Universidad Zaragoza
1542

Trabajo Fin de Grado

Desarrollo de un sistema de control de versiones software integrado en Elastic Stack.

Development of a software control version system based on Elastic Stack.

Autor

Marina Bengoechea Cuadrado

Director

Álvaro Alesanco Iglesias

Escuela de Ingeniería y Arquitectura
2021

AGRADECIMIENTOS

Con este proyecto doy por concluida una de las etapas más importantes de mi vida, con todo lo que ello implica. Ha sido un camino largo y con muchos, muchos baches, pero, como yo digo, sin sacrificio no hay victoria. Solo puedo decir que ha valido la pena las noches sin dormir, los suspensos, las desilusiones y el trabajo realizado si todo ello ha servido para que en el día de hoy me encuentre donde estoy. Esta carrera me ha permitido desarrollarme no solo como futura profesional, sino también como persona. Los valores tan altos que se exigen de implicación, esfuerzo y constancia, me han convertido en una persona mucho más fuerte, válida y preparada. Por todo ello, estaré siempre orgullosa y satisfecha con la elección que hice cuando decidí entrar en ella.

Quiero dar mi más sincero agradecimiento a Álvaro por aceptar dirigir este trabajo depositando su confianza en mí. Gracias por su guía, su constante preocupación por mí y sus sabios consejos, pero, sobre todo, gracias por despertar en mí la pasión por la ciberseguridad desde el día en el que asistí a una de sus clases.

También quiero dar las gracias a mis profesores de la rama de telemática, por abrirme los ojos y hacer que me diera cuenta de lo que realmente me apasionaba gracias a la manera que han tenido de transmitir sus conocimientos clase tras clase.

Gracias también a mis mejores amigos, Luis y Marco, las mejores personas que me ha regalado esta carrera. Gracias por todas las tardes interminables en la biblioteca, los trabajos juntos y las risas en clase. Habéis hecho que todo haya sido más llevadero y sé que siempre podré contar con vosotros.

Eternamente agradecida a mi pareja, Noel, por todos estos años de apoyo y amor incondicional, por entenderme en todo momento incluso cuando ni yo misma lo hacía, pero, sobre todo, por creer siempre en mí y en mis capacidades logrando ayudarme así a ser una mejor versión de mí misma y evolucionar. Lo que Teleco ha unido no lo puede separar nadie.

No puedo terminar sin dar las gracias a mi familia, en especial a mis padres, Blanca y José Ángel, y a mi hermano Daniel. Gracias por ser mi faro siempre, por todos los sacrificios que habéis hecho, por levantarme cuando ni yo misma tenía fuerzas para seguir y siempre tener unas palabras de ánimo o una sonrisa con la que recibirme. Sé que mi sufrimiento ha sido también el vuestro, sólo me consuela que sé que mis alegrías también han sido las vuestras. Sin vosotros nada de esto habría sido posible. Sois la suerte de mi vida.

Por último y más importante, me gustaría dedicar este trabajo a mi abuelo José Manuel, por creer siempre en mí ciegamente, por sus muestras de orgullo constantes, sus palabras de ánimo y su preocupación por mí y mis estudios. Aunque ya no puedas ver esto, siempre habrá un pedacito de ti conmigo. Esto va por ti.

RESUMEN

La seguridad en redes y servicios consiste en adoptar políticas y prácticas para prevenir y supervisar el acceso no autorizado, el uso indebido, la modificación o la denegación de una red informática y sus recursos accesibles. Actualmente, la ciberseguridad está presente en todo tipo de organizaciones y empresas, pero también cada vez más en usuarios particulares. No nos quepa duda de que, todos y cada uno de los equipos expuestos al mundo exterior reciben varios ataques al cabo del día. Pensar que nadie va a atacar nuestra red porque no somos importantes es un cibersuicidio.

Durante este Trabajo Fin de Grado se ha desarrollado un sistema de control de versiones software con la finalidad de comparar las versiones instaladas en el SO del usuario con las últimas versiones comerciales disponibles de 15 softwares de uso común en el SO Windows. El objetivo es prevenir una futura explotación de vulnerabilidades de nuestro sistema por tener instaladas versiones no actualizadas.

Se han implementado un conjunto de técnicas y herramientas que nos han permitido sincronizar información en un servidor centralizado con el objetivo de procesarla y ofrecerla a nuestros usuarios a través de una API. Para ello, se ha utilizado el framework osquery para la obtención de versiones de software instaladas en el equipo del usuario; se ha desarrollado un programa en Python, el cual, mediante técnicas de web scraping, obtiene las últimas versiones comerciales de los programas más usados en el SO Windows y se ha utilizado el conjunto de herramientas de Elastic Stack (Beats, Logstash, Elasticsearch y Kibana) para llevar a cabo el procesamiento, almacenamiento y visualización de los resultados. Finalmente, se ha realizado una API (Application Programming Interfaces) en Python que ofrece a los usuarios información de los softwares analizados por nuestro sistema.

La funcionalidad del sistema ha sido comprobada mediante su despliegue en un entorno con accesibilidad pública y se ha verificado su funcionamiento en diferentes pruebas.

Índice

Lista de acrónimos

JSON. JavaScript Object Notation

IP. Internet Protocol

API. Application Programming Interface

HTTP. Hypertext Transfer Protocol

HTTPS. Hypertext Transfer Protocol Secure

REST. Representational State Transfer

SSL. Secure Socket Layer

TLS. Transport Layer Secure

CA. Certificate Authority

SQL. Structured Query Language

WMI. Windows Management Instrumentation

SO. Sistema Operativo

VPN. Virtual Private Network

PPP. Point-to-Point Protocol

USB. Universal Serial Bus

URL. Uniform Resource Locator

Capítulo 1

Introducción

1.1. Introducción

Cada vez es más habitual encontrarnos con noticias de ciberataques a empresas de todo el mundo, desde el secuestro de ordenadores que impide el normal funcionamiento de la compañía, hasta el robo de sus bases de datos, lo que implica un enorme problema de seguridad para las mismas. Si bien es cierto que en la prensa salen los casos más graves y que afectan a compañías muy importantes, no debemos dejar de lado la idea de que este es un problema real, independientemente de dónde se ubique la empresa o cuál sea su tamaño, pues hoy en día las nuevas tecnologías son una de las principales herramientas que tienen las organizaciones para llevar a cabo sus funciones.

Los particulares también pueden ser víctimas de cualquier "ciberdelincuente", pues es mucha la información personal y confidencial que tenemos online. A nivel más personal, la importancia de la ciberseguridad tiene que ver con nuestros datos, como contraseñas o datos bancarios y de la importancia de estar informado para no caer en estafas o correos de phishing. Los atacantes aprovechan cualquier brecha de seguridad o inexperiencia del usuario para intentar atacar a nuestros sistemas.

España es el tercer país más atacado del mundo, solo por detrás de Estados Unidos y Reino Unido, con un total de 115.000 incidencias informáticas (tanto a empresas como a particulares).

Este proyecto es una muestra de cómo prevenir posibles ataques, ya que si nuestro software instalado está actualizado a la última versión habrá menos posibilidades de que el atacante utilice esa brecha de seguridad para intentar ejecutar un ataque aprovechando esa vulnerabilidad.

1.2. Objetivos

El objetivo principal de este proyecto es la elaboración de un sistema de control de versiones software utilizando técnicas de web scraping y el conjunto de herramientas de la arquitectura *Elastic Stack* con el fin de prevenir posibles ataques y explotación de las vulnerabilidades del usuario. Se plantean, además, una serie de objetivos específicos para alcanzar el desarrollo óptimo del proyecto:

- Estudio del framework *osquery* para la obtención de la información del equipo del usuario.

- Estudio y comprensión de la arquitectura *Elastic Stack* y su capacidad de procesado, análisis, almacenamiento y visualización.

- Generación de una serie de scripts en Python para la obtención de las versiones de software mediante web scraping y la creación de la API.

- Creación de visualizaciones representativas mediante *Kibana*.

- Comunicación entre herramientas de forma segura (SSL/TLS).

En la solución que se ha desplegado, la infraestructura ha sido diseñada para poder ser escalable en un futuro y hacer frente a un mayor tráfico de datos, a pesar de que para este proyecto el volumen de datos no haya sido tan elevado.

1.3. Herramientas utilizadas

- osquery. Marco de instrumentación del SO que permite explorar los datos del sistema mediante consultas SQL. Con *osquery*, las tablas SQL representan conceptos abstractos como procesos en ejecución, módulos de kernel cargados, conexiones de red abiertas, complementos de navegador, eventos de hardware o hashes de archivos.

- Elastic Stack. Solución gratuita que ofrece un conjunto de herramientas de código abierto que se combinan para crear una única plataforma de administración de registros permitiendo el procesado de eventos *(Logstash)*, análisis y almacenamiento *(Elasticsearch)* y visualización *(Kibana)* de los mismos.

- Debian. Sistema operativo en el que se ha desarrollado el proyecto. Es una distribución gratuita de Linux, con una instalación rápida e intuitiva, que presenta una gran estabilidad y cuenta con una ingente cantidad de software. Muestra una de las mejores relaciones entre funcionalidad y recursos empleados.

- Postman. Software que permite el envío de peticiones HTTP REST sin necesidad de desarrollar un cliente.

- Python. Lenguaje de programación versátil multiplataforma y multiparadigma que destaca por su código legible y limpio.

1.4. Organización de la memoria

La memoria se ha organizado siguiendo la siguiente disposición:

- **Capítulo 1.** Presentación del trabajo, los objetivos a conseguir y las herramientas utilizadas durante el desarrollo del mismo.

- **Capítulo 2.** Introducción a las herramientas utilizadas durante todo el proceso de desarrollo del proyecto.

- **Capítulo 3.** Explicación del desarrollo y funcionamiento del sistema completo.

- **Capítulo 4.** Explicación de cómo se ha desarrollado el sistema y las pruebas realizadas para la comprobación de su correcto funcionamiento.

- **Capítulo 5.** Exposición de las conclusiones del trabajo y las posibles líneas futuras.

Finalmente, se han añadido los siguientes anexos al final del documento:

- **Anexo 1.** Archivo configuración osquery

- **Anexo 2.** Scripts de Python
  - **Anexo 2.1.** elastic.py
  - **Anexo 2.2.** scraping.py
  - **Anexo 2.3.** api.py

- **Anexo 3.** Archivo configuración Filebeat

- **Anexo 4.** Archivo configuración Logstash

- **Anexo 5.** Guía de instalación Elastic Stack
  - **Anexo 5.1.** Instalación y configuración Filebeat
  - **Anexo 5.2.** Instalación y configuración Elasticsearch
  - **Anexo 5.3.** Instalación y configuración Kibana
  - **Anexo 5.4.** Instalación y configuración Logstash

- **Anexo 6.** Incorporación del protocolo SSL/TLS en la comunicación

- **Anexo 7.** Planificación temporal

Capítulo 2

Conceptos previos

2.1. Elastic Stack

Elastic Stack es un conjunto de herramientas de código abierto que se combinan para que el usuario pueda tomar de manera confiable y segura datos de cualquier fuente, en cualquier formato para después buscarlos, analizarlos y visualizarlos en tiempo real. *Elastic Stack* hace uso de tres herramientas *(Elasticsearch, Logstash y Kibana)* que, aunque pueden ser utilizadas de manera independiente, unidas crean una combinación perfecta para la gestión de eventos y es, de esta forma, como haremos uso de ellas durante el desarrollo de este proyecto.

- Logstash. Pipeline de procesamiento de datos del lado del servidor que ingesta datos de una multitud de fuentes, los transforma independientemente de su formato o complejidad y luego los envía donde el usuario desee.

- Elasticsearch. Base de datos distribuida capaz de abordar un número creciente de casos de uso. Como núcleo del *Elastic Stack*, almacena de forma central los datos para una búsqueda a velocidades asombrosas. Al estar todo indexado podemos acceder a los datos de forma prácticamente instantánea.

- Kibana. Interfaz de usuario que permite visualizar los datos de *Elasticsearch* y navegar en *Elastic Stack*. Esta visualización puede hacerse en forma de tablas, gráficos y mapas.

En la *Figura 2.1* se puede observar una vista de alto nivel del flujo de trabajo de *Elastic Stack*. El proceso comienza cuando *Logstash* recibe los logs, procesándolos y dándoles el formato requerido antes de enviarlos a *Elasticsearch* para su almacenamiento. Posteriormente, se procederá a su visualización haciendo uso de *Kibana*.

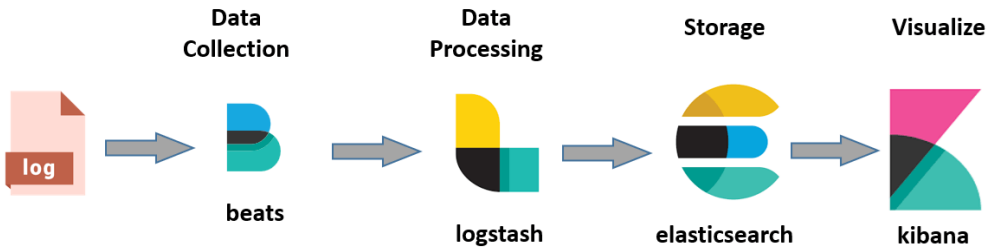

Figura 2.1: Flujo de trabajo de Elastic Stack

Cabe mencionar que, como puede apreciarse en la *Figura 2.1* , aparecen junto a *Logstash* los denominados *Beats*. Estos *Beats* se pueden añadir de forma opcional y se encargan de la recolección de información. A continuación, estos *Beats* envían los datos a *Elasticsearch*, ya sea directamente o a través de *Logstash* para que puedan ser visualizados con *Kibana*.

