enviados entre el mercado y sus usuarios. Al ir cifrados estos no se puede obtener el texto en claro. Además, para poder descifrarlos se necesitan las claves privadas de los intermediarios, que no residen en este servidor ni en el servicio, sino que solamente están en posesión de los usuarios intervinientes. Se constata la importancia, de cara a la privacidad, del uso del cifrado.

Como ya se ha indicado, si podemos capturar paquetes en la máquina donde reside el servicio onion, ya está descifrado y se puede obtener en claro, pero además tendríamos control absoluto de dicho servicio.

Máquina Whonix-Workstation – Servicio Onion.

El tercer punto donde se ha capturado tráfico ha sido en la máquina **Whonix-Gateway**. Este punto en muy interesante ya que tiene un comportamiento muy similar al de un nodo intermedio de la red TOR.

Se ha capturado el tráfico, de la misma manera que en los casos anteriores, es decir teniendo las credenciales de *root* o *administrador*, y realizando las mismas operaciones, es decir usando MarkeTOR para comprar algo, crear un usuario y consultar correo en TorBox.

En este caso, el tráfico va cifrado completamente mediante SSL/TLS y solo intervienen dos hosts, por lo que no es posible obtener ninguna información relevante. Este es el caso más parecido a la obtención se tráfico en un nodo de la red TOR. Va cifrada y los extremos de la conexión son los nodos anterior y posterior simplemente.

