Deusto
Facultad de Ingeniería
Universidad de Deusto
Ingeniaritza Fakultatea
Deustuko Unibertsitatea

Grado en Ingeniería Informática
Informatikako Ingeniaritzako Gradua

Proyecto fin de grado
Gradu amaierako proiektua

Investigación y desarrollo de técnicas de análisis de la seguridad web a partir de crawling

Alberto Fernández de Retana Fernández

Director: David Buján Carballal

Bilbao, julio de 2020

Deusto
Facultad de Ingeniería
Universidad de Deusto
Ingeniaritza Fakultatea
Deustuko Unibertsitatea

Grado en Ingeniería Informática
Informatikako Ingeniaritzako Gradua

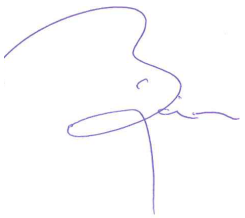
Proyecto fin de grado
Gradu amaierako proiektua

Investigación y desarrollo de técnicas de análisis de la seguridad web a partir de crawling

Alberto Fernández de Retana Fernández

Director: David Buján Carballal

Bilbao, julio de 2020



Resumen

La web se ha convertido en una parte esencial de nuestras vidas, millones de personas hacen uso de ella diariamente para multitud de propósitos, desde leer el periódico hasta gestiones del banco, es por ello que no solo los ciberdelincuentes han puesto el punto de mira en esta tecnología, si no que también entidades legales han desarrollado métodos de trackear a los usuarios mientras navegan por internet.

Al ser una tecnología tan utilizada, la velocidad a la que se desarrollan nuevas formas de aprovecharse de los usuarios y de esquivar las protecciones es muy alta. Todas estas problemáticas dieron lugar al desarrollo de un crawler avanzado que analiza y registra todo (entre otros este) tipo de actividades perjudiciales para la privacidad y la seguridad del usuario.

Por lo tanto haciendo uso de este crawler se desarrollarán nuevas implementaciones para este, para posteriormente poder analizarlo. Estas nuevas funcionalidades serán por ejemplo, registrar la comunicación de una web mediante WebSockets, una tecnología novedosa que permite abrir canales tcp bidireccional entre el cliente y el servidor. Por otra parte, se hará también especial foco en otra tecnología que permite ejecutar código nativo como son los web assemblys. Además se complementarán las ya implementadas con otros proyectos open source para ser capaces de registrar todas las actividades que realiza, sean estas, legales o no.

Descriptores
Crawler, Cryptojacking, WWW, WebAssembly, WebSocket

Índice general

1. Introducción **1**

1.1. Presentación del documento . . . 2

1.2. Motivación . . . 3

2. Antecedentes y estado del arte **5**

2.1. Herramientas de crawling . . . 5

2.1.1. GNU Wget [1] . . . 5

2.1.2. Curl [2] . . . 6

2.1.3. Selenium WebDriver [3] . . . 6

2.1.4. PhantomJS [4] . . . 7

2.1.5. VV8 [5] . . . 8

2.1.6. SlimerJS [6] . . . 9

2.1.7. Puppeter [7] . . . 9

2.1.8. Chrome remote interface [8] . . . 10

2.1.9. OpenWPM [9] . . . 10

2.2. Análisis web . . . 11

2.2.1. Fraudes (Scam) . . . 11

2.2.2. Cryptojacking . . . 14

2.2.3. Seguimiento web (Tracking) . . . 19

3. Objetivos y alcance **23**

3.1. Objetivos . . . 23

3.1.1. Principales . . . 23

3.1.2. Secundarios . . . 24

3.2. Alcance . . . 24

4. Desarrollo del proyecto **27**

4.1. Tecnologías y herramientas . . . 27

4.1.1. Python . . . 27

4.1.2. PyCharm . . . 28

4.1.3. JavaScript . . . 28

4.1.4. NodeJS . . . 28

4.1.5. VIM . . . 29

4.1.6. HTML y CSS . . . 29

4.1.7. DBeaver . . . 29

4.1.8. GitHub . . . 29

4.1.9. JSON 29
4.2. Arquitectura del proyecto 30
4.2.1. El sistema de crawling 31
4.2.2. El sistema de análisis 36
4.2.3. Evaluación de los análisis 63

5. Metodología y planificación **71**
5.1. Metodología 71
5.2. Planificación 72
5.2.1. Fases del proyecto 73
5.2.2. EDT 77
5.2.3. Organización y perfiles 77
5.2.4. Condiciones de ejecución 79
5.2.5. Diagrama de precedencias 79
5.2.6. Diagrama de Gantt 81

6. Presupuesto **89**

7. Conclusión **91**
7.1. Trabajo futuro 93
7.2. Proyectos similares 94
7.2.1. VirusTotal 94
7.2.2. Webutation 94
7.2.3. URLVoid 95
7.2.4. Zulu URL Risk Analyzer 95
7.2.5. Sucuri Sitecheck 95
7.2.6. Siteguarding Sitecheck 97

Índice de figuras

2.1. Logo: Selenium (Fuente: pngocean [10]). 6
2.2. Logo: PhantomJS (Fuente: Wikimedia commons [11]). 7
2.3. Logo: VV8 (Fuente: Página del autor [11]). 8
2.4. Técnica del Monkey-patching. 8
2.5. Logo: SlimerJS (Fuente: Página del proyecto [6]). 9
2.6. Logo: Puppeter (Fuente: Página del proyecto [7]). 9
2.7. Ejemplo de cryptojacking (Fuente: Publicación en ACM del proyecto Minesweeper [12]). 17
2.8. Arquitectura de CMTracker (Fuente: Publicación en ACM del proyecto [13]). . . . 18
2.9. Algoritmo CryptoNight (Fuente: Publicación en ACM del proyecto Minesweeper [12]). 19

4.1. Arquitectura del proyecto. 30
4.2. Logo del proyecto (IsKrawler Analyzer). 36
4.3. Información WHOIS de una página web (Web: google.com). 38
4.4. Consulta en Webutation de una página web (Web: deusto.es). 42
4.5. Consulta en Mozilla Observatory de una página web (Web: deusto.es). 43
4.6. Funciones procesadas en el navegador. 51
4.7. Visualización de las conexiones mediante WebSocket. 53
4.8. Wasm CFG by Octopus Project (Fuente: Octopus Github [14]). 54
4.9. Wasm Call Flow by Octopus Project (Fuente: Octopus Github [14]). 55
4.10. Wasm Function Analytics by Octopus Project (Fuente: Octopus Github [14]). . . . 56
4.11. Frames de una página web (Web: gearhack.com). 58
4.12. Ejemplo de ejecución. 60
4.13. Ejemplo de la exportación HTML (Web: gearhack.com). 61
4.14. Segundo ejemplo de la exportación HTML (Web: gearhack.com). 62
4.15. Diagrama Entidad-Relación. 63
4.16. Resultados de Alexa (IsKrawler Analyzer). 67
4.17. Resultados de los sitios web del SARS-CoV-2 (IsKrawler Analyzer). 68
4.18. Resultados de los sitios web de la lista que realizaban cryptojacking (IsKrawler Analyzer). 69

5.1. Analizador metodología Kanban. 73
5.2. Diagrama EDT. 78
5.3. Diagrama de precedencias para este proyecto. 80
5.4. Diagrama de Gantt para este proyecto 1/7. 82

5.5. Diagrama de Gantt para este proyecto 2/7. 83
5.6. Diagrama de Gantt para este proyecto 3/7. 84
5.7. Diagrama de Gantt para este proyecto 4/7. 85
5.8. Diagrama de Gantt para este proyecto 5/7. 86
5.9. Diagrama de Gantt para este proyecto 6/7. 87
5.10. Diagrama de Gantt para este proyecto 7/7. 88

7.1. Consulta en VirusTotal de una página web (Web: gearhack.com). 95
7.2. Consulta en URLVoid de una página web (Web: gearhack.com). 96
7.3. Consulta en Zulu URL Risk Analyzer de una página web (Web: gearhack.com). . . 96
7.4. Consulta en Sucuri Sitecheck de una página web (Web: gearhack.com). 97
7.5. Consulta en Siteguarding Sitecheck de una página web (Web: gearhack.com). . . . 97

Índice de tablas

4.1. Patrones de los distintos tipos de orquestadores de los servicios de cryptojacking. (Fuente: Publicaciones científicas [12] [15]) 45
4.2. Ejemplo de información obtenida de una cookie en la web www.deusto.es. 50
4.3. Protocolo Stratum y sus palabras claves en los frames recibidos. 52
4.4. Protocolo Stratum y sus palabras claves en los frames enviados. 53
4.5. Datos relevantes de la evaluación en el Top Alexa. 67
4.6. URL sospechosas del análisis en el Top Alexa. 67
4.7. Datos relevantes de la evaluación de sitios web relacionados con SARS-CoV-2. . . . 68
4.8. URL sospechosas del análisis en el Top Alexa. 69
4.9. Datos relevantes de la evaluación de sitios web que realizaban cryptojacking (Fuente: Proyecto Minesweeper [16]). 70
4.10. URL sospechosas del análisis en el Top Alexa. 70

5.1. Tareas del proyecto. 73
5.2. Fechas de la fase de estudios de las conferencias de seguridad web. 74
5.3. Fechas de la fase de análisis del crawler existente. 74
5.4. Fechas de la fase del analizador isKrawler Analyzer. 75

6.1. Presupuesto del proyecto. 89

Capítulo 1

Introducción

Hoy en día no seríamos capaces de imaginarnos nuestra vida sin hacer uso de la tecnología y, sobretodo, sin hacer uso de las facilidades que nos ofrece la web (World Wide Web, WWW en adelante). Tanto particulares como empresas de cualquier tamaño y tipo, tales como financieras, de marketing, de servicios, logísticas, industriales, etcétera, hacen cada vez un mayor uso de estas tecnologías, por lo que el impacto económico que suponen las mismas es enorme y todo indica que seguirá aumentando en un futuro. Valga como ejemplo, aunque circunstancial, el incremento que se ha producido por la reciente pandemia del virus SARS-CoV-2.

Como en toda tecnología, y sobretodo cuando su uso y repercusión en la economía son tan grandes como en este caso, hay personas y entes (grupos, empresas o estados) interesados en aprovecharse de ella por motivos muy dispares. En algunos casos, el motivo puede ser tan simple como poder recopilar la máxima información posible de los usuarios para conocer sus movimientos e intentar seguirlos a través de la red. En otros casos, puede ser el robo de credenciales a gran escala y así un largo etcétera. Por ello, aunque el diseño de la web desde sus comienzos se ha ido mejorando en todos los sentidos añadiendo nuevas tecnologías, los ataques que se pueden realizar sobre ella también se han ido sofisticando, pudiendo utilizar cada vez técnicas más complejas que pueden explotar nuevas vulnerabilidades.

Cada vez se está poniendo más de relieve para personas, empresas y estados que la seguridad es un pilar muy importante de las nuevas tecnologías, y no algo tan secundario como hasta hace no mucho. Por ello, al igual que se han vuelto más complejos los ataques que se producen en la web, también se han sofisticado mucho las contramedidas para que no se produzcan o, al menos, se mitiguen, reduciendo en gran medida su impacto. En este proyecto, nos vamos a centrar en los "crawlers" y los posteriores análisis que se pueden realizar con ellos, ya que se trata de una técnica muy extendida para multitud de funciones y no solo enfocada en la seguridad, convirtiéndose así en una contramedida de gran valor.