Existen múltiples alternativas a *Elastic Stack* (*Splunk, FluentD, Grafana,* etc). Sin embargo, para este proyecto se decidió el uso de la tecnología *Elastic Stack*. Las principales características por las que nos inclinamos hacia su implantación son las siguientes:

- Datos en tiempo real. *Elastic Stack* permite la obtención y el análisis de los datos de forma casi instantánea, apoyándose en gráficas y numerosos tipos de representaciones que proporcionan una mayor información de los resultados de manera clara y visual para el usuario.

- Sistema distribuido, ya que esta solución está formada por diferentes herramientas *(Beats, Logstash, Elasticsearch y Kibana)* cada una con una función específica pero que colaboran entre sí para el correcto funcionamiento del sistema global y la visualización de los resultados a partir de los datos obtenidos.

- Multi-tendencia de datos. Gracias a la indexación, nos permite obtener de forma directa los datos que necesitamos de manera prácticamente inmediata, atendiendo a diferentes agrupaciones o criterios de forma independiente.

- Escalabilidad. A medida que el volumen de datos crece, la plataforma ofrece un escalado óptimo y fácil de implementar, pudiendo balancear la carga a nivel de datos de sistema y red.

- Gran comunidad. La existencia de una gran cantidad de información acerca de esta pila de herramientas ha hecho que se haya globalizado su implantación de forma irrefutable.

2.2. Osquery

La necesidad de recopilar información del equipo local conllevó realizar una búsqueda exhaustiva sobre posibles opciones para llevar a cabo esta tarea. En un primer momento se planteó la posibilidad de obtener dicha información mediante *WMI (Windows Management Instrumentation).* Este protocolo funciona como una base de datos ofreciendo una variedad de información útil para el monitoreo de los sistemas basados en Windows. Sin embargo, aunque esta solución podría parecer válida en un principio, no se obtuvieron los resultados deseados. Nuestro objetivo era obtener el nombre y versión de los programas instalados en el equipo y con esta solución no aparecían la totalidad de estos. Esto se debe a que en los comandos *WQL* que utiliza este protocolo se utiliza la clase *Win32_Product*. Es por este motivo por el que sólo aparece el software instalado mediante *Windows Installer*. Finalmente, se optó por

descartar esta solución y buscar otra que nos proporcionara los requisitos que necesitábamos. La búsqueda finalizó con el hallazgo de *osquery*.

Osquery es un framework que instrumenta nuestro sistema operativo permitiendo luego hacer consultas en formato *SQL* sobre la infraestructura de nuestro equipo. Permite, por ejemplo, hacer consultas sobre los procesos en ejecución, usuarios, cambios en contraseñas, dispositivos USB, puertos abiertos, etc. *Osquery* exporta esta información como un conjunto de tablas *SQLite*. Esta función es clave para analizar errores, diagnosticar sistemas, errores a nivel global de la red, y arreglar problemas de rendimiento. Este framework nos ha permitido obtener de manera sencilla la información que buscábamos con un simple acceso a dos de sus tablas.

Por todo esto, se ha optado por el uso de este framework para obtener la información del usuario, en nuestro caso, el nombre y versión de los programas instalados y el SO que posee el usuario.

Capítulo 3

Arquitectura y desarrollo del sistema

3.1. Planteamiento general

Con el objetivo de comparar las versiones software instaladas por el usuario con las más actuales en el mercado, hemos desarrollado un sistema centralizado que aprovecha la arquitectura *Elastic Stack* con la finalidad de que el usuario pueda consultar las últimas versiones de sus programas y así evitar futuros ciberataques. Para ello, se ha desplegado un servidor Debian con una IP pública de forma que sea accesible desde cualquier localización. El servidor será el núcleo de nuestro sistema, sobre él se recibirán y tratarán los datos de nuestros equipos Windows y será el que albergue la mayor parte de nuestras aplicaciones.

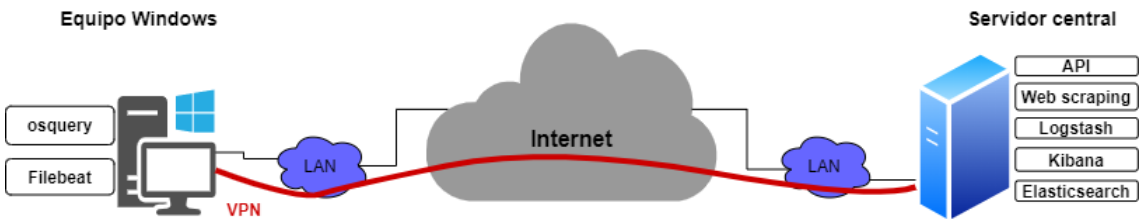

Figura 3.1: Esquema general del sistema

Sobre la estructura planteada se han instalado las aplicaciones utilizadas y desarrolladas en este proyecto, relatando el funcionamiento conjunto del sistema sobre el diagrama lógico que se muestra en la *Figura 3.2*.

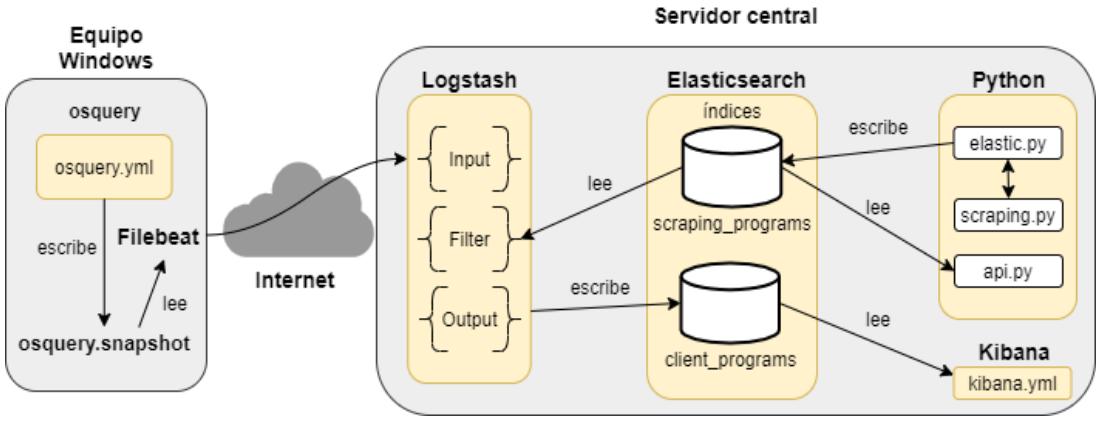

Figura 3.2: Diagrama lógico del sistema

En primer lugar, para poder comparar información de las versiones de software de nuestro equipo Windows es necesario obtenerlas. Para ello, se ha instalado y configurado *osquery* en la máquina local Windows para extraer dicha información periódicamente. La aplicación *Filebeat* cogerá dicha información y la enviará a nuestro servidor mediante el establecimiento de una conexión segura (SSL/TLS), en concreto, a *Logstash* para su posterior procesado. En paralelo, desde la aplicación de web scraping se recabarán las últimas versiones de software de los programas establecidos y se guardarán en el índice *scraping_programs* de la base de datos *Elasticsearch*.

Con la información de los programas del equipo Windows, por un lado, y la información de las últimas versiones en el índice *scraping_programs* por otro, podemos compararlos y visualizarlos.

Logstash recogerá la información que envía *Filebeat,* la comparará con los datos del índice correspondiente a web scraping, los filtrará y finalmente escribirá en el índice *client_programs* los programas del equipo Windows con su versión actual y su versión actualizada. De esta manera el usuario tendrá acceso a ello en todo momento.

Los datos e información de los equipos (*índice client_programs de Elasticsearch*) podrá ser visualizada mediante *Kibana* y la información de las últimas versiones (*índice scraping_programs de Elasticsearch*) será accesible, además, desde una API. Sobre esta información es sobre la que el usuario que desee consultar nuestro sistema real hará las peticiones. De esta manera podemos distinguir dos modos de funcionamiento de nuestro sistema: el primero, que constaría de la visualización de los resultados mediante *Kibana* los cuales se irían actualizando periódicamente, y el segundo, que estaría basado en la obtención de la información mediante el envío de queries a nuestra API.

3.2. Web scraping

Uno de los pilares sobre los que se fundamenta este proyecto son las técnicas de web scraping. Estas técnicas consisten en el desarrollo de programas de software para extraer información de sitios web mediante una conexión HTTP. En este caso se ha utilizado Python para programar una serie de métodos (uno por cada programa a consultar) en los cuales extraemos la información que nos interesa de la página de *release notes* oficial de cada software. Para este proyecto, la información a extraer será la última versión disponible en el mercado del software consultado.

Mediante la librería *Beautiful Soup*, hemos sido capaces de acceder al HTML de cada página web usando el elemento HTML *Anchor <a>* y el atributo *href.* El primero crea un enlace a otras páginas de internet, archivos o ubicaciones dentro de la misma página, direcciones de correo, o cualquier otra URL. Por otra parte, el atributo *href* contiene una URL o un fragmento de URL al cual apunta el enlace. En dicho enlace se encontraba escrito el número de la versión que buscábamos. Gracias a dichos elementos y a la programación en Python hemos sido capaces de extraer los datos necesarios para el correcto desarrollo del proyecto.

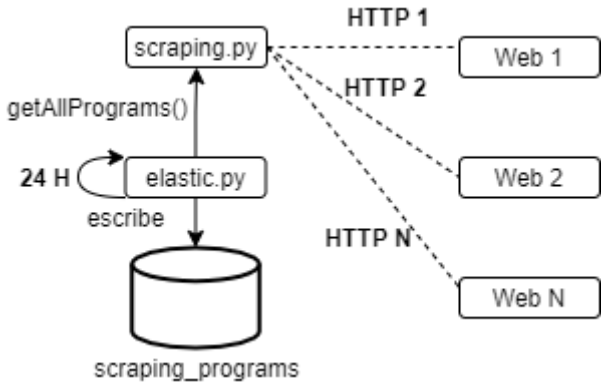

Figura 3.3: Diagrama de funcionamiento de web scraping

Tal y como representa la *Figura 3.3,* el programa lanza N peticiones HTTP (una por cada software a consultar), procesa la respuesta y la inyecta en el índice *scraping_programs*.

Entrando un poco en detalle, nuestra aplicación web scraping contiene dos scripts programados en Python, el primero se denomina *scraping.py* y el segundo, *elastic.py*. En el primero están programados todos los métodos para la obtención de las últimas versiones de nuestros 15 programas de uso común, mientras que el segundo corresponde al script desde el que lanzamos todos estos métodos guardando su respuesta en *Elasticsearch.* El código de ambos scripts puede verse detallado en el *Anexo 2*.

La aplicación de web scraping se ejecuta de forma periódica una vez al día, de tal modo que siempre tengamos la base de datos *scraping_programs* actualizada.

Los 15 programas que se han escogido para realizar su análisis de versiones con web scraping son los siguientes:

- Mozilla Firefox. Navegador y software de código abierto gratuito que se caracteriza por un gran número de posibles personalizaciones y ajustes.

- Adobe Acrobat Reader DC. Estándar global gratuito y de confianza para ver, imprimir, firmar, compartir y comentar archivos PDF. Es el único visor de PDF que puede abrir e interactuar con todo tipo de contenido PDF, incluidos formularios y contenido multimedia.

- VLC. Reproductor multimedia libre y de código abierto multiplataforma que reproduce la mayoría de archivos multimedia, así como DVD, Audio CD, VCD y diversos protocolos de transmisión.
- Malwarebytes. Software anti-malware que detecta y elimina numerosos tipos de malware proporcionando seguridad al usuario y prevención contra posibles ataques.

- Python Standard Library. Biblioteca estándar que se distribuye con Python. También describe algunos de los componentes opcionales que se incluyen comúnmente en las distribuciones de Python.

- Google Chrome. Navegador web de Google. Está diseñado para navegar de una manera simple y rápida. Además, se puede personalizar el navegador conforme a las necesidades del usuario.

- Micorsoft OneDrive. Es el servicio en la nube de Microsoft que conecta al usuario con todos sus archivos. Permite almacenar y proteger los archivos, compartirlos con otros usuarios y acceder a ellos desde cualquier lugar en todos los dispositivos.

- uTorrent. Cliente BitTorrent freeware que sirve para descargar todo tipo de archivos por redes P2P.

- WinRAR. Software de compresión de datos. Esta herramienta puede ser utilizada para abrir, comprimir y descomprimir descargas y ficheros adjuntos comprimidos.

- Oracle VM VirtualBox. Software de virtualización para arquitecturas x86/amd64, es decir, permite hacer máquinas virtuales con instalaciones de sistemas operativos.

- Telegram Desktop. Cliente oficial de la aplicación de mensajería Telegram que puede instalarse y utilizarse en Windows, macOS y GNU/Linux. Esta aplicación de mensajería y VoIP está enfocada a la mensajería instantánea, el envío de varios archivos y la comunicación en masa.