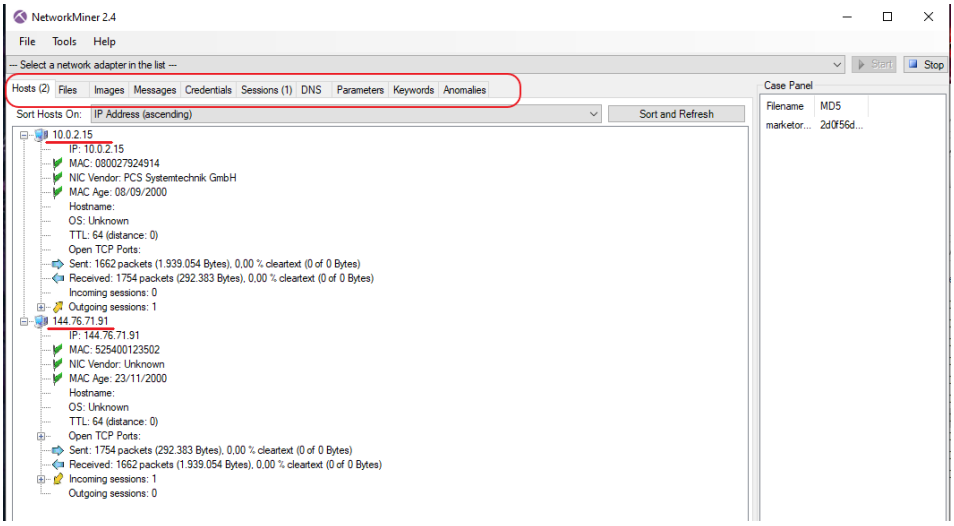

Ilustración 87 - Captura de tráfico en Whonix-Gateway – 1 – Hosts intervinientes

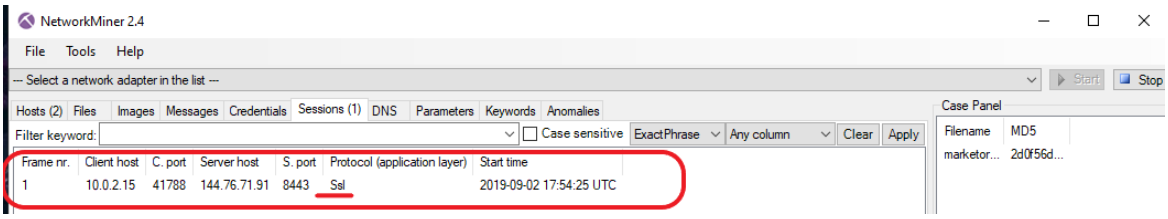

Ilustración 88 - Captura de tráfico en Whonix-Gateway - 2 - Tráfico cifrado

5. Conclusiones y trabajos futuros

5.1. Conclusiones

La red TOR y los servicios onion, antes llamados servicios ocultos, fueron creados con un propósito militar, sin embargo, se han convertido en un sistema que permite el anonimato de las comunicaciones para millones de personas en todo el mundo. Es cierto que aprovechando las características de privacidad y anonimato que ofrece esta red y estos servicios, se usa en muchos casos para actividades delictivas, como son algunos de los mercados negros existentes en TOR, que comercian con productos y servicios ilícitos, ya sea porque son ilegales, dañinos o incluso robados. La llegada de las criptomonedas ha sido otro factor determinante en el crecimiento de estos mercados ofreciendo anonimato y métodos de pago difíciles de seguir o controlar. Esto es un problema para las autoridades, fuerzas y cuerpos de seguridad de los estados cuando deben combatir los delitos que se cometen amparándose en esta red. Sin embargo, se conocen operaciones exitosas contra múltiples ciber mercados negros en la red TOR en los últimos años, pero son costosas y complicadas de llevar a cabo.

Es cierto que estas características también se usan para ofrecer servicios lícitos, que protegen el anonimato de usuarios y activistas que denuncian situaciones y eventos censurados y perseguidos en ciertos lugares del mundo. Esta red y sus servicios pueden ofrecerles la protección necesaria para expresarse libremente, comunicar y compartir contenido censurado o perseguido por regímenes autoritarios.

No se puede, por lo tanto, como en casi todos los avances tecnológicos, declararlos como algo positivo o negativo, sino que al uso concreto que se le dé.

La construcción de este mercado se ha realizado desde el punto de vista de la privacidad, tanto en los pagos como en las relaciones entre los usuarios y operadores del mismo. Se pretendía acercarse lo más posible a una situación real de este tipo de mercados.

En el análisis del tráfico se ha podido comprobar que durante su recorrido por la red TOR se encuentra bien protegido, pero los puntos más vulnerables son los extremos de la conexión.

En origen, si se atacan las vulnerabilidades del navegador TOR Browser o se logra tomar el control de la máquina del usuario que utiliza estos servicios, se podrá “*desanonimizar*” al

mismo. Lo mismo sucede en el otro extremo, en los servicios onion, si no se aplica la seguridad correctamente. Se podrá acceder y tomar el control del mismo, si no se tienen en cuenta las posibles vulnerabilidades existentes en los componentes que lo forman y si no se piensa en la seguridad y privacidad durante todas las fases de su construcción.

Las fuerzas de seguridad pueden usar un mercado de este tipo como una especie de '*honeypot*' donde estudiar qué tipos de productos se venden y compran en estos mercados, obtener información de los compradores y vendedores con el fin de desenmascararlos.

Por otro lado, un mercado de este tipo podría ofrecer productos orientados al conocimiento, educación y la formación en ciertos lugares del mundo donde por motivos políticos o religiosos se censuran.

Desde el punto de vista técnico se observa que es fundamental aplicar correctamente la seguridad en todas las capas de la aplicación, usando metodologías seguras de desarrollo. Esto implica utilizar lenguajes de programación seguros, elegir 'frameworks' y sistemas operativos libres que tengan conciencia en el ámbito de la seguridad.

Se ha observado que es fundamental el uso de las técnicas criptográficas para proteger la privacidad de las comunicaciones y la aplicación de firmas digitales sería un avance muy importante en este ámbito.

5.2. Líneas de trabajo futuro

Durante el desarrollo de este trabajo, han surgido ideas que sería interesante aplicar en futuras mejoras del proyecto.

Una idea interesante es extender a otras redes basadas en el anonimato como **I2P** o **FreeNet** el servicio. Comprender como funcionan estas alternativas y como se alojarían servicios equivalentes a MarkeTOR y comparar sus puntos fuertes y débiles.