Un crawler, que en castellano se podría traducir como "araña web", es un programa informático desarrollado en un lenguaje de programación capaz de escanear páginas webs de la WWW de una forma metódica y automatizada. Como podemos observar en la definición, su uso no tiene por qué estar estrictamente enfocado en la seguridad.

Uno de estos posibles usos, ajenos a la seguridad, puede ser el asociado al crawler Heritrix [17], desarrollado en Java, ya que es usado por Internet Archive [18] para rastrear, recopilar y almacenar páginas web a modo de archivo WWW en el tiempo. Otro posible uso más enfocado en la seguridad podría ser el asociado a Photon [19], que está desarrollado en Python con un enfoque OSINT (Open Source Intelligence [20]), recogiendo datos de fuentes disponibles de forma pública para ser utilizados en un contexto de inteligencia. Un ejemplo de sus funcionalidades, entre otras, es recopilar los emails o cuentas de RRSS que se encuentren en un dominio concreto.

En definitiva, el uso de los crawlers está muy extendido porque ofrecen muchas funcionalidades: desde interpretar las cabeceras o el DOM (Document Object Model) de una web y actuar en consecuencia, hasta analizar completamente todo tipo de ficheros y comunicaciones que incluye. Existen crawlers genéricos y otros que son bastante más avanzados y especializados, como el que se va a utilizar en este proyecto.

En el presente proyecto se va a poner foco sobre las técnicas de análisis de la seguridad web a partir del crawling. Para ello, haremos uso de un crawler desarrollado por terceros, añadiendo alguna funcionalidad e implementando un analizador de los resultados obtenidos a partir del mismo. El crawler que se va a utilizar es una reimplementación en otra interfaz más a bajo nivel de otro crawler realizado con anterioridad. El anterior crawler llamado TrackingInspector [21] fue desarrollado en el contexto de la tesis doctoral de un compañero del grupo de investigación en el que estoy dentro de DeustoTech, pero que estaba centrado en la privacidad web y en el tracking de usuarios. Este crawler estaba basado en el proyecto PhantomJS [4] y se reimplementó en una interfaz del protocolo de depuración de Chromium [8] para Node.

La solución desarrollada en este proyecto será capaz de analizar páginas web a partir de sus URLs y de parametrizar cómo deseamos llevar a cabo el análisis y cómo realizar el informe de salida del mismo, por ejemplo, por pantalla o exportando un documento HTML más amigable para un usuario humano. Por otra parte, se hará uso también de otros proyectos y desarrollos para complementar la información de nuestro análisis.

Este proyecto se basa en desarrollar un analizador enfocado en el crawler comentado, pero que utiliza técnicas para descubrir actividades sospechosas y que podrían ser fácilmente adaptadas para otros proyectos. Además, este analizador será implementado de una forma modular, permitiendo añadir o quitar funcionalidades de una manera sencilla, sin interferir con el resto de procesos llevados a cabo. Por otra parte, el IsKrawler Analyzer implementará diversas técnicas presentadas en diferentes conferencias de seguridad web convirtiéndolo en una solución más amplia, accesible y útil.

1.1. Presentación del documento

El presente documento constituye la memoria del proyecto *Investigación y desarrollo de técnicas de análisis de la seguridad web a partir de crawling*. La estructura de los capítulos y apartados se han realizado siguiendo con los criterios y recomendaciones que establecen las normativas para la memoria del PFG.

En este apartado se expone de qué forma estará estructurada la memoria y que capítulos contendrá. A continuación tenemos los capítulos y una breve descripción de estos.

- Capítulo 1: Introducción. Descripción breve del proyecto, presentación de la estructura de la memoria y finalmente, la motivación y justificación del proyecto.

- Capítulo 2: Antecedentes y estado del arte: Se describe el estado del arte de las actuales interfaces de crawling y de las distintas temáticas claves en la seguridad web.

- Capítulo 3: Objetivos y alcance. Se describirán los objetivos planteados para el proyecto, además de sus limitaciones y el alcance de estos.

- Capítulo 4: Desarrollo. Se describirá la parte más técnica del presente trabajo, explicando el diseño del mismo, las técnicas que se utilizan para analizar las actividades maliciosas y las herramientas/tecnologías de las que se han hecho uso en el transcurso del proyecto.

- Capítulo 5: Metodología y planificación. Se expondrá la metodología utilizada para realizar este trabajo, así como también se describirá la planificación que se ha llevado a cabo para el desarrollo de este proyecto.

- Capítulo 6: Presupuesto. Se describirá el presupuesto utilizado durante el transcurso del proyecto, en base a los recursos humanos y materiales necesitados.

- Capítulo 7: Conclusión. Finalmente, en este capítulo se expondrán las conclusiones a las que se ha llegado con la realización del proyecto. Se describirán también, herramientas similares a la realizada y se expondrán las posibles líneas de trabajo futuro.

1.2. Motivación

Son muchos los casos de sitios web fraudulentos que aparecen cada día y como tratan de aprovechar cualquier situación para estafar a los incautos, como por ejemplo, el caso de la pandemia del SARS-CoV-2 [22]. Por otra parte, la seguridad web y la web en sí, son ámbitos de la informática que evolucionan a una velocidad vertiginosa, por lo cual requieren de una formación continua de todos los profesionales que se quieran dedicar a ella, siendo por ende, una temática muy cautivadora.

Son varias las motivaciones por las cuales inicié el proyecto, la primera era la posibilidad de utilizar la experiencia y formación que me venía aportando mi beca de colaboración en DeustoTech, en el que venía realizando investigación sobre la seguridad web, transformándolo en un proyecto real. Otra razón era la posibilidad que se me ofrecía de poder hacer uso del crawler de la unidad, el cual fue desarrollado en el equipo, previo a mi llegada. Otro motivo que encontré para realizar este proyecto, era la posibilidad de investigar, implementar y aplicar estudios y proyectos que se habían presentado en conferencias importantes en el ámbito de la seguridad web, en una solución más amplia y con vistas, a crear una herramienta íntegra y completa. Por último, y una de las mayores razones por las que elegí emprender este proyecto, era el de poder especializarme y aprender en un área tan complicada e interesante a partes iguales, como es la seguridad y en la cual me gustaría seguir formándome con un máster al finalizar el grado.

Capítulo 2

Antecedentes y estado del arte

En este apartado se describirán el estado del arte de diversos temas, dicho de otro modo, a día de hoy, cómo se encuentra cada una de estas cuestiones. El primer tema, como no podía ser de otro modo, es el de herramientas que podemos utilizar para hacer crawling, y se describirán las distintas herramientas con las funcionalidades que se utilizan y la forma en la que realizan la navegación. Se abordarán también algunas soluciones propuestas en publicaciones de conferencias. El segundo, es el de la seguridad web, y sobretodo centrada en los documentos y proyectos de grandes conferencias usados para el proyecto, dándonos una perspectiva de en qué temas está enfocado hoy en día la investigación en este área.

2.1. Herramientas de crawling

Los crawlers, como se ha comentado en la Introducción, son "arañas informáticas" las cuales recorren la web de una forma automatizada y metódica. Debido a la velocidad vertiginosa en la que avanza la web y todo lo que a esta involucra, la misión de diseñar un crawler que sea capaz de analizar todas las acciones que realiza una página web se ha convertido en una tarea muy compleja. Para poder realizar esto, hay un montón de proyectos que nos lo facilitan, y es por ello, por lo que se añade esta sección, como forma de introducir a una persona sin grandes conocimientos en los distintos proyectos, explicando en qué casos se utilizan.

En este apartado se describirán algunas de las herramientas más utilizadas y populares para realizar crawling sobre páginas webs. En los siguientes apartados se irán describiendo de forma creciente en base a la complejidad y la funcionalidad que estas tienen, empezando por herramientas muy básicas que en algunos casos podemos utilizar puntualmente, para seguir con herramientas muy específicas que suelen ser usadas para grandes proyectos, por ejemplo, de investigación. Se incluirán y describirán también, algunos frameworks utilizados para la investigación sobre la privacidad web, en caso de que sea de este tipo, se nombrará.

2.1.1. GNU Wget [1]

Esta herramienta open source suele estar instalada por defecto en los sistemas GNU/Linux y es una de las herramientas más simples para realizar crawling. Suele ser utilizada desde línea de comandos

aunque también tiene versiones gráficas. El uso más habitual que tiene esta herramienta es el de descargar ficheros de forma sencilla.

Soporta los protocolos HTTP, HTTPS y FTP y es una gran utilidad para la descarga de ficheros de la web de forma no-interactiva. Siendo la descarga de recursos su funcionalidad, incluye una infinidad de opciones para llevarla a cabo, como puede ser el que haga uso de determinadas HTTP Cookies o hacer uso de la descarga recursiva recreando completamente la estructura de directorios de la página original.

2.1.2. Curl [2]

Esta herramienta es open source, y también suele estar por defecto instalada en los sistemas GNU/-Linux. Aún así, se trata de otra herramienta muy simple para crawling y que no suele ser utilizada para ello. Esta herramienta también se utiliza desde línea de comandos y el caso más típico y utilizado es para probar API REST dado que nos permite realizar peticiones HTTP Post de forma sencilla pudiendo añadir los datos a enviar con un simple parámetro.

Curl está construida como envoltorio o wrapper de la biblioteca libcURL. Nos permite transferir datos desde un servidor o hacia un servidor y soporta una cantidad enorme de protocolos. Por otra parte tiene una cantidad infinita de funcionalidades para realizar el crawling como puede ser indicarle un certificado TLS a usar o para simular el envío de un formulario en protocolos de la familia HTTP.

2.1.3. Selenium WebDriver [3]

Fig. 2.1: Logo: Selenium (Fuente: pngocean [10]).

Selenium, desarrollado en Java por la empresa SeleniumHQ, es una herramienta muy conocida y que su utilidad abarca más que simplemente la de un crawler. Esta herramienta nos permite automatizar la interacción de un usuario real en la mayoría de navegadores web más conocidos. Esta herramienta permite realizar este tipo de automatización de forma muy sencilla y por lo tanto, se utiliza mucho para automatizar pruebas sobre páginas webs.

Esta solución es muy utilizada, pero es una interfaz que nos abstrae mucho y no nos permite registrar todo lo que el navegador realiza por lo que en caso de necesitar una interfaz que nos registre todo tipo de actividad y sobretodo a nivel de investigación no es una opción a elegir. Está pensada más para, como se ha comentado en el anterior párrafo, automatizar pruebas o en algunos casos,

recorrer una página para descargar ficheros que se encuentran en ella, pudiendo implementar que haga clic en determinados botones o que dentro de la web deslize para la parte baja de la web.

2.1.4. PhantomJS [4]

Fig. 2.2: Logo: PhantomJS (Fuente: Wikimedia commons [11]).

PhantomJS hace uso de WebKit para ser programable con JavaScript y se trata de una navegación "headless" o sin cabeza. WebKit es el motor de navegación para navegadores como Safari, Midori y otros. La última versión estable es la 2.1 pero actualmente, desde marzo de 2018, el desarrollo está suspendido. Para poder hacer uso de este proyecto debemos desarrollar nuestro crawler en el lenguaje de programación JavaScript.