- GIMP. Programa de edición de imágenes digitales en forma de mapa de bits, tanto dibujos como fotografías. Es un programa libre y gratuito.

- Notepad++. Editor de texto y de código fuente libre con soporte para varios lenguajes de programación y soporte nativo para Microsoft Windows. Se parece al bloc de notas en cuanto al hecho de que se puede editar texto sin formato y de forma simple.

- Wireshark. Analizador de protocolos open-source. Su principal objetivo es el análisis de tráfico, pero además es una excelente aplicación didáctica para el estudio de las comunicaciones y para la resolución de problemas de red.

- PuTTY. Cliente SSH, Telnet, Rlogin, y TCP raw con licencia libre.

A continuación, podemos ver un ejemplo de los métodos creados para la realización del web scraping y obtener así la versión del software, en este caso, Mozilla Firefox.

```
def getFirefoxVersion():
      url1 = "https://www.mozilla.org/en-US/firefox/releases/"
      response = requests.get(url1)
      soup = BeautifulSoup(response.text, 'html.parser')
      quotes_html = soup.find_all("a")
      result = ""
       i = 0

      for q in quotes_html:
                  if("releasenotes" in str(q.get("href")) and i
                  == 0):
                        result = str(q.get("href")).split("/")[2]
                        i = 1
      return result
```

Listing 1: Método web scraping de Mozilla Firefox

El resto de los métodos programados se encuentran detallados en el *Anexo 2.2.*

3.3. API

Con el ánimo de ampliar la funcionalidad del servicio desarrollado, se optó por el desarrollo de una API *(Application Programming Interface)* para poder acceder al servicio mediante métodos HTTP. Una API es un conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. En nuestro caso, su finalidad será que el usuario pueda realizar consultas a la API programando en su software la petición correspondiente. La API se ha desarrollado con el objetivo de ofrecer información de forma segura y por lo tanto permitir la comunicación vía HTTPS.

A continuación, explicaremos los distintos métodos de nuestra API:

Realizando una petición HTTPS de tipo GET a la *URL* https://ipserver/getall el usuario recibirá una lista de todos los programas disponibles en nuestro sistema y su última versión disponible. Dicha lista se compondrá de un array de objetos *JSON* con el siguiente formato:

```
[{"name": "name1", "version": "version1"}, {"name":
"name2", "version": "version2"}, {"name": "name3",
"version": "version3"}, {"name": "name4", "version":
"version4"}, {"name": "name5", "version":"version5"}, {"name":
"name6", "version": "version6"}, {"name": "name7",
"version": "version7"}, {"name": "name8", "version":
"version8"}, {"name": "name9", "version": "version9"}, {"name":
"name10", "version": "version10"}, {"name": "name11", "version":
"version11"}, {"name": "name12", "version": "version12"},
```

```
10    {"name":        "name13",        "version":        "version13"},
11    {"name": "name14", "version": "version14"}, {"name":
12    "name15", "version": "version15"}]
```

Listing 2: Respuesta de la API a la petición a la URL getall

Con una petición HTTPS de tipo GET a la *URL* https://ipserver/getprograms el usuario recibirá una lista de los nombres de todos los programas disponibles a consultar en nuestro sistema. En este caso la respuesta también será un array de objetos *JSON* pero sin el campo de la versión.

```
1     ["name1", "name2", "name3", "name4", "name5", "name6", "name7",
2     "name8",  "name9",  "name10",  "name11",  "name12",  "name13",
3     "name14", "name15"]
```

Listing 3: Respuesta de la API a la petición a la URL getprograms

Con una petición HTTPS de tipo POST a la *URL* https://ipserver/lastversion el usuario podrá consultar la última versión de los programas que desee de entre todos los disponibles simplemente introduciendo como un array de strings *JSON* el nombre (campo *"name"*) de los que está interesado. El formato se presenta a continuación:

```
1     ["name1","name2", "name3","name4","nameN",]]
```

Listing 4: Campo body de la petición a la URL lastversion

A lo que la API responderá con el nombre y la versión de cada programa:

```
1     [{"name": "name1", "version": "version1"}, {"name": "name2",
2     "version": "version2"}, {"name": "name3", "version":"version3"},
3     {"name": "name4", "version": "version4"}, {"name": "nameN",
4     "version": "versionN"}]
```

Listing 5: Respuesta de la API a la petición a la URL lastversion

Realizando una petición HTTPS de tipo POST a la *URL* https://ipserver/updated el usuario puede introducir como un array de objetos *JSON* el nombre del programa o programas junto con las versiones actuales de los que desea saber si están actualizados. Este método le devolverá otro array de objetos *JSON* con la información de la última versión de ese programa (campos *"name"*, *"version"* y *"lastversion"*), y le dirá al usuario si está actualizada o no mediante el campo *"updated"*. La petición tendrá la siguiente estructura:

```
1     [{"name": "name1", "version": "version1"},{"name": "name2",
2     "version": "version2"}]
```

Listing 6: Campo body de la petición de la URL updated

La respuesta de la API a esta petición será la siguiente:

```
1     [{"name": "name1", "version": "version1", "updated": "yes/no",
2     "lastversion": "lastversion1"}, {"name": "name2", "version":
3     "version2", "updated": "yes/no", "lastversion": "lastversion2"}]
```

Listing 7: Respuesta de la API a la petición a la URL updated

Como información complementaria, cabe destacar que a nivel interno las peticiones las recibimos al puerto 443 aunque desde fuera tengan que enviarse a otra IP y puerto ya que estamos efectuando un DNAT para poder filtrar las peticiones entrantes.

3.4. Integración en Elastic Stack

Como ya se ha comentado en apartados previos, podemos ver que la columna vertebral de este proyecto es la arquitectura *Elastic Stack*, ya que gracias a todas sus herramientas nos ha permitido desarrollar un sistema unificado y flexible. Para ello, se han establecido las comunicaciones entre todas las herramientas que forman esta pila (*Beats, Logsatsh, Elasticsearch y Kibana)* de forma que se comuniquen conjuntamente mediante el protocolo de cifrado de datos SSL/TLS en el caso de la comunicación entre *Filebeat* y *Logstash,* y mediante autenticación con contraseña en el caso del acceso a *Elasticsearch.*

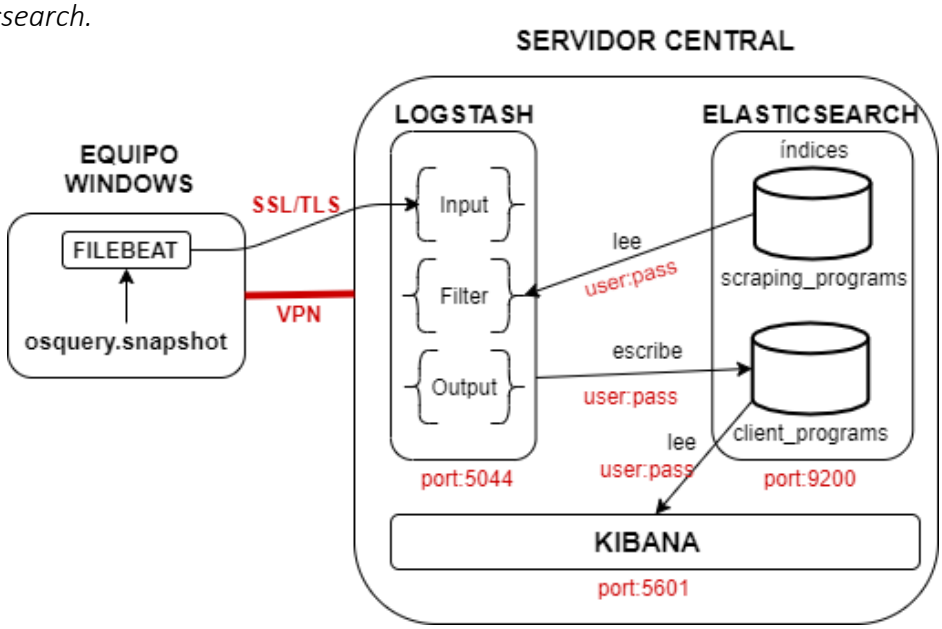

Figura 3.4: Integración en Elastic Stack

3.4.1. Filebeat

Anteriormente se ha comentado la existencia de *Beats* dentro de la estructura de *Elastic.* Dicha familia de *Beats* son diferentes tipos de agentes para diferentes tipos de datos. Para este proyecto, de entre toda esta familia, se ha elegido *Filebeat* ya que está especializado en archivos de log. Este *Beat* ha sido seleccionado ya que su función es el envío del archivo de log creado por *osquery* a *Logstash* para su posterior procesamiento.

Para la configuración de *Filebeat* será necesario instalar el servicio en el equipo del usuario que vamos a monitorizar, es decir, que quiera usar nuestro sistema. Además, deberemos configurar la salida para que se dirija hacia *Logstash* por su puerto correspondiente siendo el flujo de paquetes de la siguiente manera:

Equipo local → Filebeat → Logstash → Elasticsearch → Kibana

Cabe mencionar que se ha expedido un certificado para *Filebeat* y por lo tanto se ha otorgado una comunicación segura entre este y *Logstash* para garantizar la integridad de los datos del usuario.

3.4.2. Logstash

En cuanto a la herramienta *Logstash*, su función es procesar los datos que le llegan a través de *Filebeat*. Este procesado consiste en hacer *match* entre los programas instalados por el usuario y los programas obtenidos mediante web scraping.
Para ello, se han configurado cada una de las tres secciones de *Logstash* en su archivo de configuración: *input, filter y output.*

En la sección de *entrada o input* hemos definido y utilizado dos *plugins* diferentes para el tratamiento de los datos de entrada.

- beats. Permite a *Logstash* recibir eventos de la infraestructura *Beats*. En nuestro caso particular, los datos son recibidos a partir de *Filebeat*.

- ssl. Mediante la habilitación de este plugin, conseguimos que el protocolo SSL/TLS esté activo mediante la posterior asignación de un certificado de *Autoridad de Certificación (ca.crt),* un certificado para *Logstash (logstash.crt)* y una *clave (logstash.key).*

Con los datos ya recibidos en *Logstash*, necesitamos definir un filtrado para poder quedarnos con los datos que necesitamos y hacer *match* en este caso entre los datos provenientes de *Filebeat* y los que ya tenemos en el índice *scraping_programs.*

Para ello, en la sección *filter* se han empleado los plugins *mutate, elasticsearch* y *drop*. Con el primero tenemos la posibilidad de aplicar numerosos tipos de filtros. En nuestro caso, han sido los siguientes:

- split. Obtiene variables de forma individual en lugar de un mensaje unificado.

- remove_field. Elimina el campo seleccionado de entre todos los campos del evento.

- add_field. Añade un campo al evento, enriqueciéndolo.

El plugin *elasticsearch* busca eventos previos de log de *Elasticsearch* y copia algunos campos al evento actual. En nuestro caso lo que hacemos es coger el campo *<version>* de cada uno de los programas obtenidos mediante web scraping y asignarlo al campo *<latest_version>* del evento actual. De esta manera, ya tendremos actualizado ese campo y sabremos cuál es la última versión de cada uno de los softwares analizados.

Finalmente, con el plugin *drop* simplemente conseguimos que *Logstash* descarte todo lo que llegue hasta él. En nuestro caso, descartaremos todos aquellos programas que tenga instalados el usuario pero que no se encuentren contemplados entre todos los analizados vía web scraping, ya que no podremos ofrecerle ningún tipo de información acerca de ellos al usuario.

La sección de *salida u output*, etapa final del flujo de datos de *Logstash*, envía la información obtenida a través del input y procesada a través del filtro a un destino en particular. Para este proyecto se ha utilizado únicamente el plugin *elasticsearch*. Mediante el uso de dicho plugin hacemos que los datos ya procesados se introduzcan en la base de datos *Elasticsearch*. Además, se han añadido los campos *<user>* y *<password>* para el acceso a esta base de datos consiguiendo segurizar así nuestro sistema.

En el *Anexo 4* se explican todos los pasos que se han seguido en el procesamiento de los datos mediante *Logstash.*

3.4.3. Elasticsearch

En la base de datos *Elasticsearch* guardaremos cada uno de los programas con sus respectivas versiones actualizadas. Se crearán dos bases de datos (denominaos índices para el caso particular de *Elasticsearch*), una para guardar la información de los programas obtenidos mediante web scraping denominada *scraping_programs* y otra para guardar la información de los programas instalados en el equipo del usuario (*client_programs)*.

Los campos que se guardan en el índice *scraping_programs* son los siguientes:

- ***<name>***: indica el nombre del software analizado.
- ***<version>:*** indica la versión del software analizado.