En el ámbito del uso de criptomonedas, lo interesante es permitir el uso de más tipos, sobre todo algunas con orientación a la privacidad como pueden ser **Monero** o **DASH**. Se podría definir un API que abstrajera el uso de las criptomonedas, permitiendo añadir más tipos fácilmente.

Sin abandonar este ámbito, explorar otros sistemas como la **plataforma Ethereum y sus contratos inteligentes**.

Ya en el ámbito de la funcionalidad del MarkeTOR o servicios similares, se pueden aplicar mejoras en distintos ámbitos.

En las comunicaciones se ha observado la importancia que tiene la criptografía de clave pública para mantenerlas privadas. Se podría implementar un sistema interno de mensajería usando este tipo de criptografía para una comunicación más directa sin abandonar el servicio. También un sistema de valoración de productos y vendedores mejoraría de manera importante el servicio.

Pensando en la disponibilidad del servicio, se podría avanzar en un modelo distribuido, tanto del servicio como de la base de datos. Por ejemplo, se podría diseñar como un sistema basado en contenedores que pueden crecer horizontalmente de manera sencilla. Otra alternativa sería el uso de múltiples dispositivos IoT similares a Raspberry Pi o Tessel para el despliegue del servicio.

Bibliografía

Appelbaum, J. (2015, octubre). *RFC 7686 - The ".onion" Special-Use Domain Name.* Retrieved from https://tools.ietf.org/html/rfc7686

Ardións, A. (2017). *Cómo funciona Tor y para qué deberíamos utilizarlo*. Obtenido de https://www.profesionalreview.com/2017/01/01/como-funciona-tor/

Ballesteros, I. (17 de Octubre de 2018). *Estudio y análisis de vulnerabilidades de la Deep Web mediante la implementación de un nodo Tor.* Obtenido de https://upcommons.upc.edu/bitstream/handle/2117/123445/memoria.pdf?sequence=1&isAllowed=y

BBC News. (2013). *FBI shuts down Silk Road website*. Obtenido de https://www.bbc.com/news/av/technology-24378137/fbi-shuts-down-silk-road-website

BBC News. (2019, julio 22). *Russian intelligence 'targets Tor anonymous browser*. Retrieved from https://www.bbc.com/news/technology-49071225

boum.org. (s.f.). *Privacidad para todos en todas partes*. Obtenido de https://tails.boum.org

Cagiga, I. (2017 de Enero). *Deep Web: acceso, seguridad y analisis de tráfico.* Obtenido de https://repositorio.unican.es/xmlui/bitstream/handle/10902/10116/390204.pdf?sequence=1

cathugger. (2019). *mkp224o - Vanity address generator for tor onion v3 (ed25519) hidden services*. Obtenido de https://github.com/cathugger/mkp224o

dborbon. (s.f.). *¿Qué es Tor?* Recuperado el 29 de 05 de 2019, de http://www.utic.edu.py/citil/index.php/8-destacados/105-que-es-tor

Derechos Digitales América Latina. (2018). *Torifícate.* Obtenido de https://tor.derechosdigitales.org/torificate/

Díaz, M. (30 de Agosto de 2017). *Es legal en España navegar por la Deep Web*. Obtenido de https://clickjuridico.es/es-legal-navegar-por-la-deep-web/

Dingledine, R., Mathewson, N., & Syverson, P. (2004, agosto). *Tor: The Second-Generation Onion Router*. Retrieved from https://svn.torproject.org/svn/projects/design-paper/tor-design.html

Dingledine, R., Mathewson, N., & Syverson, P. (Junio de 2004). *Tor: The Second-Generation Onion Router.* Obtenido de https://svn.torproject.org/svn/projects/design-paper/tor-design.pdf

Electronic Frontier Foundation. (n.d.). *How HTTPS and Tor Work Together to Protect Your Anonymity and Privacy.* Retrieved from https://www.eff.org/pages/tor-and-https

Geeky Theory. (2019). *¿Qué es y cómo funciona la red Tor?* . Obtenido de https://geekytheory.com/que-es-y-como-funciona-la-red-tor