Al igual que todos los crawlers, el nuestro también permite automatizar la navegación por las diferentes páginas web. Al contrario que Selenium WebDriver [3], nuestro proyecto ofrece más funcionalidades, como son el análisis del rendimiento o la capacidad de examinar los recursos que carga la página, entre otros. Este proyecto fue, hasta su abandono, uno de los más utilizados en la investigación de seguridad y privacidad web. Un ejemplo es el crawler que vamos a utilizar en el proyecto [21], que estuvo siendo desarrollado con PhantomJS [4] hasta que el proyecto se abandonó y se decidió adaptar a otro proyecto.

El problema con PhantomJS [4] es que hace uso de la navegación sin mucho criterio. Esto en algunos casos puede ser una ventaja para aumentar el rendimiento y la velocidad del crawler, pero en otros puede ser una desventaja. Además, existen técnicas para detectar este tipo de navegación, por lo que si una web que realiza acciones maliciosas detecta esta navegación, podrá suponer que se trata de un crawler y, por lo tanto, devolverá una página benigna sin dar comienzo a las actividades malignas, escondiéndose de forma muy sencilla. Un ejemplo de las técnicas que se utilizan para llevar a cabo este tipo de comportamiento se pueden ver en el siguiente proyecto de GitHub, llamado Detect Headless [23], en el que entre otras cosas revisan el User Agent de la petición o los lenguajes que acepta el navegador para comprobar si dicha navegación es "headless" o no.

2.1.5. VV8 [5]

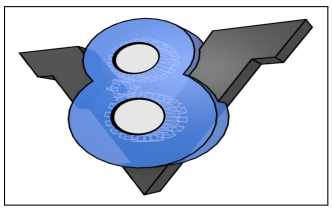


Fig. 2.3: Logo: VV8 (Fuente: Página del autor [11]).

Este proyecto es el más reciente y uno de los más novedosos. Fue presentado en la conferencia de ACM Internet Measurement Conference (IMC) de 2019 [24] mediante el artículo "VisibleV8: In-browser Monitoring of JavaScript in the Wild" [25]. También se pueden consultar el vídeo de la charla [26], las transparencias [27] y el proyecto mismo [28].

Esta herramienta, no es un proyecto como tal para realizar crawling, pero gracias a él se puede modificar Chromium para recopilar información respecto a las ejecuciones de JavaScript y de WebAssembly que se realizan en el mismo. Dado que añade un envoltorio al navegador Chromium que permite recopilar información que antes no era posible recuperar simplemente desde este navegador, se considera relevante mencionarlo en esta sección.

Como el nombre indica, el proyecto hace uso del proyecto V8 [29]. V8 [29] es un motor de código abierto para JavaScript y WebAssembly utilizado por Chromium, Microsoft Edge y Brave. Por eso el logo del proyecto incluye el símbolo del motor V8, pero en transparente y con unos engranajes visibles, una forma de dejar patente esa característica que ofrece esta nueva herramienta.

Esta herramienta permite añadir un wrapper a Chromium en el que se utiliza la técnica "Monkey-patching" y que está desarrollado en JavaScript para funcionar encima del mismo, permitiendo que sea más fácil de mantener debido a los pocos cambios que se tienen que realizar entre versiones. Básicamente, esta técnica de "Monkey-patching" se utiliza para que cada vez que se invoque una función nativa, se llame primero a una función nuestra que realice una determinada acción de nuestro interés, que en este caso es hacer logging de los argumentos. En la figura 2.4 se muestra un ejemplo de lo que nos permite realizar este proyecto.

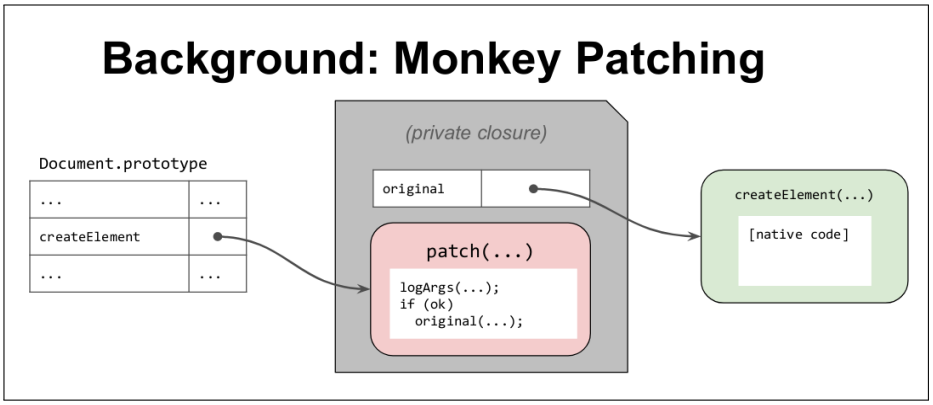


Fig. 2.4: Técnica del Monkey-patching.

2.1.6. SlimerJS [6]

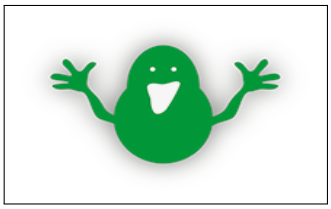


Fig. 2.5: Logo: SlimerJS (Fuente: Página del proyecto [6]).

SlimerJS es un proyecto muy similar al PhantomJS [4] explicado previamente, pero en este caso en vez de hacer uso del motor WebKit scriptable con JavaScript, está basado en Gecko, que es el motor de Mozilla. A la hora de realizar un proyecto con SlimerJS, tenemos que saber que el código que queramos implementar será muy similar al que implementaríamos en PhantomJS [4].

Las funcionalidades que nos ofrece SlimerJS son también parecidas a las de PhantomJS [4]. Por otro lado, el problema que teníamos de que fuese una navegación "headless" o sin cabeza sigue existiendo por la forma en la que están implementados estos proyectos, es decir, dado que son wrappers de un motor de renderizado, tanto SlimerJS [6] como PhantomJS [4], no se nos permite configurar que no haga uso de la navegación "headless' y lance un crawling completo del navegador.

2.1.7. Puppeter [7]

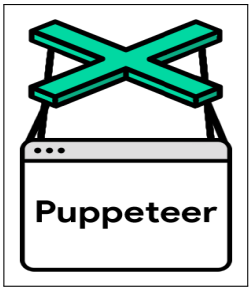


Fig. 2.6: Logo: Puppeter (Fuente: Página del proyecto [7]).

Puppeter es una librería de Node que nos proporciona una API de alto nivel para controlar Chrome o Chromium mediante el protocolo DevTools [30]. Este proyecto nos permite lanzar el crawling de forma "headless" o sin cabeza, al igual que PhantomJS [4] o SlimerJS [6], pero además podemos utilizarlo completamente, es decir, sacando provecho de toda su funcionalidad.

La mayoría de funcionalidades que se podrían obtener utilizando manualmente un navegador, se pueden realizar utilizando Puppeter [7]. En su web se muestran algunos ejemplos de los usos que se le podrían dar:

- Generar capturas de pantallas y PDFs de las páginas.
- Recorrer una SPA (Single-Page Application) y generar un contenido prerenderizado (i.e. "SSR' (Renderizado en el lado del servidor)).

- Automatizar el envió de formularios, testeo de UI, entrada de teclado, etcétera.
- Crear un entorno de pruebas actualizado y automatizado. Poder ejecutar las pruebas directamente desde la última versión de Chrome utilizando las últimas funciones de JavaScript y las últimas funcionalidades de Chrome.
- Capturar el seguimiento de la línea de tiempo del sitio web para poder diagnosticar más fácilmente los problemas de rendimiento.
- Pruebas de extensiones de Chrome.

Este proyecto nos ofrece una web [31] en la que poder desarrollar un pequeño ejemplo del funcionamiento, dicho de otro modo, nos ofrece un editor de texto para poner el código, ejecutarlo y ver los datos recogidos y los resultados.

La ventaja de este proyecto frente a los anteriores comentados es que hace de, como indica su nombre, titiritero de Chrome. Esto significa que utiliza un navegador completo para hacer el crawling, proporcionando mayor realismo y haciendo pasar al crawler por una navegación humana. La desventaja principal es que al hacer uso de un navegador completo, tarda mucho más tiempo en realizar la navegación.

2.1.8. Chrome remote interface [8]

Este proyecto, utiliza al igual que Puppeter [7], el protocolo de depuración de Chrome [30]. Esta herramienta también sirve para desarrollar un crawler en JavaScript. En nuestro caso, el crawler que vamos a utilizar en este trabajo, IsKrawler, está desarrollado haciendo uso de Chrome remote interface [8].

Simplemente deberíamos añadir el paquete de npm del repositorio [32] y podríamos comenzar a implementar nuestro propio crawler. Como ejemplo, haciendo un poco de auto-publicidad, realice un pequeño crawler demo sobre cómo se realizaría esta implementación y que se encuentra publicada en mi cuenta de GitHub [33].

La ventaja que nos ofrece frente al anterior es que permite hacer el crawling a más bajo nivel, recopilando e implementado la máxima funcionalidad posible, dado que queremos comprobar todas las acciones que realiza la web. El problema que tiene, al igual que el proyecto Puppeter [7], además del rendimiento, es que la documentación es escasa y la API de la que hacen uso tiene muchas funciones inestables.

2.1.9. OpenWPM [9]

OpenWPM [9] es un framework para la medición de la privacidad web que facilita la recopilación de datos para estudios de privacidad en una escala de miles a millones de sitios. Está construido sobre Firefox, con una automatización proporcionada por Selenium. Incluye varios “hooks” o ganchos para la recopilación de datos.

Este proyecto open source fue desarrollado por la Universidad de Princeton [34], pero se migró a Mozilla [35]. A partir del mismo se han presentado diversos artículos en conferencias, como puede ser el "OpenWPM : An automated platform for web privacy measurement" [36] o el "Online Tracking: A 1-million-site Measurement and Analysis" [37].

Este proyecto nos ofrece una gran cantidad de información en su GitHub [9], destacando entre ella un apartado de una demo simple de crawling [38].

Las ventajas de este proyecto son que hace uso de un navegador completo permitiendo las ejecuciones de JavaScript con normalidad y que permite una instrumentación a bajo nivel del código JavaScript y del protocolo HTTP. Las desventajas son que es mucho más lento y que es demasiado complejo, ya que muchos usuarios solamente necesitan un crawling a alto nivel porque solo están interesados en el contenido de la web y esta no sería la mejor elección.

2.2. Análisis web

En esta sección se hará una introducción a algunos de los temas más relevantes dentro de las seguridad y privacidad web. Además de la introducción, se aportará información de publicaciones de conferencias importantes de investigación sobre cada uno de los temas para poder comprender en qué estado se encuentran cada una de ellas.

2.2.1. Fraudes (Scam)

El fraude en internet, también conocido como "scam" o estafa en línea, engloba cualquier tipo de esquema de fraude que se focaliza en el uso de servicios de internet, con el objetivo de engañar a personas incautas para poder aprovecharse de ellas, como por ejemplo, para robarles información personal que puede derivar luego en suplantaciones de identidad o en robos de cuentas, etcétera.