Los campos que se guardan en el índice *client_programs* son los siguientes:

- ***<name>***: indica el nombre del software analizado.
- ***<version>:*** indica la versión del software instalado.
- ***<last_version>***: indica la última versión comercial disponible obtenida mediante web scraping.
- ***<updated>***: este campo indica si la versión está actualizada o no, es decir, si el campo *<version>* coincide con *<last_version>*, en cuyo caso este campo tendrá el valor de "yes". En caso contrario, su valor será "no".
- ***<_id>***: indica el identificador asignado a cada elemento de la tabla. Este identificador es aleatorio. Este identificador es una combinación de letras y números.
- ***<_index>***: indica el nombre asignado a la base de datos. En este caso, *client_programs.*
- ***<_type>***: indica el tipo de evento que se ha recolectado. Este parámetro sirve para poder realizar una búsqueda más rápida entre todos los eventos atendiendo al tipo.
- ***<@timestamp>***: etiqueta temporal que indica el momento exacto en el ocurrió el evento.
- ***<host.architecture>***: indica el tipo de arquitectura del dispositivo en el que se originó el evento (e.g. x86_64, arm, ppc, mips).
- ***<host.hostname>***: indica el nombre del dispositivo en el que se originó el evento.
- ***<host.id>***: indica el identificador que designa de forma única al equipo monitorizado en el que se originó el evento. Este identificador consta de una combinación de letras y números.
- ***<host.ip>***: lista de direcciones IP (tanto IPv4 como IPv6) del equipo monitorizado en el que se originó el evento.

- ***<host.mac>:*** lista de direcciones MAC del equipo monitorizado en el que se originó el evento.

- ***<host.name>*:** indica el nombre del dispositivo en el que se originó el evento, al igual que el campo *<host.hostname>*.

- ***<host.os.build>*:** información acerca de la versión o "build" de la versión de Windows 10 que tiene el usuario (e.g. "18D109").

- ***<host.os.family>*:** indica la familia de SO a la cual pertenece el equipo en el que se originó el evento (e.g redhat, debian, freebsd, windows).

- ***<host.os.kernel>*:** indica la versión de kernel del SO del equipo monitorizado mostrado como un único string.

- ***<host.os.name>*:** nombre del SO específico del equipo en el que surgió el evento (e.g "Windows 10 Pro").

- ***<host.os.platform>*:** plataforma del SO de la máquina que originó el evento (e.g centos, ubuntu, windows).

- ***<host.os.version>*:** versión del SO del equipo monitorizado mostrado como un string (e.g "10.14.1").

Dicha tabla podrá visualizarse accediendo vía HTTPS a *Kibana*. De esta manera, puede tenerse una visión más compacta y clara de los resultados. Tanto el acceso a *Kibana* como el acceso a los índices de *Elasticsearch* se realiza mediante autenticación con contraseña.

Capítulo 4

Despliegue y pruebas

4.1. Despliegue

El desarrollo realizado nos proporciona un sistema flexible y portable que permite la realización de pruebas reales sin la necesidad de instalar aplicaciones sobre una localización fija. La aplicación *osquery* se ha instalado sobre un dispositivo Windows 10. A su vez, *Filebeat* está necesariamente instalado en dicho dispositivo. Tanto la base de datos *Elasticsearch, Logstash, Kibana*, la API y la aplicación de web scraping han sido instalados en el servidor central, una máquina Debian 10 con 8 GB de RAM sobre la red de la universidad.

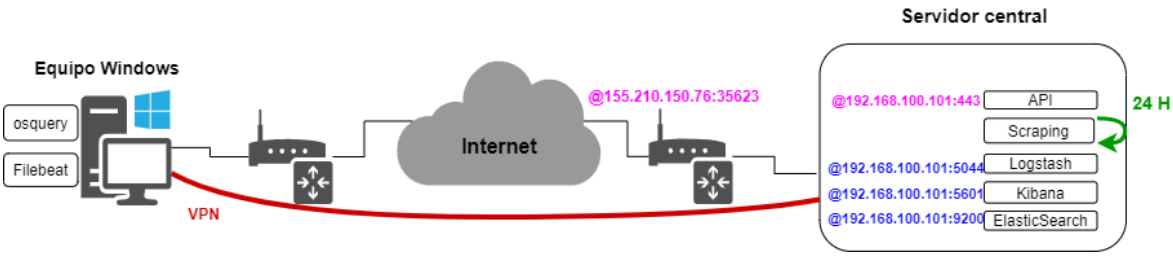

Figura 4.1: Despliegue del sistema

En la *Figura 4.1* hemos podido ver en el flujo representado que es necesario un DNAT para el acceso a nuestra API. Las peticiones de envían a la IP @155.210.150.76:35623 y el DNAT las redirecciona al interior de la red de la universidad (@192.168.100.101:443). Como puede verse, el puerto 443 es el puerto en el que la API espera peticiones HTTPS para poder responderlas. También cabe destacar que la periodicidad con la que se ejecuta los scripts de scraping se produce gracias al fichero *cron* de Linux, donde podemos especificar los scripts que queremos ejecutar y cada cuanto tiempo lo harán.

4.2. Pruebas

Para el caso del uso de nuestro proyecto como sistema de monitorización se hará uso de *Kibana* para la visualización completa de los resultados. Una vez al día dichos resultados se verán actualizados en función de si se han encontrado nuevas versiones de los softwares contemplados o no.

Time	name	version	latest_version	updated
Jun 10, 2021 @ 22:54:30.768	GIMP 2.10.22	2.10.22	2.10	yes
Jun 10, 2021 @ 22:54:30.768	Oracle VM VirtualBox 6.1.22	6.1.22	6.1.22	yes
Jun 10, 2021 @ 22:54:30.768	Google Chrome	91.0.4472.77	91	yes
Jun 10, 2021 @ 22:54:30.768	Mozilla Firefox 88.0.1 (x64 es-ES)	88.0.1	89.0	no
Jun 10, 2021 @ 22:54:30.768	VLC media player	3.0.11	3.0.15	no
Jun 10, 2021 @ 22:54:30.768	WinRAR 5.91 (64-bit)	5.91.0	6.00	no
Jun 10, 2021 @ 22:54:30.768	PuTTY release 0.74 (64-bit)	0.74.0.0	0.75	no
Jun 10, 2021 @ 22:54:30.768	Malwarebytes version 4.3.0.98	4.3.0.98	4.4	no
Jun 10, 2021 @ 22:54:30.768	Notepad++ (32-bit x86)	7.9.1	8.0	no
Jun 10, 2021 @ 22:54:30.768	Wireshark 3.4.2 64-bit	3.4.2	3.4.6	no
Jun 10, 2021 @ 22:54:30.768	Adobe Acrobat Reader DC - Español	21.001.20155	21.005.20048	no

Figura 4.2: Visualización de resultados mediante Kibana

En la *Figura 4.2* podemos ver un ejemplo de resultados obtenidos usando *Kibana*. Gracias a esta herramienta conseguimos un resumen compacto y completo de los programas instalados, su versión actual y la última versión disponible. Además, nos muestra también el parámetro *<updated>*, el cual nos indica si el software consultado está actualizado o no sin necesidad de analizar primero el número de versión para compararla. La información obtenida se encuentra organizada en todo momento, existiendo también la posibilidad de ordenarla por *@timestamp* (marca temporal) o mediante cualquiera de las variables monitorizadas que encontramos en el margen izquierdo de la sección *Discover* de *Kibana*.

Las pruebas relacionadas con la API pueden realizarse mediante el software *Postman* anteriormente comentado o mediante *CURL* desde un dispositivo con conexión a Internet.

La *Figura 4.3* muestra la realización de una petición para obtener la lista de los programas que ofrece nuestra API con sus respectivas últimas versiones. Para ello, se ha hecho una petición HTTPS realizando una petición a la URL *getall.* En este caso, al ser una petición de tipo GET, no hace falta introducir nada en el apartado *body*.

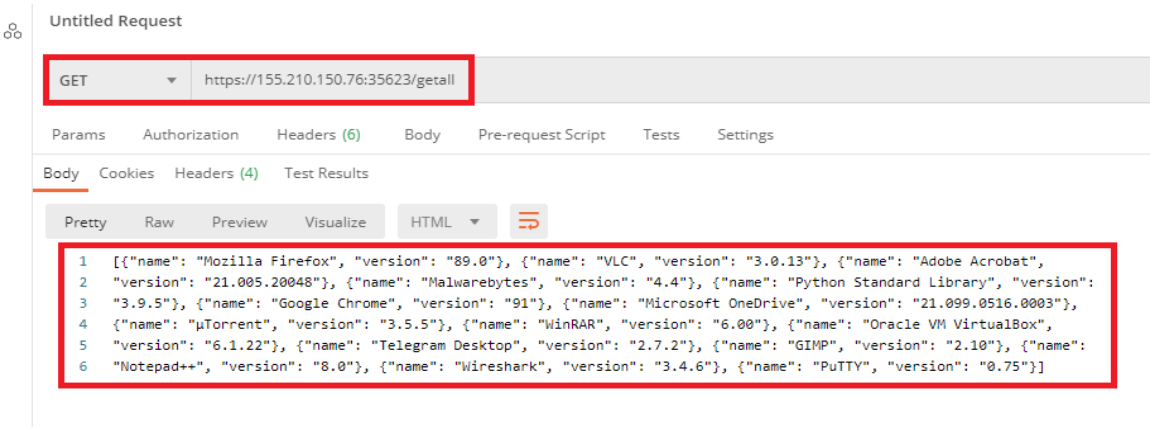

Figura 4.3: Ejemplo petición a la URL getall

Para el segundo caso, tal y como muestra la *Figura 4.4* se ha realizado una petición HTTPS también de tipo GET, pero en este caso realizándola URL *getprograms.* Su fin es obtener únicamente la lista de los programas con lo que nuestra API trabaja y están disponibles para que hacer consultas. En este caso tampoco ha sido necesario el uso de apartado *body.*

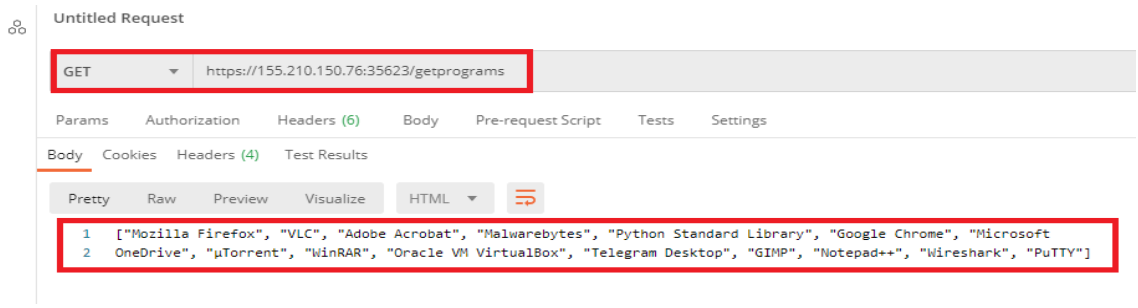

Figura 4.4: Ejemplo de petición a la URL getprograms

En la *Figura 4.5* se ejemplifica una petición HTTPS POST esta vez realizando una petición a la URL *lastversion.* Con este método lo que conseguimos es que la API nos devuelva como resultado el nombre y la última versión del software o conjunto de softwares por los que hemos preguntado. Ese programa o programas de los cuales queremos hacer la consulta debemos introducirlos en el apartado *body* de nuestra query.

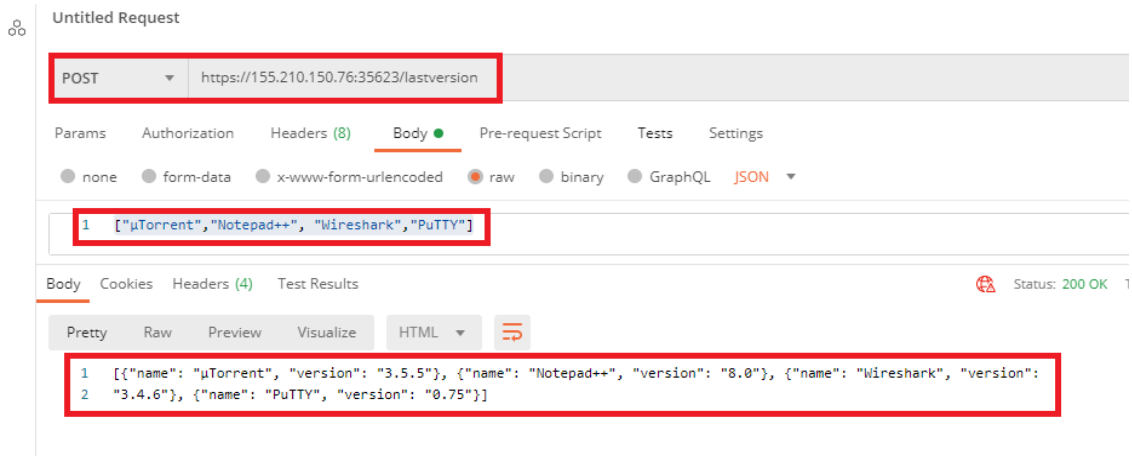

Figura 4.5: Ejemplo de petición a la URL lastversion

Por último, en la *Figura 4.6* podemos ver el ejemplo de una petición HTTPS POST realizando una petición a la URL *updated.* El fin de esta query es averiguar si el software o conjunto de softwares por los que estamos preguntando están actualizados. Para ello, deberemos introducir manualmente en la sección *body* de la query el nombre y la versión que tenemos en ese momento del software a consultar.