Goldfeder, S., Kalodner, H., Reisman, D., & Narayanan, A. (29 de agosto de 2018). *When the cookie meets the blockchain: Privacy risks of web payments via cryptocurrencies*. Obtenido de https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0207000

González, R. (Marzo de 2018). *Estudio y análisis de vulnerabilidades de la Deep Web mediante la implementación de un nodo Tor.* Obtenido de https://repositorio.unican.es/xmlui/bitstream/handle/10902/13370/406800.pdf?sequence=1

Guardian Project. (s.f.). *Orbot: Tor for Android*. Obtenido de https://guardianproject.info/apps/orbot/

Guardian Project. (s.f.). *Orfox: A Tor Browser for Android*. Obtenido de https://guardianproject.info/apps/orfox/

H.E. (27 de Agosto de 2012). *¿Qué es la Deep Web?* Obtenido de https://marcianosmx.com/que-es-la-deep-web/

Hochstadt, A. (2019). *The Complete List of Blocked Websites in China & How to Access Them*. Retrieved from https://www.vpnmentor.com/blog/the-complete-list-of-blocked-websites-in-china-how-to-access-them/

Juhász, P. L., Stéger, J., Kondor, D., & Vattay, G. (13 de diciembre de 2018). *A Bayesian approach to identify Bitcoin users*. Obtenido de https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0207000

Lasse , Ø., & Syverson, P. (mayo de 2006). *Locating Hidden Servers.* Obtenido de https://apps.dtic.mil/dtic/tr/fulltext/u2/a462140.pdf

Lavín, I. (Junio de 2018). *Un Paseo por la Deep Web.* Obtenido de http://openaccess.uoc.edu/webapps/o2/bitstream/10609/81908/6/irelaviTFM0618memoria.pdf

Lev, O. (mayo de 2019). *bit - Python`s bitcoin library*. Obtenido de https://pypi.org/project/bit/

Lueth, K. L. (2018). *State of the IoT 2018: Number of IoT devices now at 7B – Market accelerating*. Obtenido de https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/

M. Goldschlag, D., G. Reed, M., & F. Syverson, P. (1997). *Anonymous Connections and Onion Routing.* Retrieved from http://www.cs.jhu.edu/~fabian/courses/CS600.424/course_papers/syverson97anonymous.pdf

M. Goldschlag, D., G. Reed, M., & F. Syverson, P. (n.d.). *Hiding Routing Information.* Retrieved from

http://www.cs.jhu.edu/~fabian/courses/CS600.424/course_papers/goldschlag96hiding.pdf

Metrics Portal - Tor Portal. (2018). Retrieved from https://metrics.torproject.org/

Monero. (2019). Retrieved from https://www.getmonero.org/

Murdoch, S. J. (2008). *New Tor distribution for testing: Tor Browser Bundle*. Obtenido de https://lists.torproject.org/pipermail/tor-talk/2008-January/007837.html

Netresec.com. (s.f.). *NetworkMiner*. Obtenido de https://www.netresec.com/?page=networkminer

Oscuro, B. E. (20 de Febrero de 2017). *Que hay en los tres niveles de internet*. Obtenido de https://www.owldetect.com/es/blog-el-lado-oscuro/posts/niveles-profundidad-internet-clear-deep-dark-web/

OWL - El blog del lado Oscuro. (20 de febrero de 2017). *Qué hay en los tres niveles de Internet*. Obtenido de https://www.owldetect.com/es/blog-el-lado-oscuro/posts/niveles-profundidad-internet-clear-deep-dark-web/

Pontiroli, S. (octubre de 2013). *PGP – Privacidad, seguridad y autenticación fiables para todos*. Obtenido de https://www.kaspersky.es/blog/pgp-privacidad-seguridad-y-autenticacion-fiables-para-todos/1781/

Ranchal, J. (24 de Enero de 2014). *Buceando por la 'Internet Invisible' Deep Web*. Obtenido de https://www.muycomputer.com/2014/01/24/deep-web-introduccion/