Este tipo de estafas suelen aplicar técnicas de ingeniería social e influencia social [39]. Además un ejemplo muy actual, son todas las estafas relacionadas con el coronavirus [40], ya sea para que te realicen pruebas gratuitas, para hacer la compra online, para realizar un pedido de mascarillas, u otros tipos.

Los atacantes, es decir, los estafadores o "scammers", tratan de apelar algún sentimiento en las víctimas, como puede ser, la buena fe, la necesidad, la avaricia o el enfado. Este tipo de técnica suele evolucionar y variar con el paso del tiempo. Como forma de intentar mitigar estas estafas, existen organizaciones dedicadas a registrar cada sitio web que realiza esta actividad maliciosa y trabajan para prevenir que más gente siga cayendo en estas trampas.

Algunas de estas organizaciones son gobiernos [41] [42] [43] o entidades de protección a los consumidores [44]. Normalmente, la técnica más utilizada para mitigar el scam, y otras actividades maliciosas, es manteniendo listas negras en las que se van añadiendo sitios web según se van reportando casos maliciosos. Normalmente estas estafas están agrupadas por temáticas como se puede

ver a continuación [45]:

- **Dinero inesperado**: Se engloban aquí todas las estafas en las que una persona por diversas razones te ingresará una cantidad bastante grande de dinero.
- **Premio inesperado**: En este grupo entran todas los fraudes que te hacen crecer que te ha tocado algún tipo de premio.
- **Falsa caridad**: Entran dentro de esta clase las estafas en las que apelan a tu buena fe para donar dinero a una ONG o causa justa.
- **Romance & Citas**: Se engloban aquí todas las estafas dirigidas a personas que están buscando una pareja o citas por algún tipo de red social.
- **Compra o Venta**: Aquí se incluyen todos los fraudes dirigidos a empresas o personas en los que tratan de venderte o comprarte algún producto, tratando de ganar información y en los que las transacciones no suelen ser legítimas.
- **Trabajos o inversión**: En este grupo entran todas las estafas que tratan de aprovecharse de gente que necesita dinero de una manera rápida.
- **Intentan ganar información personal**: Se incluyen todo tipo de técnicas para tratar de ganar información personal tuya para multitud de finalidades, como puede ser el usar tu tarjeta de crédito, suplantarte la identidad o abrir una cuenta del banco a tu nombre.
- **Amenazas y extorsiones**: Entran dentro de este grupo todo tipo de scam que trate de amenazarte de cualquier forma, como que han hackeado tus cuentas y ordenadores, o que saben donde vives y trabajas, etcétera.

El fraude o "scam" no es nada novedoso y según han ido avanzando los años y las tecnologías han ido evolucionando y expandiéndose. El desarrollo de la tecnología, les permitió llegar a una gran cantidad de gente con un mínimo esfuerzo, y el primer ejemplo, fueron las llamadas telefónicas y los mensajes SMS. Un caso típico en estos fraudes, es el de una víctima que recibe un mensaje de texto indicándole que si manda determinada cantidad de mensajes SMS recibirá cierto servicio o premio por ello, y que en caso, de la víctima caer en la trampa estaría embolsando a los estafadores el dinero de los SMS.

Los casos más relevantes de estafa que se suelen dar utilizan como medio de distribución el correo electrónico o email. Este ha sido uno de los recursos más utilizado y explotado por los estafadores para realizar las estafas, consiguiendo llegar a una cantidad enorme de personas, las cuales muchas, no tienen ningún tipo de conocimiento informático.

Por último, otro gran caso en el que se dan estas estafas es en las páginas web. Los estafadores utilizan los sistemas de comentarios para difundir estafas, haciendo que mucha gente, que por ejemplo, se encuentra viendo un vídeo o una noticia, clique fácilmente.

Por todo esto, la comunidad de investigación en seguridad pone mucho esfuerzo en comprender y mitigar este tipo de actividad maliciosa. Es por ello que se ha realizado una gran cantidad de

publicaciones en conferencias abordando esta temática, y en los siguientes párrafos se tratará de describir de los más relevantes.

El primero de ellos, se titula "Botnet Spam Campaigns Can Be Long Lasting: Evidence, Implications, and Analysis" [46], y se trata de un profundo análisis sobre las características, independientemente del contenido o de las campañas de spam. Por ejemplo, la duración y la distribución de red de los remitentes de correo no deseado, para determinar si pueden ayudar a los métodos simples de agrupamiento basados en etiquetas para identificar campañas y generar firmas de campaña. Este estudio observó, que casi todas las campañas de correo no deseado, cuando se realizaban desde grandes dominios de destino, exhibían una duración de ráfaga mucho más larga que el periodo de cinco días que utilizaban como umbral de ráfaga de campaña en un estudio anterior. La consecuencia de esto, es que no se podía agrupar como una misma campaña simplemente todo el correo no deseado de una gran cantidad de fuentes de correo por contener URLs similares y cuya entrega finalizaba en un corto periodo de tiempo.

Otra publicación sobre "scam" que se titula "Spamscatter: Characterizing Internet Scam Hosting Infrastructure" [47], se centraba en caracterizar la infraestructura que utilizaban, para poder después entender las dinámicas y funcionamientos de estos grupos. En este estudio utilizan una técnica que denominan "Spamscatter" para identificar la infraestructura de los fraudes. Crearon un sistema que extraía el correo electrónico en cuestión, identificaba las URLs en tiempo real y sigue cada uno de los enlaces al servidor destino. Posteriormente, evalúan continuamente estos servidores para analizar los comportamientos, su disponibilidad y la duración. Por lo tanto, utilizaron esta técnica a una gran escala para obtener los datos. Descubrieron, que aunque en las estafas que se publicaban en Internet solían utilizar una gran cantidad de servidores, las estafadas individuales residían en una sola máquina. Por otra parte, localizaron que la mayor parte de servidores de estas campañas estaban alojadas en Estados Unidos. Y por último, que las campañas solían iniciarse enviando una gran cantidad de correos durante medio día y el servidor permanecía activo alrededor de una semana.

Muchas estafas, debido a su popularidad e impacto, se han hecho bastante conocidas y, a causa de eso se han realizado publicaciones científicas centradas en ellas. Uno de esos fraudes, es el conocido como estafa nigeriana o "scam 419". Adquiere su nombre por el número de artículo del código penal de Nigeria que viola [48].

Un estudio que analiza bastante en profundidad las estafas nigerianas que se han dado en el conocido y concurrido sitio web Craiglist es el titulado como "Scambaiter: Understanding Targeted Nigerian Scamson Craigslist" [49]. El objetivo de este estudio era comprender la economía clandestina de este tipo de estafas. Para ello, publicaron anuncios trampa, preparados para atraer a estos estafadores pero alejar a los usuarios normales. Tras esto, reciben y contestan una gran cantidad de correos electrónicos fraudulentos para analizar posteriormente. Para todo esto, desarrollaron un sistema que publicaba anuncios, recopilaba correos fraudulentos y que interactuaba respondiendo a esos correos. Después de realizar el análisis sobre una cantidad inmensa de datos, se dieron cuenta que la mayor parte de estafas se podían vincular a 10 grupos principales, que estaban dirigidos a anuncios distribuidos en muchas clases de productos y regiones geográficas de Craiglist. También descubren, entre otras cosas más, que el 23 % de las direcciones de envío se encuentran en Estados

Unidos, aunque la mayoría de las direcciones IP de los servidores y de envío se encontraban en Nigeria.

En esta sección, se ha tratado de realizar una introducción y un estado del arte sobre una de las actividades maliciosas más relevantes en los últimos años como es el "scam". Podemos también afirmar que en los últimos años y debido a estos desarrollos científicos se han ido reduciendo enormemente este tipo de técnicas, aunque obviamente, avanzan al igual que las mitigaciones.

2.2.2. Cryptojacking

El cryptojacking, llamado también como minería de criptomonedas maliciosa, es una actividad ilícita que utiliza los recursos de procesamiento de la máquina y electricidad de la víctima para minar diversas formas de monedas digitales. En este caso, dado que nos hemos centrado en la seguridad web, lo reduciremos a sitios web, dejando de lado otro tipo de técnicas. Por lo tanto, se trata básicamente, que una persona que visita determinada web, según esta navegando por la misma, la web está ejecutando operaciones de hash para el beneficio de un tercero utilizando los recursos de la máquina del visitante.

Se trata de una actividad ilícita, dado que no nos pide nuestro consentimiento para ello, utiliza nuestro rendimiento del sistema y nuestro consumo de suministros eléctricos, afectando negativamente a la vida del dispositivo. Aún así, esta técnica en algunos casos no tiene por qué ser maliciosa, si se nos explicita que se va a realizar, como puede ser el sitio web que proporciono Unicef llamado "The hope page" [50]. En este sitio web permitimos que utilicen nuestro ordenador para minar criptomonedas pero con un fin benéfico. Además, puede ser una técnica alternativa a los anuncios invasivos para que puedan rentabilizar las distintas web de la WWW. Por otra parte, en esta sección no se va a tratar todo lo relacionado con las criptomonedas, ni las operaciones de minado, dado que podrían dar para un capítulo entero, por lo tanto, nos centraremos en cómo los atacantes realizan este tipo de operaciones desde un sitio web.

El cryptojacking, fue desarrollado por Jeremy Rubin, estudiante en su momento del MIT, que desarrolló un programa denominado Tidbit con algunos compañeros en el Hackaton llamado "Node Knockout" en noviembre de 2013. Este programa permitía minar Bitcoins en los ordenadores de los clientes, sustituyendo a la publicidad habitual. Tidbit fue presentada como una prueba de concepto y ganó el premio de la mayor innovación del hackathon.

En diciembre de 2013, el fiscal general de New Jersey emitió una citación general a Rubin y Tidbit, buscando el código fuente, los documentos y las respuestas de cómo funcionaba Tidbit, en que lugares estaba instalado y a que cuentas de Bitcoin estaban asociadas. Con la ayuda del abogado Frank Corrado, la Electronic Frontier Foundation (EFF) se movió para anular la citación e impugnarla.

Tras muchos movimientos por las dos partes, en mayo de 2015, las partes firmaron una orden de consentimiento para poner fin a la investigación de Tidbit en Nueva Jersey con diversas restricciones y con el pago de 25.000$.

En 2017, nació la plataforma Coinhive, que permitía a cualquier desarrollador incrustar en su sitio web un código sencillo para utilizar el hardware de los visitantes para minar criptomonedas. Dicho de otro modo, te ofrecía una API JavaScript para minar la criptomoneda Monero (XMR) de una forma sencilla, como se puede ver en el fragmento de código. Esta plataforma cesó su actividad el año pasado, pero su modelo de negocio basado en que los beneficios de las monedas se repartían entre el cliente y ellos, les permitió amasar una gran cantidad de dinero durante el tiempo que estuvieron activos. El fragmento que vemos de JavaScript es el encargado de iniciar el proceso, también se le suele denominar "Orchestrator", en castellano orquestador.

```
<script src="https://coinhive.com/lib/coinhive.min.js"></script>
<script>
    // Arrancar minero
    var miner = new CoinHive.Anonymous('ID-CLIENTE', {throttle:
        ↪ 0.9});
    miner.start();
</script>
```

Listing 2.1: JavaScript para realizar cryptojacking (Coinhive).