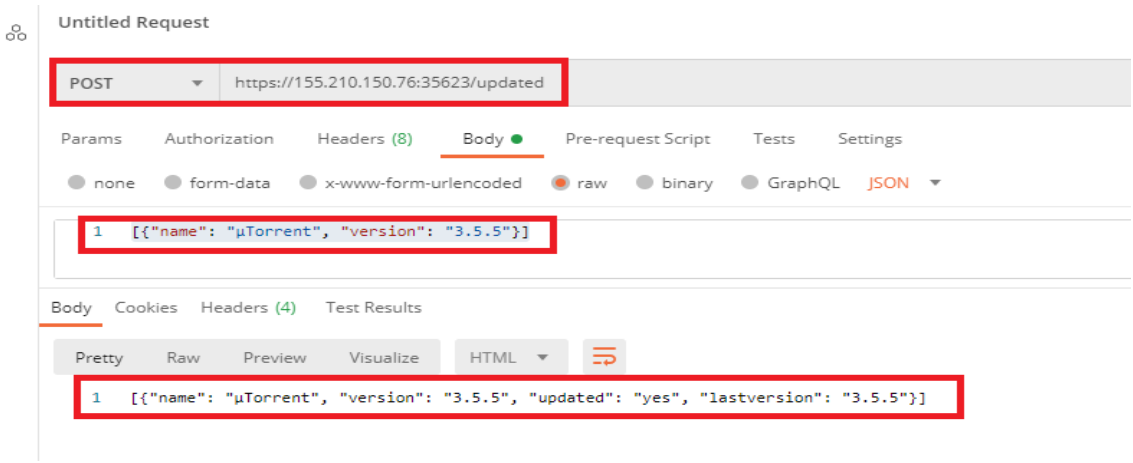

Figura 4.6: Ejemplo de petición a la URL updated

La idea de este modo de funcionamiento basado en la API es ofrecer a los desarrolladores la posibilidad de utilizar la API para su propio consumo mediante la programación de las peticiones por ellos mismos. En nuestro caso en particular se han hecho mediante *Postman* pero se podría usar cualquier otra forma.

Capítulo 5

Conclusiones y líneas futuras

5.1. Conclusiones

En este proyecto se ha desarrollado un sistema para comparar las versiones de software que tiene instaladas el usuario con la más actual disponible. Tal y como se ha explicado en la sección *3.1*, distinguimos dos modos de funcionamiento de nuestro sistema. El primero, más enfocado a empresas, con el que podemos realizar una monitorización del software instalado mediante *Kibana*, y un segundo, más enfocado a desarrolladores, con el que cualquier persona que se ciña al formato de envío de las peticiones puede realizar consultas a nuestra API para obtener la información deseada. En este segundo modo el usuario ha podido obtener los datos para realizar la consulta mediante *osquery* o no.

Se ha desarrollado una API ubicada en una MV Debian que permite al usuario consultar en cualquier momento y beneficiarse de sus prestaciones, obteniendo así la información que este desee sobre las versiones de software. Por otra parte, se han añadido certificados y se ha utilizado el protocolo SSL/TLS, añadiendo así una capa de seguridad al sistema.

Durante el desarrollo de este Trabajo Fin de Grado se han adquirido conocimientos sobre las herramientas *Elastic Stack, osquery* y el software *Postman*. Además, se han ampliado los conocimientos tanto de programación en Python como de administración de sistemas.

La importancia de la seguridad en sistemas crece cada día más, no sólo a nivel empresarial sino también a nivel de usuario. La finalidad de este proyecto ha sido darle al usuario la posibilidad de evitar una futura explotación de vulnerabilidades en su sistema.

5.2. Líneas futuras

A continuación, se presentan una serie de mejoras del trabajo realizado que pueden resultar interesantes de cara al futuro.

- Integración de una cantidad más elevada de programas obtenidos mediante web scraping, pues, la limitación temporal y de recursos nos ha llevado a limitar el número de software a analizar.

- Obtención de softwares de uso común de otros SO (Linux, macOS, etc) para su posterior comparación de versiones.

- Programación de una nueva query en la API en la que se devuelva al usuario el enlace web para la descarga de la última versión de un programa o programas si estos no están actualizados.

- Sustituir los certificados actuales por unos firmados por una autoridad de certificación verificada ya que, actualmente, los certificados son autoafirmados debido a que es más que suficiente para este proyecto, pero no lo sería para un entorno de producción más complejo.

- Creación de alertas en *Kibana* que avisen al usuario cuando su versión instalada esté desactualizada, es decir, sea inferior a la obtenida mediante web scraping.

Capítulo 6

Referencias

[1] Documentación Elastic Stack

https://www.elastic.co/guide/en/elastic-stack-get-started/6.8/get-started-elastic-stack.html#logstash-setup

https://www.elastic.co/guide/en/beats/filebeat/current/logstash-output.html

https://www.elastic.co/guide/en/logstash/current/config-examples.html

https://www.elastic.co/guide/en/elastic-stack-get-started/7.10/get-started-elastic-stack.html

[2] Librería web scraping

https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[3] Documentación osquery

https://osquery.readthedocs.io/en/stable/introduction/using-osqueryi/

[4] Librería Flask para API

https://blog.miguelgrinberg.com/post/running-your-flask-application-over-https

[5] Documentación para la creación de certificados

https://benjaminknofe.com/blog/2018/07/08/logstash-authentication-with-ssl-certificates/

[6] Documentación para la aplicación del protocolo SSL/TLS

https://github.com/Busindre/How-to-configure-SSL-for-FileBeat-and-Logstash-step-by-step

Capítulo 7

Anexos

A.1. Archivo configuración osquery

El archivo que se presenta a continuación corresponde a la configuración del framework *osquery*. Dicho archivo será necesario a la hora de recolectar la información del equipo local con el fin de saber qué programas tiene instalados el usuario en su equipo. Para ello, los dos puntos clave a configurar en este archivo son las queries que queremos realizarle al sistema y cada cuánto tiempo queremos que se realicen. En este caso, ha sido posible realizarlo gracias a los parámetros *query* e *interval* que podemos ver en el código a continuación.

```
1     {
2       //Configure the daemon below:
3       "options": {
4
5         // Select the osquery config plugin.
6         "config_plugin": "filesystem",
7
8         // Select the osquery logging plugin.
9         "logger_plugin": "filesystem",
10
11
12
13        "disable_events": "false",
14
15        "enable_windows_events_publisher": "true",
16        "enable_windows_events_subscriber": "true",
17        "enable_ntfs_event_publisher": "true",
18        "enable_powershell_events_subscriber": "true",
19
20        "utc": "true"
21      },
22
23      //Define a schedule of queries:
24            "schedule": {
25            },
26
27      //Decorators are normal queries that append data to every
28        query.
29            "decorators": {
30            },
31
32            "packs": {
33              "system-snapshot": {
34                "queries": {
35                  "some_query1": {
36                      "query": "SELECT name,version FROM
37                               programs;SELECT name,version
38                               FROM os_version;",
39                      "snapshot": true,
40                      "interval": 84600
```

```
41            }
42          }
43        }
44
45      },
46
47      "feature_vectors": {
48      "character_frequencies": [
49           0.0,       0.0,       0.0,       0.0,        0.0,       0.0,
50    0.0,
51           0.0,       0.0,       0.0,       0.0,        0.0,       0.0,
52    0.0,
53           0.0,       0.0,       0.0,       0.0,        0.0,       0.0,
54    0.0,
55           0.0,       0.0,       0.0,       0.0,        0.0,       0.0,
56    0.0,
57           0.0,        0.0,          0.0,          0.0,           0.0,
58    0.00045,  0.01798,
59           0.0,             0.03111,    0.00063,    0.00027,       0.0,
60    0.01336,  0.0133,
61           0.00128,         0.0027,    0.00655,   0.01932,    0.01917,
62    0.00432,  0.0045,
63           0.00316,   0.00245,     0.00133,     0.001029,     0.00114,
64    0.000869, 0.00067,
65           0.000759,  0.00061,    0.00483,    0.0023,         0.00185,
66    0.01342,  0.00196,
67           0.00035,    0.00092,     0.027875,  0.007465,     0.016265,
68    0.013995, 0.0490895,
69           0.00848,    0.00771,    0.00737,    0.025615,    0.001725,
70    0.002265, 0.017875,
71           0.016005,   0.02533,     0.025295,  0.014375,     0.00109,
72    0.02732,  0.02658,
73           0.037355,  0.011575,  0.00451,     0.005865,     0.003255,
74    0.005965, 0.00077,
75           0.00621,  0.00222,    0.0062,      0.0,             0.00538,
76    0.00122,  0.027875,
77           0.007465,    0.016265,    0.013995,    0.0490895,    0.00848,
78    0.00771,  0.00737,
79           0.025615,   0.001725,   0.002265,   0.017875,      0.016005,
80    0.02533,  0.025295,
81           0.014375,  0.00109,    0.02732,    0.02658,       0.037355,
82    0.011575, 0.00451,
83           0.005865,   0.003255,   0.005965,   0.00077,        0.00771,
84    0.002379, 0.00766,
85           0.0,       0.0,       0.0,       0.0,        0.0,       0.0,
86    0.0,
87           0.0,       0.0,       0.0,       0.0,        0.0,       0.0,
88    0.0,
89           0.0,       0.0,       0.0,       0.0,        0.0,       0.0,
90    0.0,
91           0.0,       0.0,       0.0,       0.0,        0.0,       0.0,
92    0.0,
93           0.0,       0.0,       0.0,       0.0,        0.0,       0.0,
94    0.0,
95           0.0,       0.0,       0.0,       0.0,        0.0,       0.0,
96    0.0,
97           0.0,       0.0,       0.0,       0.0,        0.0,       0.0,
98    0.0,
99           0.0,       0.0,       0.0,       0.0,        0.0,       0.0,
100   0.0,
```

```
101         0.0,        0.0,        0.0,        0.0,        0.0,        0.0,
102   0.0,
103         0.0,        0.0,        0.0,        0.0,        0.0,        0.0,
104   0.0,
105         0.0,        0.0,        0.0,        0.0,        0.0,        0.0,
106   0.0,
107         0.0,        0.0,        0.0,        0.0,        0.0,        0.0,
108   0.0,
109         0.0,        0.0,        0.0,        0.0,        0.0,        0.0,
110   0.0,
111         0.0,        0.0,        0.0,        0.0,        0.0,        0.0,
112   0.0,
113         0.0,        0.0,        0.0,        0.0,        0.0,        0.0,
114   0.0,
115         0.0,        0.0,        0.0,        0.0,        0.0,        0.0,
116   0.0,
117         0.0,        0.0,        0.0,        0.0,        0.0,        0.0,
118   0.0,
119         0.0,        0.0,        0.0,        0.0,        0.0,        0.0,
120   0.0,
121         0.0,       0.0,       0.0
122       ]
123     }
124   }
```

Listing A.1.1: Configuración de osquery

Cabe destacar la opción *snapshot* habilitada a *"true"*. Esto representa el modo en el que se recolectarán los datos. Se ha elegido el modo *snapshot* ya que este modo creaba una nueva línea en el archivo *osqueryd.snapshots* con los programas actualmente instalados en ese momento cada vez que el framework efectúa las queries, en nuestro caso, una vez al día.

A.2. Scripts de Python

A.2.1. elastic.py

```
1     from elasticsearch import Elasticsearch
2     from project import scraping
3     import json
4
5     elastic_ip = "localhost"
6     elastic_port = 9200
7     listofVersions = scraping.getAllVersions()
8     es = Elasticsearch(host=elastic_ip, port=elastic_port,
9     http_auth=('elastic', 'elastic'))
10    _index = "scraping_programs"
11    _type = "sversions"
12    _index2 = "client_programs"
13
14    #creamos indice con datos de scraping
15    es.indices.create(index=_index, ignore=400)
16
17    #creamos indice con datos de logstash para kibana
18    es.indices.create(index=_index2, ignore=400)
19
20    i = 1
21    for ver in listofVersions:
22        _body = json.dumps(ver.__dict__)
23        res = es.index(index=_index, doc_type=_type, id=i
          body=_body)
25
27        i = i + 1
```