Recker, A. (n.d.). *A GPG implementation for Django*. Retrieved from https://github.com/arecker/django-gpg

Reuters. (noviembre de 2013). *EEUU cierra Silk Road, un mercado negro de armas y drogas en la Red*. Obtenido de https://www.elmundo.es/america/2013/10/02/estados_unidos/1380737961.html

Rodriguez de Luis, E. (2018). *Qué es la dark web y qué podemos encontrar en ella*. Obtenido de https://urbantecno.com/tecnologia/que-es-dark-web

Scaife, J. (2018, febrero 8). *Tor Onion v3 Vanity Address*. Retrieved from https://www.jamieweb.net/blog/onionv3-vanity-address/

Schober, S. (2015, enero 27). *Deep Dark Web Of The Internet Iceberg*. Retrieved from https://scottschober.com/deep-dark-web-of-the-internet-iceberg/

tcpdump.org. (s.f.). *TCPDUMP&LIBPCAP*. Obtenido de https://www.tcpdump.org/

The Tor Project. (2009-2018). *Tor Metrics*. Obtenido de https://metrics.torproject.org/services.html

The TOR Project. (26 de abril de 2017). *Tor 0.3.0.6 is released: a new series is stable!* Obtenido de https://blog.torproject.org/tor-0306-released-new-series-stable

The TOR Project. (2018). *The Tor Relay Guide*. Retrieved from https://trac.torproject.org/projects/tor/wiki/TorRelayGuide

The TOR Project. (2019). *Software & Services*. Retrieved from https://2019.www.torproject.org/projects/projects.html.en

The TOR Project. (2019). *Tor Rendezvous Specification - Version 3*. Obtenido de https://gitweb.torproject.org/torspec.git/tree/rend-spec-v3.txt

The TOR Project. (2019). *TOR specification*. Obtenido de https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt

The TOR Project. (2019). *Tor: Onion Service Protocol*. Retrieved from https://2019.www.torproject.org/docs/onion-services.html.en

The TOR Project. (2019). *Tor: Onion Service Protocol*. Retrieved from https://2019.www.torproject.org/docs/onion-services.html.en

The Tor Project. (2019). *Tor: Pluggable Transports*. Obtenido de https://2019.www.torproject.org/docs/pluggable-transports.html.en

The TOR Project. (s.f.). *Historia del Proyecto TOR*. Obtenido de https://www.torproject.org/es/about/history/

The TOR project. (s.f.). *Onion Services*. Obtenido de https://tb-manual.torproject.org/es/onion-services/

The Tor Project. (s.f.). *Software & Services*. Obtenido de https://2019.www.torproject.org/projects/projects.html.en

The Tor Project. (n.d.). *Stem*. Retrieved from https://stem.torproject.org/#

The Tor Project. (s.f.). *Tor*. Obtenido de https://tb-manual.torproject.org/es/

Tor Nyx. (2019). Obtenido de https://nyx.torproject.org/

Varios Editores. (29 de Junio de 2019). *The Deep Web, The Dark Web & Tor: How To Surf The Secret Internet at WhoIsHostingThis.com*. Obtenido de https://www.whoishostingthis.com/blog/2017/03/07/tor-deep-web/

Whonix: A High Security Method of Surfing the Internet. (s.f.). Obtenido de https://www.whonix.org/

Wikipedia. (2019). *Encaminamiento cebolla*. Obtenido de https://es.wikipedia.org/wiki/Encaminamiento_cebolla

Winter, P., Edmundson, A., Roberts, L. M., Dutkowska-Zuk, A., Chetty, M., & Feamster, N. (06 de 2018). *How Do Tor Users Interact With Onion Services?* Obtenido de https://nymity.ch/onion-services/pdf/sec18-onion-services.pdf

wireshark.org. (s.f.). *WIRESHARK*. Obtenido de https://www.wireshark.org/

Worldometers. (19 de 05 de 2019). *internetlivestats*. Obtenido de https://www.internetlivestats.com