Esta técnica se expandió y nacieron muchas más plataformas que realizaban operaciones similares ofreciendo scripts de JavaScript con la misma funcionalidad. Por ejemplo, la plataforma StatDynamic, la cual ofrece funcionalidades similares que podemos observar en el siguiente trozo de código:

```
<script src="//statdynamic.com/lib/crypta.js"></script>
<script>
    // Arrancar minero
    var miner = new CRLT.Anonymous('CLIENTE-ID', {threads:2,
        ↪ throttle:0.5, coin: "upx"});
    miner.start();
</script>
```

Listing 2.2: JavaScript para realizar cryptojacking (StatDynamic).

Esta técnica, permite a los cibercriminales con poco esfuerzo, poder monetizar sitios web en su propio beneficio. Esto se puede conseguir comprometiendo servidores (vía exploits [51] [52] [53] [54] [55]) o aprovechándote de malas configuraciones [56]) o instalando mineros JavaScript y distribuirlos mediante anuncios (incluyendo DoubleClick de Google en Youtube [57] o en la plataforma de anuncios AOL [58]), o comprometiendo librerías de terceros [59] que utilizan una gran cantidad de sitios web.

Esta actividad maliciosa, que normalmente se realiza utilizando JavaScript, tuvo su auge entre 2017 y 2018 debido a su dependencia de las criptomonedas, que tuvieron también su auge en esos años. Antes de dar comienzo a la descripción de la algunas de las publicaciones más relevantes

sobre estas actividades se definirán alguna de las tecnologías de las que suelen utilizar en el cryptojacking para entender el cómo lo realizan.

- **WebAssembly & asm.js**: Para permitir una ejecución más rápida de código dentro del navegador, Mozilla desarrolló asm.js, una técnica para traducir lenguajes de alto nivel, como C y C++, en JavaScript para ser utilizado por el navegador [60]. Esta técnica, lanzada en 2013, permitía ejecutar código más rápido dentro del navegador. WebAssembly (Wasm) es un lenguaje novedoso de scripting desarrollado por el consorcio de la World Wide Web en 2017 y es capaz de compilar lenguajes de alto nivel, como C, C++ y Rust para que puedan ser usado dentro del navegador en aplicaciones web. Se ejecuta en una "sandbox" dentro del navegador y su objetivo es ejecutarse tan rápido como el código de máquina nativo. Se complementa con JavaScript, debido a que es controlado por código JavaScript después de su compilación. La diferencia entre estas dos tecnologías, es que Wasm se compila una única vez y se ejecuta en velocidad nativa, mientras que el código asm.js es compilado y optimizado en tiempo de ejecución, disminuyendo así la velocidad de ejecución. Estas dos técnicas son muy atractivas para la minería basada en el navegador y por eso se explican en este apartado.

- **WebSockets & Stratum**: WebSockets es un protocolo HTML5 que proporciona comunicación bidireccional entre el cliente y un servidor a través de una única conexión TCP [61]. El protocolo permite una fácil transferencia de datos en tiempo real sin actualizar (una parte de) la página web. La comunicación se realiza a través de los mismos puertos TCP que el navegador web, lo que lo hace robusto a las estrictas reglas de firewall u otros bloqueos. Los desarrolladores son libres de definir el formato de los mensajes enviados a través de las conexiones WebSocket. Sin embargo, existe un protocolo diseñado específicamente para las comunicaciones de cryptojacking: el protocolo de minería Stratum, un protocolo basado en líneas con mensajes codificados en formato de texto sin formato JSON-RPC [62] [63] [64] [65]. Los servidores se comunican con sus clientes utilizando Stratum para autorizar a los nuevos mineros en el grupo, distribuir trabajos basados en la dificultad y recuperar los hashes encontrados de los mineros.

En la figura 2.7 se puede observar un ejemplo de cómo funciona esta actividad maliciosa denominada como cryptojacking. Se trata de una figura extraída de una de las publicaciones explicada más adelante [12].

Algunas de las soluciones propuestas para mitigar este problema son heurísticas para mediar el uso de la CPU y, al igual que para otras actividades maliciosas web, el guardar en listas negras los sitios que realizan estas actividades. Estas mitigaciones, realmente no buscan las raíces de la actividad maliciosa y por lo tanto, se pueden sortear muy sencillamente con la reducción de CPU del minado y con la aleatorización de las URLs. Es por ello, que en los siguientes párrafos se presentarán distintas publicaciones científicas en las que ahondan en esta problemática y sus posibles contramedidas.

Una de las primeras publicaciones respecto a esta temática fue en marzo de 2018, para la conferencia IEEE de seguridad y privacidad, y se llamaba "A first look at browser-based Cryptojacking" [15]. En este estudio realizan un primer análisis sobre esta temática. También realizan mediciones para

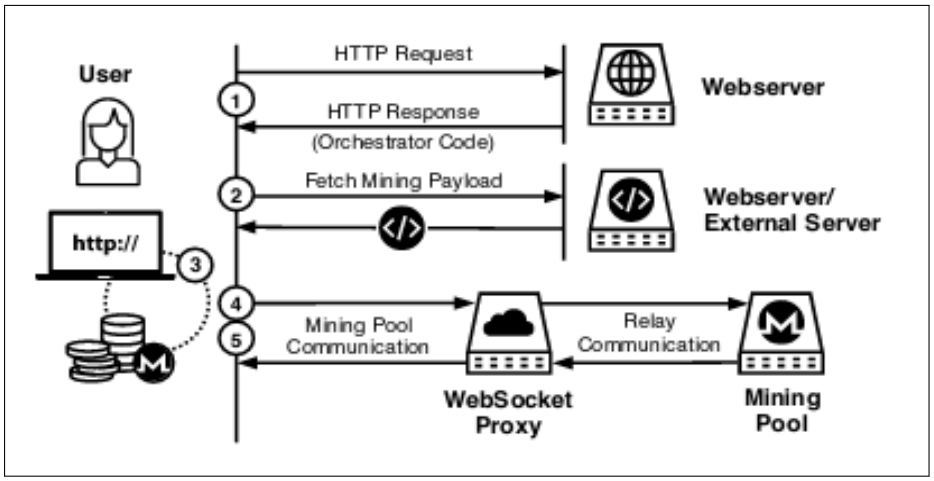


Fig. 2.7: Ejemplo de cryptojacking (Fuente: Publicación en ACM del proyecto Minesweeper [12]).

establecer su prevalencia y rentabilidad. Por último, realizan una descripción del marco ético para considerar si debe clasificarse como un ataque o como una oportunidad comercial. Por otro lado, hacen sugerencias para la detección, mitigación y/o prevención de la minería basada en el navegador que no requiere de consentimiento del usuario.

Otra publicación que aborda esta temática, fue la presentada en octubre de 2018 en ACM Computer and Communications Security (2018), que se denominaba "How You Get Shot in the Back: A Systematical Study about Cryptojacking in the Real World" [13]. En este estudio presentaban el proyecto CMtracker [66], en el cual presentaban una forma especial de identificar el cryptojacking. La forma en la que este proyecto detecta el cryptojacking es haciendo uso de la información referida a los perfiles del procesador en el navegador. Es importante este proyecto, dado que haremos uso de ciertas partes de su proyecto, modificándolas para el IsKrawler Analyzer.

La primera de las técnicas que utiliza es la búsqueda de operaciones hash en el procesador. Consiste básicamente en buscar dentro de los nodos de las funciones que se obtiene con el crawler. Buscando alguna de las librerías hash más utilizadas como "cryptonight" o "sha256". Por otro lado, se busca la cantidad de tiempo que pasa una página web realizando operaciones de hash. Normalmente, las páginas web no suelen realizar muchas operaciones de hash, por lo que si se supera un tiempo designado, nos indicará que puede ser actividad maliciosa.

El otro análisis de este estudio se basa en la pila del navegador, en la que si se repite una cadena de llamadas periódicamente, y además, esta cadena de llamadas ocupa más del 30 % del tiempo de ejecución se entenderá que este sitio web en cuestión está realizando actividades maliciosas de cryptojacking. En la figura 2.8 se muestra el ejemplo de la arquitectura del proyecto CMTracker y los dos análisis que realiza para determinar si una web realiza estos procesos maliciosos o no.

Una de las publicaciones, a mi juicio de las más relevantes, y que será la última para describir el estado del arte respecto a esta temática, es la presentada en octubre de 2018 en ACM Computer and Communications Security (2018), al igual que la anterior, llamado "MineSweeper: An In-depth Look into Drive-by Cryptocurrency Mining and Its Defense" [12]. Algunas partes de este proyecto,

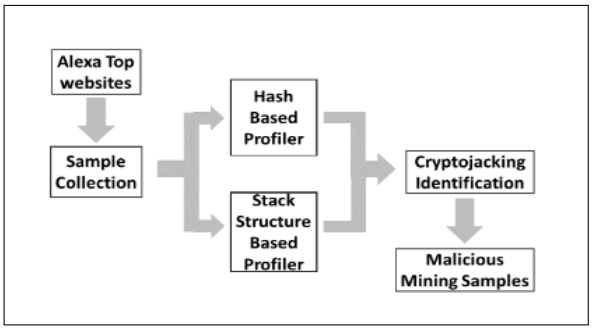


Fig. 2.8: Arquitectura de CMTracker (Fuente: Publicación en ACM del proyecto [13]).

dado la complejidad y lo relevante de este estudio, están replicadas y adaptadas en el IsKrawler Analyzer. Este estudio está realizado en Alexa top 1 millón, y se identificaron 1735 sitios web que realizaban cryptojacking, además de 20 campañas activas. El estudio que realizaron, demostró que siempre se basaban en el Algoritmo de CryptoNight para minar. Para llevar a cabo este estudio, desarrollaron el proyecto llamado como la publicación, es decir, Minesweeper [16].

Este proyecto, cuenta con una gran cantidad de análisis para averiguar si una página web realiza este tipo de actividad maliciosa o no. Primeramente intentan detectar palabras clave en los JavaScript, como la de los ejemplos que hemos visto, y además detectar si están haciendo uso de los ficheros WebAssembly. Por otro lado, detectará las comunicaciones mediante WebSockets y hará uso, al igual que en los JavaScript, para detectar palabras claves de la comunicación y determinar si se trata del protocolo Stratum o no.

Estos análisis nos permiten hacer un análisis superficial de la página web para detectar si lo realizan o no. Es un análisis superficial debido a que existen técnicas para evadir esto, como código empacado, código en Charcode, ofuscación de nombres, inyección de código muerto, aleatorización de las URL, protocolo Stratum codificado en base64 (detectado en 174 sitios), trucos antidepuración (detectado en 139 sitios) o reducción de la CPU a menor del 25 % (detectado en 12 sitios). Lo que les ha permitido identificar campañas, ha sido el “Site-key” o “CLIENTE-ID” que se encuentra en el orquestador y por otro lado el proxy con el que se realizaba la conexión WebSocket.

Asimismo, detectaron que las características comunes que compartían todas las páginas maliciosas que realizaban cryptojacking era que estaban basadas en el algoritmo CryptoNight y por lo tanto, minaban una moneda acorde a ese algoritmo, como puede ser Monero. Por otro lado, que este algoritmo solía estar implementado en los ficheros WebAssembly, que realizaban comunicaciones mediante WebSockets y la gran cantidad de WebWorkers en uso.