Listing A.2.1: Script elastic.py

A.2.2. scraping.py

```
1     import requests
2     from bs4 import BeautifulSoup
3
4
5     class Program:
6         def __init__(self, _name, _version):
7             self.name = _name
8             self.version = _version
9
10    class Full_Program:
11        def __init__(self, _name, _version,_updated,_lastversion):
12            self.name = _name
13            self.version = _version
14            self.updated = _updated
15            self.lastversion = _lastversion
```

```
16    #MOZILLA FIREFOX
17
18    def getFirefoxVersion():
19        url1 = "https://www.mozilla.org/en-US/firefox/releases/"
20        response = requests.get(url1)
21        soup = BeautifulSoup(response.text, 'html.parser')
22        quotes_html = soup.find_all("a")
23        result = ""
24        i = 0
25        for q in quotes_html:
26            if("releasenotes" in str(q.get("href")) and i == 0):
27                result = str(q.get("href")).split("/")[2]
28                i = 1
29        return result
30
31
32    #ADOBE ACROBAT READER
33
34    def getAdobeVersion():
35        url2 = "https://helpx.adobe.com/es/acrobat/release-
36        note/release-notes-acrobat-reader.html"
37        response = requests.get(url2)
38        soup = BeautifulSoup(response.text, 'html.parser')
39        quotes_html = soup.find_all("a")
40        result = ""
41        i = 0
42        for q in quotes_html:
43            if("ReleaseNotesDC" in str(q.get("href"))):
44                if(i == 1):
45                    result = q.get_text().split("
46                    ")[1].strip("(").strip(")")
47                i = i+1
48        return result
49
50
51    #VLC
52
53    def getVLCversion():
54        url3 = "https://www.videolan.org/vlc/releases/"
55        response = requests.get(url3)
56        soup = BeautifulSoup(response.text, 'html.parser')
57        quotes_html = soup.find_all("a")
58        result = ""
59        i = 0
60        for q in quotes_html:
61            if("releases" in str(q.get("href")) and i == 0):
62                result = q.get_text().split(" ")[1]
63                i = 1
64        return result
```

```
65    #MALWAREBYTES
66
67    def getMalwarebytesVersion():
68        url4 = "https://support.malwarebytes.com/hc/en-
69        us/sections/360005863113-Release-History-News"
70        response = requests.get(url4)
71        soup = BeautifulSoup(response.text, 'html.parser')
72        quotes_html = soup.find_all("a")
73        result = ""
74        i = 0
75        for q in quotes_html:
76            if("Release-Notes" in str(q.get("href")) and i == 0):
77                result = q.get_text().split(" ")[3]
78                i = 1
79        return result
80
81
82    #PYTHON STANDARD LIBRARY
83
84    def getPythonVersion():
85        url5 = "https://www.python.org/doc/versions/"
86        response = requests.get(url5)
87        soup = BeautifulSoup(response.text, 'html.parser')
88        quotes_html = soup.find_all("a")
89        result = ""
90        i = 0
91        for q in quotes_html:
92            if("release" in str(q.get("href")) and i == 0):
93                result = q.get_text().split(" ")[1]
94                i = 1
95        return result
96
97
98    #GOOGLE CHROME
99
100   def getChromeVersion():
101       url6 =
102       "https://support.google.com/chrome/a/answer/7679408?hl
103          =en#zippy="
104       response = requests.get(url6)
105       soup = BeautifulSoup(response.text, 'html.parser')
106       quotes_html = soup.find_all("a")
107       result = ""
108       i = 0
109       for q in quotes_html:
110           if("/chrome/a/answer/" in str(q.get("href")) and
111              "noopener" in str(q.get("rel"))):
112                   if (i == 3):
113                       result = q.get_text().split(" ")[1]
114                       result = result.split(":")[0]
115                   i = i+1
116       return result
```

```
117  #MICROSOFT ONEDRIVE
118
119  def getOneDriveVersion():
120      url7 = "https://support.microsoft.com/en-us/office/onedrive-
121      release-notes-845dcf18-f921-435e-bf28-4e24b95e5fc0"
122      response = requests.get(url7)
123      soup = BeautifulSoup(response.text, 'html.parser')
124      quotes_html = soup.find_all("div")
125      result = ""
126      i = 0
127      for q in quotes_html:
128          if("ocpExpandoHeadTitleContainer" in str(q.get("class"))
129             and i == 0):
130                 result = q.get_text().split(" ")[1]
131                 i = 1
132      return result
133
134
135  #  µTORRENT
136
137  def getuTorrentVersion():
138      url8 = "https://blog.utorrent.com/releases/windows/"
139      response = requests.get(url8,headers={
140      "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X
141      10_11_2) AppleWebKit/537.36 (KHTML, like Gecko)
142      Chrome/47.0.2526.106 Safari/537.36"
143      })
144      soup = BeautifulSoup(response.text, 'html.parser')
145      quotes_html = soup.find_all("a")
146      result = ""
147      i = 0
148      for q in quotes_html:
149          if("release-notes" in str(q.get("href")) and i == 0):
150              result = q.get_text().split(" ")[1]
151              i = 1
152      return result
153
154
155  #WINRAR
156
157  def getWinrarVersion():
158      url9 = "https://www.win-rar.com/latestnews.html"
159      response = requests.get(url9)
160      soup = BeautifulSoup(response.text, 'html.parser')
161      quotes_html = soup.find_all("a")
162      result = ""
163      i = 0
164      for q in quotes_html:
165          if("singlenewsview" in str(q.get("href")) and "Final" in
166             str(q.get("title")) and  i == 0):
167              result = q.get_text().split(" ")[1]
168              i = 1
169      return result
```

```
170   #VIRTUALBOX
171
172   def getVirtualBoxVersion():
173       url10 = "https://www.virtualbox.org/wiki/Changelog"
174       response = requests.get(url10)
175       soup = BeautifulSoup(response.text, 'html.parser')
176       quotes_html = soup.find_all("strong")
177       result = ""
178       i = 0
179       for q in quotes_html:
180           if(i == 1):
181               result = q.get_text().split(" ")[1]
182           i = i+1
183       return result
184
185
186   #TELEGRAM DESKTOP
187
188   def getTelegramVersion():
189       url11 = "https://desktop.telegram.org/changelog"
190       response = requests.get(url11)
191       soup = BeautifulSoup(response.text, 'html.parser')
192       quotes_html = soup.find_all("h3")
193       result = quotes_html[0].get_text().split(" ")[1]
194
195       return result
196
197
198   #GIMP
199
200   def getGimpVersion():
201       url12 = "https://www.gimp.org/release-notes/"
202       response = requests.get(url12)
203       soup = BeautifulSoup(response.text, 'html.parser')
204       quotes_html = soup.find_all("a")
205       result = ""
206       i = 0
207       for q in quotes_html:
208           if("gimp-" in str(q.get("href")) and i == 0 ):
209               result = q.get_text().split(" ")[-1]
210               i = 1
211
212       return result
213
214
215   #NOTEPAD++
216
217   def getNotepadVersion():
218       url13 = "https://notepad-plus-plus.org/news/"
219       response = requests.get(url13)
220       soup = BeautifulSoup(response.text, 'html.parser')
221       quotes_html = soup.find_all("a")
222       result = ""
223       i = 0
224       for q in quotes_html:
225           if ("/downloads/" in str(q.get("href")) and i == 0):
226               result = q.get_text().split(" ")[-1]
227               i = 1
228       return result
```

```
#WIRESHARK

def getWiresharkVersion():
    url14 = "https://www.wireshark.org/docs/relnotes/"
    response = requests.get(url14)
    soup = BeautifulSoup(response.text, 'html.parser')
    quotes_html = soup.find_all("a")
    result = ""
    i = 0
    for q in quotes_html:
        if ("wireshark-" in str(q.get("href")) and i == 0):
            result = q.get_text().split(" ")[-1]
            i = 1
    return result


#PUTTY

def getPuTTYversion():
    url16 =
      "https://www.chiark.greenend.org.uk/~sgtatham/
      putty/changes.html"
    response = requests.get(url16)
    soup = BeautifulSoup(response.text, 'html.parser')
    quotes_html = soup.find_all("a")
    result = ""
    i = 0
    for q in quotes_html:
        if("releases" in str(q.get("href")) and i == 0):
            result = q.get_text()
            i = 1
   return result



def getAllVersions():

    listofVersions_ = []
    r1 = getFirefoxVersion()
    r2 = getVLCversion()
    r3 = getAdobeVersion()
    r4 = getMalwarebytesVersion()
    r5 = getPythonVersion()
    r6 = getChromeVersion()
    r7 = getOneDriveVersion()
    r8 = getuTorrentVersion()
    r9 = getWinrarVersion()
    r10 = getVirtualBoxVersion()
    r11 = getTelegramVersion()
    r12 = getGimpVersion()
    r13 = getNotepadVersion()
    r14 = getWiresharkVersion()
    r15 = getPuTTYversion()
```

```
282     listofVersions_.append(Program("Mozilla Firefox",str(r1)))
283     listofVersions_.append(Program("VLC",str(r2)))
284     listofVersions_.append(Program("Adobe Acrobat",str(r3)))
285     listofVersions_.append(Program("Malwarebytes",str(r4)))
286     listofVersions_.append(Program("Python Standard
287     Library",str(r5)))
288     listofVersions_.append(Program("Google Chrome",str(r6)))
289     listofVersions_.append(Program("Microsoft OneDrive",str(r7)))
290     listofVersions_.append(Program("  Torrent",str(r8)))
291     listofVersions_.append(Program("WinRAR",str(r9)))
292     listofVersions_.append(Program("Oracle VM
293     VirtualBox",str(r10)))
294     listofVersions_.append(Program("Telegram Desktop", str(r11)))
295     listofVersions_.append(Program("GIMP", str(r12)))
296     listofVersions_.append(Program("Notepad++", str(r13)))
297     listofVersions_.append(Program("Wireshark", str(r14)))
298     listofVersions_.append(Program("PuTTY", str(r15)))
299
300
301
302     return listofVersions_
```

Listing A.2.2: Script scraping.py

A.2.3. api.py

```
1     import json
2     from project import scraping
3     from flask import Flask, request, jsonify
4     from elasticsearch import Elasticsearch
5     app = Flask(__name__)
6
7     elastic_ip = "localhost"
8     elastic_port = 9200
9     api_ip = "api.tfg"
10    api_port = 443
11    cert = "/home/marina/tfg/ficherosconf/certificates/cert.pem"
12    key = "/home/marina/tfg/ficherosconf/certificates/key.pem"
13    index = "scraping_programs"
14    es = Elasticsearch(host=elastic_ip, port=elastic_port,
15    http_auth=('elastic', 'elastic'))
16
17    #Leemos del elastic ya que ahi tenemos guardadas las ultimas
18    versiones de nuestros programas
19
20    @app.route("/getall",methods=['GET'])
21    def getAllPrograms():
22        try:
23          listofprograms_ = []
24          res = es.search(index=index, doc_type="sversions", body={
25                'size': 100,
26                'query': {
27                    'match_all': {}
28                }
29            })
```

```
30          for obj in res['hits']['hits']:
31                name = obj['_source']['name']
32                version = obj['_source']['version']
33                listofprograms_.append(scraping.Program(name,
34                version))
35            response = json.dumps([ob.__dict__ for ob in
36                       listofprograms_], ensure_ascii=False)
37            return response
38        except:
39            contenido = {
40               "value": "Hubo un error en la solicitud"
41            }
42            jresponse = jsonify(contenido)
43            jresponse.status_code = 400
44            return jresponse
45
46    @app.route("/getprograms",methods=['GET'])
47    def getProgramsName():
48        try:
49            res = es.search(index=index, doc_type="sversions",
50            body={'size': 100,'query': {'match_all': {}}
51            })
52            listofprograms_ = []
53            for obj in res['hits']['hits']:
54                name = obj['_source']['name']
55                listofprograms_.append(scraping.Program(name, ""))
56            response = json.dumps([ob.__dict__.get('name') for ob in
57            listofprograms_], ensure_ascii=False)
58            return response
59        except:
60            contenido = {
61                "value": "Hubo un error en la solicitud"
62            }
63            jresponse = jsonify(contenido)
64            jresponse.status_code = 400
65            return jresponse
66
67    @app.route("/lastversion",methods=['POST'])
68    def getLastVersion():
69        try:
70            data = request.json
71            listofprograms_ = []
72            for prog in data:
73                pr = prog
74                res = es.search(index=index, doc_type="sversions",
75                body={'size': 100,'query': {'match': {'name': prog}}
76                })
77                if (len(res['hits']['hits']) != 0):
78                    name = res['hits']['hits'][0]['_source']['name']
79                    version =
80                    res['hits']['hits'][0]['_source']['version']
81                else:
82                    name = prog
83                    version = "no version found"
84                listofprograms_.append(scraping.Program(name,
85                version))
86
87            response = json.dumps([ob.__dict__ for ob in
88            listofprograms_], ensure_ascii=False)
89            return response
90        except:
```

```
91            contenido = {
92                "value": "Hubo un error en la solicitud"
93            }
94            jresponse = jsonify(contenido)
95            jresponse.status_code = 400
96            return jresponse
97
98    @app.route("/updated",methods=['POST'])
99    def getUpdated():
100       try:
101           data = request.json
103           listofprograms_ = []
104           for prog in data:
105               _name1 = prog['name']
106               res = es.search(index=index, doc_type="sversions",
107               body={'size': 100,'query': {
108                     'match': {'name': _name1}}})
109
119               if (len(res['hits']['hits']) != 0):
120                   name = res['hits']['hits'][0]['_source']['name']
121                   lversion =
122                   res['hits']['hits'][0]['_source']['version']
123                   version = prog['version']
124                   updated = "no"
125                   if prog['version'] == lversion:
126                       updated = "yes"
127               else:
128                   name = prog['name']
129                   version = "no version found"
130                   updated = "no"
131                   lversion = "no version found"
132               listofprograms_.append(scraping.Full_Program(name,
133               version, updated, lversion))
134           response = json.dumps([ob.__dict__ for ob in
135           listofprograms_], ensure_ascii=False)
136           return response
137       except:
138           contenido = {
139               "value": "Hubo un error en la solicitud"
140           }
141           jresponse = jsonify(contenido)
142           jresponse.status_code = 400
143           return jresponse
144
145   app.run(host=api_ip, port=api_port, debug=True,
146   ssl_context=(cert,key))
```

Listing A.2.3: Script api.py

A.3. Archivo configuración Filebeat

En este archivo los únicos apartados a configurar serán los apartados *filebeat.input* y *output.logstash*. Recordemos que este *Beat* se encarga de recolectar los datos que a su vez ha recolectado *osquery* y enviarlos a *Logstash* para su procesado. Cabe recordar que *Filebeat* está corriendo como demonio en el equipo recolector de los datos del usuario, al igual que osquery.

En *Listing A.3.1* hemos indicado el path del archivo en el que se guarda la información obtenida por *osquery* (osqueryd.snapshots).

```
1     filebeat.inputs:
2
3     # Each - is an input. Most options can be set at the input
4     #level, so you can use different inputs for various
5     #configurations.
6
7     # Below are the input specific configurations.
8
9     - type: log
10
11      # Change to true to enable this input configuration.
12      # enabled: true
13
14      # Paths that should be crawled and fetched. Glob based
15      # paths.
16
17      paths:
19        -C:\\Program Files\\osquery\\log\\osqueryd.snapshots.log
```

Listing A.3.1: Configuración del input de Filebeat

En la siguiente imagen, correspondiente a la configuración de la salida hacia *Logstash,* hemos tenido que configurar una serie de parámetros. En primer lugar, con el parámetro *hosts* indicamos la IP y el puerto en el que está ubicado *Logstash. logstash.tfg* corresponde a la IP del servidor de la universidad, únicamente hemos hecho un mapeo en el fichero *hosts* de Windows para asignarle a dicha IP el nombre *logstash.tfg.* Esto se ha realizado simplemente para casos futuros en lo que la IP pudiera cambiar, simplemente deberíamos ir al fichero *hosts* y cambiarla desde ahí directamente en lugar de ir cambiando individualmente cada archivo de configuración en los que aparece. Además, para la dotar al sistema de una capa de seguridad, hemos creado una serie de certificados y una clave con el fin de usar el protocolo SSL/TLS. Sólo queda indicar en el archivo de configuración dónde se encuentran dichos certificados y dicha clave.

```
output.logstash:
  # The Logstash hosts
  hosts: ["logstash.tfg:5044"]

  # Optional SSL. By default is off.
  # List of root certificates for HTTPS server verifications
  ssl.certificate_authorities:
  ["C:\\Users\\Marina\\Desktop\\certificates\\ca.crt"]

  # Certificate for SSL client authentication
  ssl.certificate:
  "C:\\Users\\Marina\\Desktop\\certificates\\beat.crt"

  # Client Certificate Key
  ssl.key: "C:\\Users\\Marina\\Desktop\\certificates\\beat.key"
```

Listing A.3.2: Configuración del output de Filebeat

A.4. Archivo configuración Logstash

El archivo de configuración presentado a continuación es necesario para recolectar los datos procedentes de *Filebeat*, procesarlos, y enviarlos a *Elasticsearch.* Como ya se ha especificado en la sección *3.4.2*, *Logstash* consta de tres bloques: entrada, filtrado y salida. Este archivo estará alojado en el servidor de la Universidad. A continuación, se presentan los bloques desarrollados durante este proyecto.

En primer lugar, tenemos la entrada o input, en la cual, como se ha comentado en el apartado *3.4.2*, debemos configurar de dónde va a leer los datos nuestros *Logstash.* Además, dada la búsqueda de la seguridad en este proyecto, se han creado dos certificados (uno de CA y otro para el propio *Logstash)* y una clave para hacer uso del protocolo SSL/TLS.

```
input {
  beats {
    host => "logstash.tfg"
    port => 5044
    ssl => true

    ssl_certificate_authorities                                    =>
    ["/home/marina/tfg/ficherosconf/certificates/ca.crt"]

    ssl_certificate                                               =>
    "/home/marina/tfg/ficherosconf/certificates/logstash.crt"

    ssl_key                                                        =>
    "/home/marina/tfg/ficherosconf/certificates/logstash.key"

    ssl_verify_mode => "force_peer"
    codec => "json"
  }
}
```

Listing A.4.1: Configuración del input de Logstash

Al igual que el caso de *Filebeat*, tenemos mapeado en el archivo ubicado en el directorio */etc/hosts* el nombre *"logstash.tfg"* con la dirección IP para simplificar el fichero de configuración consiguiendo así una mayor flexibilidad.

Siguiendo con el archivo de configuración de *Logstash,* nos encontramos con el apartado *"filter"*, en el cual se produce el procesado de los datos, la parte más importante en nuestro caso.

```
1     filter {
2       split{ field => "snapshot" }
3       mutate { add_field => { "[name_aux]" => "%{[snapshot][name]}"
4     }}
5       mutate { split => { "[name_aux]" => " " }
6                add_field       =>       {"[comparator_name]"       =>
7     "%{[name_aux][0]}"}}

8       elasticsearch {
9           user => elastic
10          password => elastic
11          index => "scraping_programs"
12          hosts => [ "http://elastic.tfg:9200"]
13          query => 'name:%{[comparator_name]}'
14          enable_sort => false
15          fields =>{"[version]" => "[latest_version]"
16                    "[name]" => "[py_name]"}
17      }
18      if([latest_version] and [py_name] in [snapshot][name] ){
19
20        mutate { remove_field => [ "@version","agent", "log", 20
21        "counter", "ecs" , "unixTime", "action" , "input" ,
22        "name","numerics", "epoch","tags","comparator_name",
23        "py_name","calendarTime","hostIdentifier" ]}
24
25        mutate { add_field => { "[name]" => "%{[snapshot][name]}" }}
26
27        mutate     {     add_field     =>     {     "[name_aux]"     =>
28        "%{[snapshot][name]}" }}
29
30        mutate     {     add_field     =>     {     "[version]"     =>
31        "%{[snapshot][version]}" }}
32
33        mutate { remove_field => ["snapshot","name_aux"] }
34
35        if ([latest_version] in [version]){
36             mutate { add_field => { "[updated]" => "yes" }}
37        }
38        else {
39             mutate { add_field => { "[updated]" => "no" }}
40        }
41      }
42      else{
43        drop{}
44      }
45
46    }
```

Listing A.4.2: Configuración del filter de Logstash

En esta parte de procesado lo que hacemos es separar cada uno de los campos que aparecen en *snapshot*, procedente de *Filebeat.* Guardamos el nombre del programa en la variable *comparator_name* y lo utilizamos para sacar la información del índice *scraping_programs* (última versión del programa) gracias al módulo *elasticsearch* ya definido en el apartado *3.4.2* de la memoria. Con la versión del programa ya en la variable *latest_version* lo que hacemos es borrar la información que no nos interesa, y, por otro lado, comparar la versión actual del programa con la versión que devuelve web scraping (es decir, comparar el campo "*version"* con *"latest_version").* Si ambos campos coinciden añadiremos al campo *"updated"* el valor *"yes"*. De lo contrario, el valor a añadir en ese campo será *"no".*

Por último, atendiendo al apartado de salida u output, el objetivo será enviar toda esa información que hemos procesado a la base de datos *Elasticsearch* y escribir así en el índice *client_programs.* Para ello, se ha realizado el mismo proceso que en los otros archivos y se ha añadido el nombre *"elastic.tfg"* al fichero *hosts* del servidor. Cabe destacar también, que se ha añadido seguridad mediante el uso de credenciales, en este caso usuario y contraseña.

```
1     output {
2       elasticsearch{
3             user => elastic
4             password => elastic
5             hosts => [ "http://elastic.tfg:9200"]
6             index => "client_programs"
7       }
8       stdout {codec => rubydebug}
9     }
```

Listing A.4.3: Configuración del output de Logstash

A.5. Guía de instalación Elastic Stack

Debemos prestar una atención especial a esta instalación ya que las herramientas de *Elastic Stack* deben comunicarse correctamente entre sí para poder efectuar la monitorización de datos correctamente. A lo largo de este anexo iremos desarrollando cada una de las instalaciones de las distintas herramientas.

Asimismo, cabe destacar que dicha instalación se ha realizado de forma remota, por lo que hemos habilitado una VPN que nos proporcionará acceso a la red de la Universidad. Para ello, se ha configurado una VPN al router del laboratorio que creará un túnel PPP *(Point to Point Tunneling Protocol)* con una dirección de rango 192.168.100.0/24. Nos conectaremos a la máquina donde vamos a instalar todo creada dentro de este servidor mediante el protocolo SSH. Este protocolo permite el acceso remoto a un servidor por medio de un canal seguro en el que toda la información está cifrada. El comando que utilizaremos para ello será el siguiente:

```
ssh root@192.168.100.101 -i /path/claveSSH
```

A continuación, veremos el proceso de instalación llevado a cabo de cada una de las herramientas de *Elastic Stack.*

A.5.1. Instalación y configuración Filebeat

1. Para empezar, instalaremos *Filebeat* en nuestro propio equipo que será el encargado de recopilar la información. Para ello, nos dirigiremos a la página (https://www.elastic.co/es/downloads/beats/metricbeat) de descargas de *Elastic* y descargaremos el zip para su instalación.

2. Extraeremos su contenido en *C:\Program Files* con el nombre de *Filebeat.*

3. Continuaremos abriendo el *Powershell* de Windows como administrador y nos posicionaremos en la carpeta donde hemos extraído la información.

4. Instalaremos *Filebeat* como un servicio de Windows para que esté continuamente en funcionamiento:

```
PS .\install-service-filebeat.ps1
```

5. A continuación, configuraremos *Filebeat.* Para ello deberemos habilitar los módulos que necesitamos, en nuestro caso será el módulo *osquery.*

```
PS .\filebeat.exe modules enable osquery
```

6. Ahora solo nos queda inicializar el servicio para que funcione permanentemente.

```
PS Start-Service filebeat
```

A.5.2. Instalación y configuración Elasticsearch

1. En primer lugar, siempre es recomendable actualizar el sistema con los paquetes más recientes antes de comenzar cualquier instalación. Para ello, debemos usar el siguiente comando:

```
apt update && apt-get upgrade
```

2. A continuación, deberemos instalar Java-8 en el sistema, ya que es requerido por *Elasticsearch* y *Logstash* para su correcto funcionamiento. Esto lo haremos instalando el Java Developement Kit (JDK) por defecto:

```
apt install openjdk-8-jdk
```

3. Tras instalar Java, el siguiente paso es importar la clave pública de *Elasticsearch* que utilizada el método cifrado GPG:

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-
```

La respuesta debería ser OK.

4. Seguidamente, necesitamos permitir el acceso a nuestros repositorios a través de HTTPS. Para ello deberemos instalar un paquete de transporte APT:

```
sudo apt-get install apt-transport-https
```

5. El siguiente paso es agregar el repositorio de *Elastic* al directorio *sources.list.d* ubicado en */etc/apt/.* Usaremos el comando *echo* y el comando *sudo tee* para redireccionar la salida del archivo.

```
echo "deb https://artifacts.elastic.co/packages/7.x/apt stable main" | sudo
tee -a /etc/apt/sources.list.d/elastic-7.x.list
```

6. Tras todos estos pasos, ya podemos comenzar la instalación de *Elasticsearch* propiamente dicha. Para esta instalación, primero actualizaremos sus listas de paquetes y seguidamente lo instalaremos:

```
sudo apt-get update
```

```
sudo apt-get install elasticsearch
```

7. Una vez se haya completado la instalación de Elasticsearch podemos modificar su archivo de configuración para customizar su funcionamiento.

```
nano /etc/elasticsearch/elasticsearch.yml
```

8. A continuación, iniciaremos el servicio *Elasticsearch* gracias al comando *systemctl:*

```
systemctl start elasticsearch
```

También mediante el uso de este comando, podremos comprobar el estado en el que se encuentra nuestro servicio.

```
systemctl status elasticsearch
```

A.5.3. Instalación y configuración Kibana

Atendiendo a la documentación oficial, deberemos instalar *Kibana* sólo después de haber instalada *Elasticsearch*. La instalación en este orden garantiza que los componentes de los que depende cada producto estén correctamente implementados.

1. Debido a que ya hemos hecho la adición previa de los paquetes de Elastic en el *Anexo 5.2*, podremos instalar el resto de los componentes de la pila *Elastic Stack* sin más dilación:

```
sudo apt-get install kibana
```

2. Ahora deberemos realizar alguna modificación el archivo *kibana.yml* para conseguir su correcto funcionamiento. Este archivo se encuentra en el directorio */etc/kibana/* y podremos hacer las modificaciones pertinentes usando el comando *"nano".* Dichas modificaciones será, en primer lugar, cambiar el valor del parámetro *server.host.*

```
1     server.host: "kibana.tfg"
```

   En segundo lugar, añadiremos autenticación mediante el uso de credenciales (usuario y contraseña).

```
1     elasticsearch.username: "elastic"
2     elasticsearch.password: "elastic"
```

   Y, por último, habilitaremos el uso del protocolo SSL/TLS especificando los certificados y la clave que hemos creado previamente.

```
1     server.ssl.enabled: true
2
3     server.ssl.certificate:
4     "/home/marina/tfg/ficherosconf/certificates/ca.crt"
5
5     server.ssl.key:
6     "/home/marina/tfg/ficherosconf/certificates/ca.key"
```

3. A continuación, iniciaremos *Kibana.*

```
systemctl start kibana
```

4. Como hemos explicado en el apartado de instalación de *Elasticsearch* (A.4.2), podremos verificar el estado del servicio mediante el uso del comando *systemctl* también.