Por otra parte, el análisis que aportaba más a esta publicación, era el análisis que realizaba sobre los ficheros Wasm para detectar el algoritmo CryptoNight. Este algoritmo es una prueba de funcionamiento (PoW) propuesta en 2013. Su proyecto se aprovecha de dos características muy importantes, la primera es, la diversidad de funciones estándar de criptografía que utiliza, y la segunda es que se trata de un algoritmo que hace uso de mucha memoria de cache. Se puede observar estas dos características en la figura 2.9.

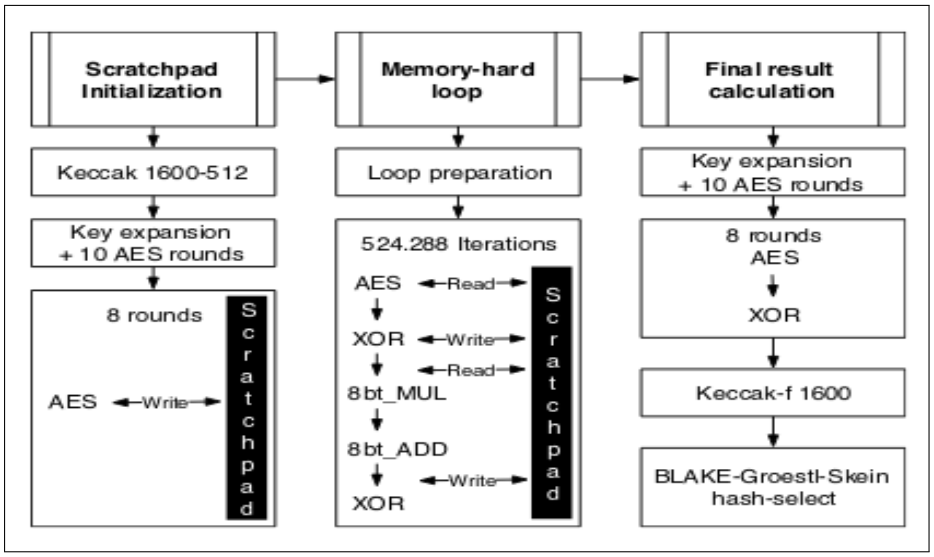


Fig. 2.9: Algoritmo CryptoNight (Fuente: Publicación en ACM del proyecto Minesweeper [12]).

El análisis que realiza MineSweeper sobre los ficheros Wasm, empieza haciendo uso de WABT [67] para descompilar estos ficheros. Posteriormente identificar las operaciones criptográficas (XOR, shift y operaciones de rotación) que se encuentran dentro de bucles, contando el número de bucles y la cantidad de operaciones criptográficas. Después, gracias a lo recopilado en el anterior paso, debemos identificar si alguna de las funciones se corresponde a alguno de los algoritmos de CryptoNight (Keccak, AES, Blake-256, Groestl-256 o Skein-256). Esta es una de las soluciones que más ataja la raíz del problema de las publicaciones vistas en conferencias.

Por otro lado, la segunda opción se trataba de monitorizar los eventos (Cargar evento o guardar evento) de la cache de la CPU. Previamente se comprobó la actividad de la cache con todo tipo de actividades como juegos en JavaScript, sin actividad, juegos en WebAssembly, videojuegos y mineros en Wasm. Los resultados proporcionaron que tanto utilizando la mineria con toda la potencia de la CPU como reduciendo su actividad al 20 %, los eventos que se daban en la cache L1 del procesador eran muy superiores en la minería al resto de eventos en las demás actividades.

Por último, hicieron uso del parámetro `--dump-wasm-module`, el cual no está casi documentado, en Chrome para descargar los ficheros WebAssembly cargados en la navegación de webs. Del millón top de Alexa que navegaron (Solo la página inicial) se encontraron con 748 ficheros Wasm, de los cuales 40 eran únicos y de esos 40, solo 4 determinaron que eran benignos.

2.2.3. Seguimiento web (Tracking)

El tracking o seguimiento web, es una técnica bastante extendida en Internet, en la que se recopilan los datos de los usuarios para realizar publicidad en línea, personalización del contenido o autenticación del usuario. En términos generales, el seguimiento web permite que sitios web de terceros (third-party) o los propios sitios, conozcan el historial de navegación de los usuarios y la configuración de navegación para ese fin. Para este estado del arte, se hará bastante uso de la publicación “The Web is Watching You: A Comprehensive Review of Web Tracking Techniques

and Countermeasures" [68].

Estas técnicas, en algunos casos se pueden usar para mejorar la experiencia de los usuarios y mejorar su navegación en Internet. Sin embargo, dado que a veces implican la recuperación de datos del usuario, incluso sin su consentimiento, pueden considerarse una violación de la privacidad en algunas ocasiones. Las empresas de publicidad web se niegan a reconocer el seguimiento web como una amenaza e incluso lo defienden públicamente argumentando que tiene una gran relevancia en la economía de internet [69].

Las técnicas de seguimiento web (Web-tracking) suelen dividirse en dos grandes categorías, las que mantienen un estado o las que no, dependiendo de si requieren o no almacenar datos en la máquina del usuario para funcionar correctamente. Normalmente, se suele considerar que los métodos de seguimiento sin estado son más difíciles de mitigar y bloquear porque evitan fácilmente las contramedidas más comunes contra el seguimiento, como la navegación privada o la eliminación de cookies [70]. Muchas publicaciones [70] [71] [72] [73] [74] han estudiado la prevalencia de diferentes tipos de rastreo web y huellas digitales. Estos trabajos dilucidaron que la práctica del seguimiento web es muy habitual y que no solo se utiliza en formas poco dañinas como el análisis web, sino también técnicas bastante avanzada de reconocimiento o de huellas dactilares (fingerprint) como son la prueba de fuentes o el reconocimiento de canvas.

Las técnicas de seguimiento que guardan estado utilizan los métodos disponibles para guardar información en la máquina del cliente. Uno de estos ejemplos son las Cookie, las cuales estaban diseñadas para permitir almacenar datos del sitio web, para recuperarlas cuando el usuario regresaba a la página, para mantener el estado de la navegación, que de otro modo se perdería, aumentando así la usabilidad. El problema, es que una web, está compuesta por la página web principal y también por los recursos de terceros, que tienen la capacidad de incluir cookies juntos con su recurso proporcionado. Es por ello, que si este servicio de terceros proporciona este mismo recurso a varios sitios web, puede realizar un seguimiento del usuario muy sencillamente. Este comportamiento, se llama cookies de terceros y es una de las técnicas más extendida para el seguimiento web en las técnicas que guardan el estado.

El otro tipo de seguimiento, es el que no requiere de ninguna información en la máquina de los usuarios. Estas técnicas sin estado permiten a este tipo de compañías eludir el modo de navegación privada y todo tipo de regulaciones legales existentes. Actualmente, hay diversas características para poder identificar únicamente a los dispositivos como pueden ser distintos aspectos del navegador. La identificación mediante JavaScript se realiza inspeccionando los recursos accesible del navegador, como por ejemplo, información respecto al proveedor del navegador, la versión, los complementos o plugins y tipos MIME compatibles, información del sistema operativo, etcétera. Otra característica utilizada comúnmente es la información relativa las resoluciones del monitor del usuario, así como la profundidad de color y píxel. Por otra parte, también se puede recopilar información abusando de la función de color de los enlaces visitados, recopilando la lista de fuentes instaladas en el dispositivo o calculando las diferencias que existen mediante elementos creados con el elemento canvas HTML.

Han surgido varias contramedidas contra este seguimiento web masivo gracias a las publicaciones

científicas. Una propuesta importante para limitar el seguimiento web es la iniciativa DoNotTrack (DNT) [75] que permite a los usuarios expresar su disposición a evitar ser rastreados. Otra línea de trabajo es el uso de navegadores personalizados. Como por ejemplo, los presentados en conferencias, Privaricator [76] o TrackingFree [77]. En los siguientes párrafos se describirán técnicas de seguimiento web y contramedidas.

Existen otras formas de categorizar una técnica de seguimiento web pero dos formas muy utilizadas [72] son, dependiendo la funcionalidad de la técnica o por el comportamiento que este tiene. Dependiendo de su funcionalidad se considera que puede ser: anuncios de terceras partes, anuncios con ventanas emergentes de terceras partes, redes de anuncios de terceras partes o widget/artilugio social de terceras partes. En cambio dependiendo de su tipo de comportamiento se definen los siguientes:

- **Comportamiento analítico**: Rastrean al usuario dentro de un sitio específico y sirve para un motor de analíticas de cada sitio web.
- **Comportamiento Vanilla**: Se trata de una posición de tercera parte que almacena información del usuario para poder rastrearlo a través de distintos sitios.
- **Comportamiento forzado**: Este comportamiento obliga a los usuarios a visitar su dominio directamente, ya sea por ventanas emergentes o redirecciones.
- **Comportamiento referido**: Este comportamiento, se basa en un rastreado tipo Vanilla, forzado o personal, para filtrar identificadores únicos. en lugar de en su propio cliente, para rastrear a los usuarios en todos los sitios.
- **Comportamiento personal**: Este rastreador es visitado directamente por el usuario en otros contextos.

Respecto a las contramedidas para mitigar el tracking web, se pueden utilizar varias configuraciones de navegador, aunque, obviamente, la solución más efectiva y simple, es la de deshabilitar JavaScript ya que la mayoría de técnicas descritas usan o dependen en alguna manera de ello, pero nos afectaría muy negativamente a la navegación. Otra opción, es la de utilizar los navegadores privados, para evitar almacenar información y caché de los sitios web visitados. Por otra parte, para evitar el reconocimiento sin estado, los más sencillo es deshabilitar ciertos conjuntos de fuentes específicos, deshabilitar las cookies y usar técnicas de listas negras de dominio y de scripts. Aún con todas estas acciones, no podremos evitar todo tipo de técnicas de rastreo web dado que las más avanzadas consiguen evitarlas.

Capítulo 3

Objetivos y alcance

Este capítulo está formado por dos secciones. La primera desarrollará los objetivos que ha tenido la realización de este proyecto y el por qué se ha decidido emprender este y no algún otro. En la segunda sección, se tratará el alcance que ha tenido el proyecto para especificar que abordará y que no.

3.1. Objetivos

Los objetivos de un proyecto son muy importantes para decidir emprender a hacerlo, y suponen una gran motivación en caso de tener unos objetivos que te sean de exigentes. Es primordial la elección de unos objetivos que sean a la vez exigentes e inspiradores pero que sean realistas ajustados a todo tipo de factores, como puede ser, el tiempo. Por ello, es un punto de gran relevancia dentro de un proyecto, como puede ser, en este.

En este apartado se explicarán los objetivos que se plantean en la realización del proyecto. Para ello se ha optado por dividir los objetivos en principales, los cuales son los objetivos primordiales a la hora de emprender el proyecto, y los relegados a secundarios, que son objetivos con una menor relevancia pero que también han ayudado a decidirse a realizarlo.

3.1.1. Principales

En este apartado se engloban los objetivos principales, los cuales han sido argumentos de peso para elegir este proyecto frente a otros posibles desarrollos. El objetivo más importante y exigente de este proyecto ha sido el de poder construir una herramienta completa, que complementándose con el crawler, ya desarrollado, permitiera un análisis integral y extenso de una página web. Este objetivo es exigente pero ajustado a la realidad.

Otro objetivo principal, similar al primero, es que el analizador, no se limite a revisar y procesar los resultados del crawler sino que recopile toda la información que el IsKrawler no es capaz de obtener debido a que no está preparado para ello, dicho de otro modo, todo tipo de información que no se puede recoger solamente recorriendo la web, dado que se necesita comunicarte con otros

puntos de información y bibliotecas.

3.1.2. Secundarios

En esta sección, se describen objetivos que debido a su relevancia son objetivos secundarios, pero que aún con una menor importancia, siguen siendo objetivos de valor para el proyecto.

Debemos realizar un analizador que sea intuitivo y fácil de usar para los usuarios de la herramienta, pudiendo ser utilizada tanto por alguien con muchos conocimientos en la materia, como por alguien que no los tenga. Ofreciendo los resultados de la máximas formas posibles. Además el analizador debe estar desarrollado de una forma que sea sencillo de añadir, modificar y eliminar secciones sin que pierda su validez y siendo fácil de mantener. Por otra parte, tiene que estar diseñado para que un futuro si se le quiera dar un portal web, no sea muy complejo.

Otro objetivo para el proyecto, es que sea de calidad, sirviendo para poder hacer uso real de él y no quedándose como un mero proyecto fin de grado, como en muchas ocasiones ocurre, es decir, que tenga un uso más allá del de la propia asignatura. El analizador debe de ser capaz de sintetizar todos los análisis y técnicas realizadas en un resultado para facilitar al usuario su visualización.

El analizador debe de ser óptimo y recabar toda la información y resultados en un tiempo considerablemente bajo. Además, el analizador debe de ser polivalente y flexible dependiendo el fin que se le quiera dar al resultado del análisis, permitiéndonos versatilidad a la hora de elegir la funcionalidad y la salida de los resultados, dependiendo por ejemplo, si es un análisis a un sitio web o a una enorme cantidad de sitios webs. Por otra parte, el desarrollar este proyecto también tiene como objetivo secundario el aprender de qué forma funcionan los estudios en las conferencias y toda la investigación entorno a la seguridad web, aplicando los estudios analizados en nuestro proyecto.

Otro objetivo secundario es el de poder evaluar esta infraestructura en un entorno real, con páginas webs reales y comprobando su desempeño. Dentro de este objetivo, se incluye, como forma de especificar el objetivo, un dato estimado a superar para la evaluación del analizador, es decir, el proyecto debe ser evaluado en como mínimo 250 páginas web analizadas.

3.2. Alcance

Este apartado tratará sobre el alcance del proyecto, dicho de otro modo, se describirán que procesos serán abordados dentro del proyecto y cuales no. En caso, de no abordar alguna cuestión, significará que esta, no entra dentro del proyecto y por lo tanto, no se llevará a cabo. Este alcance debe reunir todo lo necesario para alcanzar los objetivos, anteriormente mencionados, tantos los principales como los secundarios que perseguimos desarrollando este proyecto.

Se hará una recopilación de los últimos proyectos presentados en conferencias importantes de la seguridad web. Tras el mismo, se estudiarán, sintetizaran y recopilarán ideas para aplicar a nuestro

proyecto.

Se desarrollará un analizador, que se llamará IsKrawler Analyzer, de la información que nos proporciona el crawler, tanto de información de la navegación de la página como de los recursos que la página utiliza. Para ello, se diseñará y desarrollará este, de forma modular, para facilitar los cambios en él. Por otra parte, se aplicarán técnicas extraídas de documentos de conferencias de investigación consiguiendo con esto, una solución más amplia pero a la vez utilizando estrategias novedosas. Además el analizador haciendo uso de bibliotecas de Python y de otras APIs, será capaz de recopilar información la cual no es capaz de recoger el crawler con la simple navegación de la página.

El IsKrawler Analyzer deberá de ser flexible permitiéndote elegir la salida de los resultados como puede ser una salida por pantalla o un fichero HTML, el cual es la opción más intuitiva y sencilla, o para grandes cantidades de datos como es una base de datos. Por otra parte, se hará uso de diagramas para poder visualizar mejor alguno de los análisis y resultados.

Se aplicarán cambios y adiciones en el crawler, estudiando su funcionamiento para que aún añadiendo cosas tenga un correcto funcionamiento. Estas adiciones en el crawler, serán consecuencia del estudio de los documentos y de las técnicas que se quieran utilizar en el analizador.

Finalmente, el proyecto también abarcará un periodo de evaluación de la herramienta completa sobre datos reales. Para ello se lanzará el crawler nada más realizar las modificaciones pertinentes en el mismo. Y tras el desarrollo del analizador, se lanzará el mismo sobre los datos recopilados por el IsKrawler. El periodo de evaluación analizará como mínimo 250 páginas webs, y dependiendo del tiempo del que se disponga se aumentará la cifra.

También se evaluará la herramienta en tiempo y rendimiento tratando de identificar si se encuentra en unos parámetros asequibles, en caso de no estarlo, se añadirán parámetros para poder acelerar el análisis desactivando los módulos identificados. Por otra parte, no entraŕa dentro del proyecto la puesta en marcha de la herramienta en una infraestructura de producción. Tampoco entrará dentro del alcance, el diseñar e implementar una web para recibir y devolver los resultados.

Capítulo 4

Desarrollo del proyecto

En este capítulo se describirá detenidamente todo el proyecto. Para ello, se ha dividido el capítulo en dos secciones, la primera es Tecnologías y herramientas que se han usado, dicho de otro modo, se explicarán todas las utilidades y tecnologías que se han necesitado durante el transcurso del mismo, explicando de qué forma ha hecho falta, de qué forma se ha empleado y cuales son sus otras posibles utilizaciones.

Por otra parte, el capítulo de Arquitectura del proyecto, que ahondará en el proyecto explicando en qué ha consistido el proyecto, desmigando módulo por módulo todo el contenido del mismo.

4.1. Tecnologías y herramientas

4.1.1. Python

Es el lenguaje de programación que se ha elegido para desarrollar el analizador. Los motivos principales para elegir esta solución frente a otros, es la gran cantidad de bibliotecas que existen de las cuales puedes hacer uso, además de su legibilidad del código y su facilidad para realizar procesos muy complicados con pocas líneas de código. Por estas razones, el lenguaje en el que se ha desarrollado el IsKrawler Analyzer ha sido este.

Como se ha comentado, uno de los puntos fuertes de este proyecto es la infinidad de bibliotecas que podemos utilizar. Durante este proyecto, se hace uso de una gran cantidad de bibliotecas y en este punto vamos a tratar de recopilarlas todas.

La primera de ellas es la de Art [78] la cual nos permite crear un título presentable para el analizador haciendo uso del llamado arte ASCII. El segundo que se utiliza es el argparse [79] que nos permite fácilmente parsear los argumentos que se le indican al programa e incluso sacar un mensaje de ayuda con las opciones que podemos utilizar.

Dentro del análisis de la URL que se explicarán en la sección 4.2, hacemos uso de las bibliotecas pythonwhois [80] y geoip2 [81]. En el estudio de los ficheros HTML hacemos uso de Beautiful Soup [82]. También se hace uso de la biblioteca de jsbeutifier [83] para los ficheros JavaScript. Otra biblioteca que también se hace uso, es de la del proyecto Octopus [14]. No puede faltar, la

biblioteca para expresiones regulares como es re [84].

En los últimos procesos del analizador, hace uso de las bibliotecas Graphviz [85] para poder crear los diagramas de los análisis, SQLAlchemy [86] y Pyscopg2 [87] para hacer las conexiones con la base de datos de tipo PostgreSQL y Jinja2 [88] para poder crear y exportar los resultados a un formato HTML de forma sencilla.

Por último, se nombran dos herramientas de las cuales hemos hecho uso para comprobar primero, que no hay ningún tipo de vulnerabilidad en las bibliotecas utilizadas y segundo, comprobar que todas las licencias de cada una de las bibliotecas usadas no permiten hacer uso de ellas. Para el primer caso hemos hecho uso de Safety [89], que nos permite buscar esas vulnerabilidades. Para el segundo caso, el de comprobar las licencias, se hace uso del proyecto Scancode-toolkit [90].

4.1.2. PyCharm

PyCharm es el entorno de desarrollo integrado (IDE) que se ha utilizado para el IsKrawler Analyzer, realizado en Python. Se ha hecho uso de este entorno debido a las facilidades que nos aporta, como es la autocompletación de las palabras, el destacar la sintaxis del código, el poder incluir o quitar bibliotecas de forma sencilla, etcétera.

Este entorno de desarrollo tiene distintas licencias, teniendo que pagar para poder utilizar la más completa. En nuestro caso, gracias a la cuenta de estudiante de la universidad, hemos podido obtener una licencia completamente gratuita. Por último, destacar la facilidad de uso y el poder tener los directorios, el código y una consola en la misma ventana. Además analiza errores y te ofrece sugerencias, por todo esto, ha sido la elección para el desarrollo del IsKrawler Analyzer.

4.1.3. JavaScript

Este lenguaje de programación, es en el cual estaba desarrollado el IsKrawler. Por eso, para cada una de las adiciones realizadas en él, que se explicarán en la siguiente sección 4.2, se ha necesitado hacer uso de él. JavaScript es un lenguaje ligero e interpretado. Su mayor característica y por lo que está aumentando enormemente su popularidad, es debido a que es asíncrono y orientado a eventos. Además de hacer uso de esta tecnología en el crawler, se ha hecho uso de ella para, dentro del analizador, desarrollar la exportación de un HTML con una funcionalidad básica de pestañas que se abren o esconden.

4.1.4. NodeJS

NodeJS, según la definición de Wikipedia [91], es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor (no limitándose a ello) basado en el lenguaje de programación JavaScript, asíncrono, con E/S de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google. En otras palabras, y de manera muy simplificada, sirve para poder ejecutar código JavaScript fuera del navegador. Por ello, se hace uso de NodeJS para poner

el crawler, desarrollado en JavaScript, en ejecución.

4.1.5. VIM

Vim es una versión mejorada del editor de texto Vi, que se encuentra, por defecto, en todos los sistemas UNIX. Es un editor el cual nos permite realizar una gran cantidad de movimientos con atajos de teclado y que se puede extender fácilmente con plugins. Se ha hecho uso de este editor en todo lo relacionado con el IsKrawler debido a la facilidad de uso cuando tienes un poco de experiencia y los pocos recursos que consume.

4.1.6. HTML y CSS

El lenguaje de marcado HTML y las hojas de estilo CSS se han utilizado para la exportación de los resultados. En caso de indicarle con el parámetro "–html" significará que exportará los resultados con este formato. Esto permite visualizarlos sencillamente con el navegador. Se ha hecho uso de un CSS incrustado en el HTML para facilitar el desarrollo. Para la creación de HTML junto a CSS, como se ha comentado anteriormente, se ha hecho uso de Jinja2 [88].

4.1.7. DBeaver

DBeaver es una aplicación de software cliente de SQL y una herramienta de administración de bases de datos. Nos permite visualizar los datos de las tablas de la base de datos de Heroku [92] de una forma sencilla. Su interfaz es intuitiva y está para el sistema operativo GNU/Linux. Además, el diagrama 4.15 está realizado por el programa.