```
systemctl status kibana
```

De esta forma, podemos comprobar como introduciendo el nombre asignado a la IP de *Kibana (*https://kibana.tfg:5601) en cualquier navegador con acceso a la dirección, podemos acceder a esta herramienta. En nuestro caso, para poder conectarnos desde nuestro propio ordenador a *Kibana*, se utiliza una VPN y la habilitación de un reenvío de puertos (DNAT) para poder conectarnos al puerto 5601 de la máquina en la que se encuentra *Kibana.* Dicho acceso, como ya se ha comentado previamente se hará mediante usuario y contraseña.

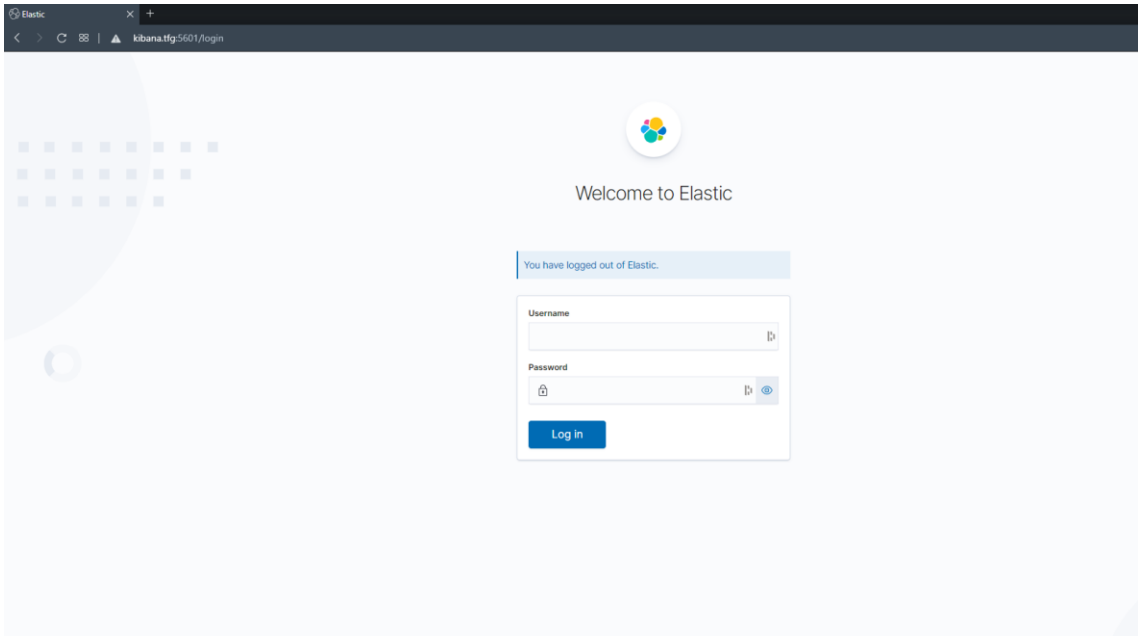

Figura A.5.1: Login inicial de Kibana

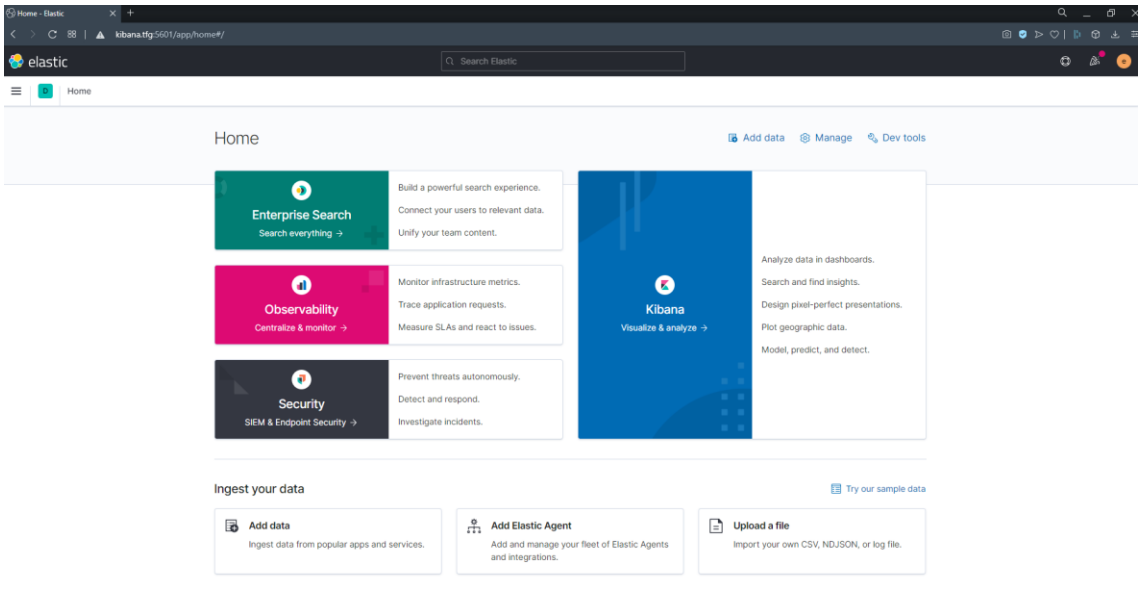

Figura A.5.2: Menú principal de Kibana

A.5.4. Instalación y configuración Logstash

1. En primer lugar, ha de actualizarse las listas de paquetes del sistema, y al igual que para la instalación de *Elasticsearch* y *Kibana*, introduciremos el siguiente comando:

```
sudo apt-get install logstash
```

El trabajo de *Logstash* es tomar datos y generar una salida gracias al procesado previo de la sección de filtrado, por lo que necesitamos archivos de configuración en *Logstash* que contentan la configuración de entrada, filtrado y salida.

2. Crearemos los archivos de configuración *(input, filter y output)* especificados en el *Anexo 4,* ubicados en el directorio */etc/logstash/conf.d*.

3. A continuación, iniciaremos el servicio mediante el siguiente comando:

```
systemctl start logstash
```

4. Por último, verificamos el estado del servicio para comprobar que está activo:

```
systemctl status logstash
```

A.6. Incorporación del protocolo SSL/TLS en la comunicación

1. En primer lugar, necesitamos crear un certificado de autoridad de certificación (CA), que más adelante firmará los certificados de *Logstash*. De esta forma, se podrá verificar si la conexión proviene de un cliente conocido. Mediante los siguientes comandos, creamos la clave y el certificado, respectivamente:

```
openssl genrsa -out ca.key 2048
```

```
openssl req -x509 -new -nodes -key ca.key -sha256 -days 3650 -out ca.crt
```

2. Seguidamente, crearemos un fichero customizado de OpenSSL (*logstash.conf)* con el contenido que especificaremos a continuación, en el que necesitaremos la IP donde se alberga nuestro *Logstash.* En nuestro caso, al haber realizado el mapeo IP-nombre en el fichero *"hosts",* tal y como se indica en el *Anexo 4*, bastará con introducir el nombre mapeado (*logstash.tfg).*

```
[req]
distinguished_name = req_distinguished_name
req_extensions = v3_req
prompt = no

[req_distinguished_name]
countryName                     = XX
stateOrProvinceName             = XXXXXX
localityName                    = XXXXXX
postalCode                      = XXXXXX
organizationName                = XXXXXX
organizationalUnitName          = XXXXXX
commonName                      = XXXXXX
emailAddress                    = XXXXXX

[v3_req]
keyUsage = keyEncipherment, dataEncipherment
extendedKeyUsage = serverAuth
subjectAltName = @alt_names

[alt_names]
DNS.1 = logstash.tfg
```

Figura A.6.1: Configuración logstash.conf

3. A continuación, creamos la clave y el certificado de *Logstash:*

```
openssl genrsa -out logstash.key 2048
```

```
openssl req -sha512 -new -key logstash.key -out logstash.csr -config logstash.conf
```

4. Seguidamente, debemos tomar el *serial* de la CA, que encontraremos en la última línea al ejecutar el siguiente comando:

```
openssl x509 -in ca.crt -text -noout -serial
```

Para guardarlo en un nuevo documento utilizando el comando:

```
echo "AEE7043158EFBA8F" > serial
```

5. Ahora, ya podremos utilizarlo para crear y firmar el certificado de *Logstash* mediante los siguientes comandos:

```
openssl x509 -days 3650 -req -sha512 -in logstash.csr -CAserial serial -CA
ca.crt -CAkey ca.key -out logstash.crt -extensions v3_req -extfile
logstash.conf
```

```
mv logstash.key logstash.key.pem && openssl pkcs8 -in logstash.key.pem -
topk8 -nocrypt -out logstash.key
```

A.7. Planificación temporal

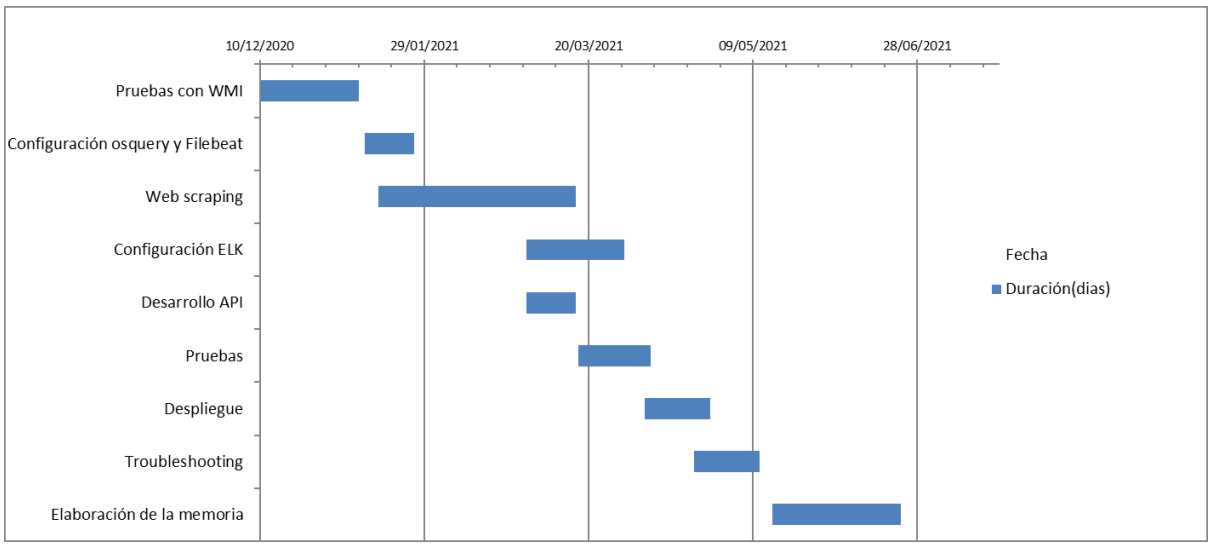

Figura A.7.1: Diagrama de Gantt

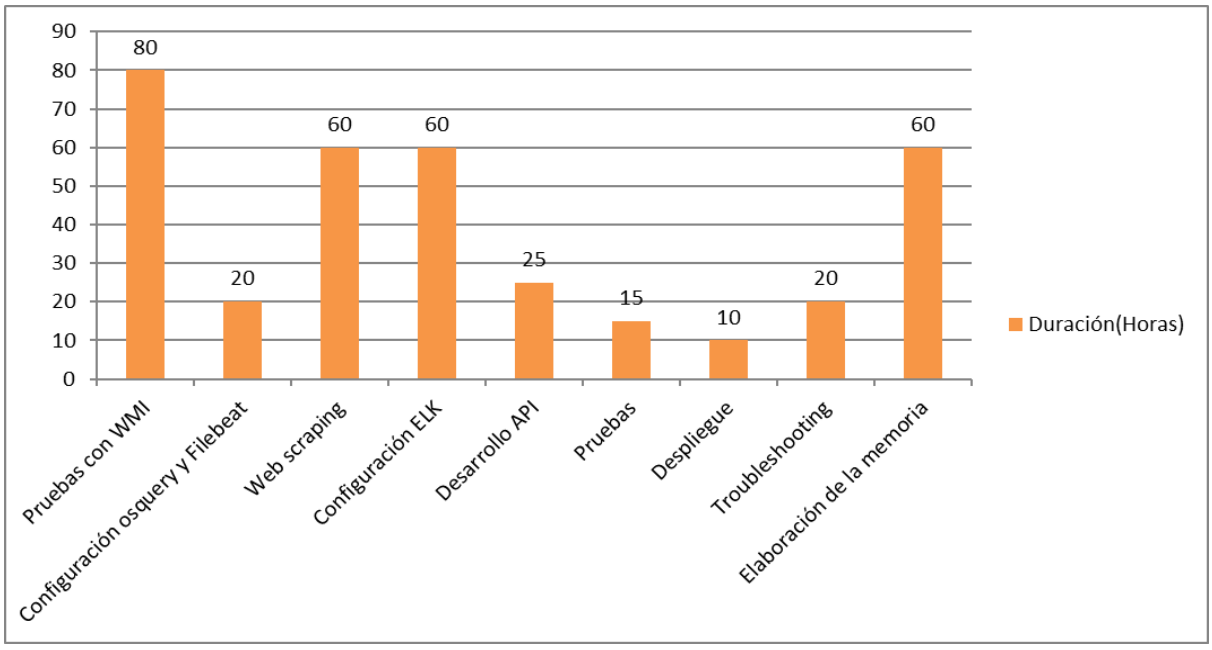

Figura A.7.2: Distribución de las horas de trabajo