4.1.8. GitHub

En este proyecto se ha hecho uso de un sistema de control de versiones Git. Para ello, se ha hecho uso de un repositorio remoto alojado en Github. En el caso del proyecto, todas las interacciones con este repositorio se han hecho mediante la terminal. Se ha hecho uso continuamente de este repositorio para el control de versiones subiendo las adiciones o modificaciones del proyecto asiduamente.

4.1.9. JSON

Se trata de un formato de texto sencillo para intercambiar datos. Se ha hecho uso de este formato, tanto para la recopilación de los resultados del IsKrawler como para poder exportar los resultados del analizador. Es un formato bastante utilizado y nos permite esa facilidad de lectura y escritura.

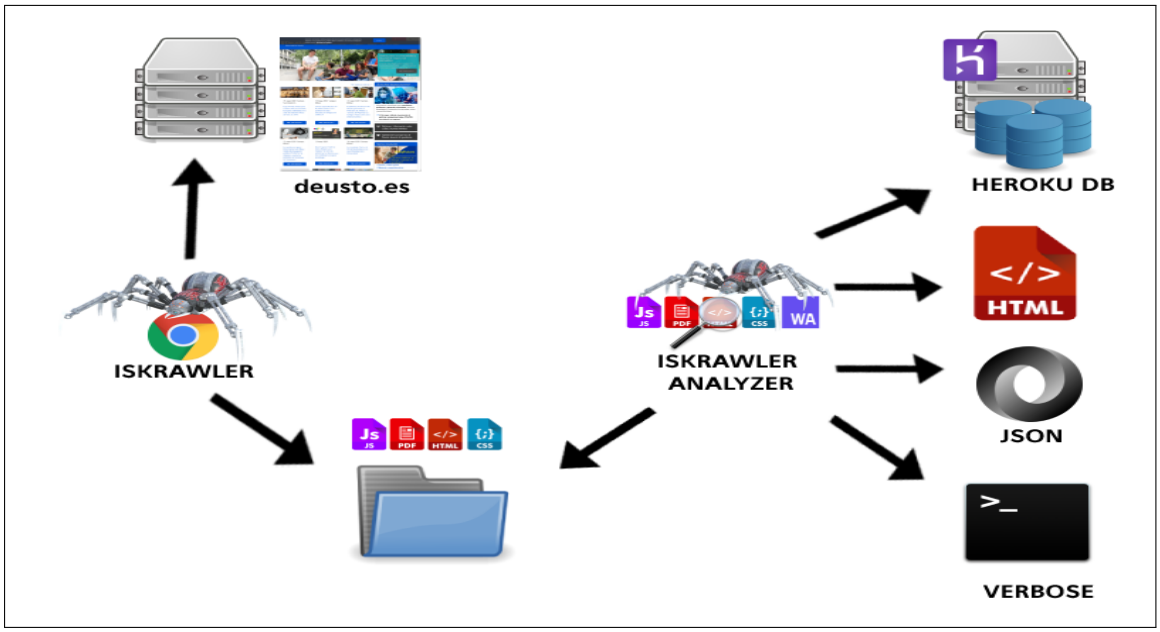


Fig. 4.1: Arquitectura del proyecto.

4.2. Arquitectura del proyecto

En esta sección se explicará la arquitectura que ha seguido el proyecto y el modo en el que funciona, sin entrar en detalle cómo se ha desarrollado específicamente cada uno de los apartados, dado que estarán explicados en cada uno de las siguientes secciones. Se ha incorporado la imagen 4.1 para representar visualmente la arquitectura y que sea más fácil de comprender los procesos que se llevan a cabo a la hora de analizar un sitio web. La disposición de las flechas en el gráfico es debido a quién realiza los procesos y no pensando en el flujo de información, intentando que aún siendo menos sencillo de entender tuviera más lógica. Como se ha explicado anteriormente, el objetivo de este proyecto ha sido el de crear una herramienta completa que haciendo uso de crawler ya existente fuera capaz de analizar un sitio web de la forma más exhaustiva. Para ello, la arquitectura se dividiría en dos, en la primera parte entraría el IsKrawler, dicho de otro modo, el crawler ya existente, el cual es el encargado de ir recorriendo los sitios webs y en la segunda parte, y haciendo uso de los resultados del primer proceso, realizar un análisis íntegro del sitio web, emitiendo sus resultados de la mejor forma posible.

El proceso que se lleva a cabo empezaría con el crawling de la página web, en este caso, el IsKrawler recorrería la web de deusto, guardando toda la información y recursos del sitio (HTML, JS, CSS...) en una carpeta del sistema. Este crawler tiene una gran cantidad de funcionalidades pero simplemente pasándole la URL de un sitio web la recorrería en unos segundos. Este crawler ya estaba desarrollado pero se realizaron modificaciones, que se explicarán en los siguientes apartados más a fondo, para recabar otros tipos de información que no recogía hasta la fecha.

Posteriormente, se ejecutaría el analizador, llamado IsKrawler Analyzer, indicándole donde se encuentra la carpeta de los resultados. El IsKrawler Analyzer ejecutaría un análisis exhaustivo sobre el sitio web y toda la información que proporciona el crawler emitiendo tras ello, un resultado. El analizador tiene implementado 4 distintas vías para emitir los resultados, la primera y una de las

más sencillas y normales, es que te lo vaya reportando por la terminal los resultados del análisis que has llevado a cabo, dado que en hay muchas personas, las cuales puede que no este familiarizados con la terminal o prefieren otros métodos más visuales se han desarrollado otras vías para la emisión de los resultados. La segunda vía, pensada para esas personas que prefieren verlo de una forma más visual o para casos en los que necesiten añadir los resultados en algún informe, se ha desarrollado que emita la solución por HTML, permitiendo así poder observarlo más sencillamente con el navegador. Esta segunda forma, igual es una forma de emitir los resultados no tan usual, pero que herramientas como puede ser NMAP, aunque no sea una opción muy conocida, lo implementan debido a esa facilidad que aporta a la hora de revisar los resultados y poder incrustarlos en otros documentos.

La tercera forma es que el analizador emite los resultados en un JSON, esto está pensado principalmente para personas que quieran hacer uso de estos resultados con otros tipos de programas o ejecutables, facilitándoles el trabajo de la recogida de los resultados. Esta forma de exportarlo a un json incluye toda la información del análisis y permite fácilmente poder trabajar con ella o incluso en un futuro poder servila como una API. La última vía, es la pensada y centrada un poco más en la investigación, en la que se ejecuta el IsKrawler y el analizador durante un tiempo determinado recorriendo sitios web y en el que te permite de una forma sencilla tener toda la información acumulada y en caso de necesitarlo poder después acudir a los resultados en HTML para investigar más a fondo un caso. Esta forma, está pensada para poder recopilar una gran cantidad de información y poder hacer estudios grandes, teniendo la información más relevante y en la cual estemos interesados en la base de datos. En esta última vía, se ha hecho uso de una base de datos PostgreSQL de Heroku [92] la cual nos permite hasta cierta cantidad de datos hacer uso de ella gratuitamente, pero en cualquier caso podríamos implementar un servidor de base de datos y simplemente cambiando un par de parámetros funcionaría de la misma manera.

4.2.1. El sistema de crawling

En este apartado se comentará lo relativo a las adiciones hechas en el crawler (IsKrawler), el cual ya estaba desarrollado pero para algunas de las funcionalidades que se realizan en el analizador (IsKrawler Analyzer) era necesario realizar modificaciones y/o añadidos en algunas de sus partes. Estas explicaciones se limitarán únicamente a las adiciones realizadas para el proyecto, debido a que el resto del programa actualmente no es público. Aun así, el crawler está desarrollado en JavaScript y hace uso del entorno de ejecución de NodeJS para poder ejecutarlo.

La interfaz de la cual hace uso para poder realizar el crawling se basa en el protocolo de depuración del navegador Chrome, lo que nos permite que sea un crawling a bajo nivel recopilando todo tipo de información posible, y el proyecto se puede encontrar en Github [8]. Las adiciones realizadas al crawler se pueden agrupar en tres ámbitos distintos, el primero es la capacidad de que recoja los ficheros WebAssembly (wasm), lo segundo es que recoja todas las conexiones realizadas mediante WebSockets y por último el que sea capaz de realizar los perfiles del procesador en el navegador.

A la hora de modificar el crawler para que fuese capaz de reconocer los ficheros WebAssembly y que los recogiera por defecto fue bastante complicado comprender el por qué de algunas cosas.

Por ejemplo, el caso curioso que ocurre con los WebAssembly a la hora de reconocerlos desde la interfaz. Haciendo uso de esta interfaz, los ficheros "wasm" son reconocidos por el motor de renderizado, y a su vez por esta interfaz, como ficheros "Fetch". El motor de renderizado de Chrome se llama Blink y por algún motivo que desconozco los reconoce así, pudiendo verse en la documentación del protocolo de depuración de Chrome [93] como no hay ningún tipo de información respecto a estos ficheros, ni cuales se engloban dentro de esta categoría a parte de los WebAssembly.

Por estos problemas, se realizo una etapa de crawling tratando de encontrar páginas que tuvieran algún tipo de recurso que no fueran "wasm" que se englobaran en la categoría "Fetch" además de una gran cantidad de tiempo buscando en la escasa información que había. Por otra parte, se hizo una pequeña modificación para que a la hora de recoger los recursos, detectara si estabas codificados en Base64 con una expresión regular, y en caso de estarlo, decodificarlo. Esta adición, en la cual el crawler por defecto descargara los ficheros "Fetch" no tuvo más complicación que las anteriores mencionadas.

Otra modificación realizada fue la de reconocer y recoger las conexiones mediante la tecnología WebSocket. La tecnología WebSocket proporciona un canal de comunicación bidireccional y full-duplex sobre un único socket TCP. Está diseñada para ser implementada en navegadores y servidores web, pero puede utilizarse por cualquier aplicación cliente/servidor [94]. Es una tecnología muy utilizada para la actividad maliciosa del Cryptojacking o para otras benignas como páginas web para mantener conversaciones. Los métodos implementados en el crawler hacen uso de la interfaz y están descritos en la documentación [95].

En esta modificación se desarrollo dos formas de funcionamiento dependiendo de un parámetro a la hora de ejecutar el crawler, y es que dependiendo de si se ha indicado este parámetro o no, recogerá el contenido de todos los frames o solamente recogerá los frames enviados y recibidos sin guardar consigo los mensajes dentro de los frames. Para este apartado, y dando por hecho que el parámetro está indicado para recoger toda la información posible, por cada conexión WebSocket creada, además de la información recogida de la conexión , es decir la url y otros datos, se crean dos listas vacías, una para los frames enviados y otra para los frames recibidos, que se irán rellenando según se estén enviando o recibiendo. Además dentro de los frames, tanto enviados o recibidos, se realiza una comprobación con una expresión regular, para determinar si están codificados en base64 o no y en caso afirmativo, se guardan los frames decodificados. Finalmente, en el caso de que se encuentren, lo que tenemos es una lista de conexiones o objetos WebSockets, los cuales contienen información relativa a la conexión y dos listas para los frames enviados y para los recibidos. A continuación se puede observar la parte principal del código, quitando partes en las que se entrelaza con el propio crawler, añadido:

```
// ----------------------------
// Network.webSocketCreated
// ----------------------------
/**
 * webSocketCreated
 *
```

```
 * @param {object} params - Received parameters from the event
 * @param {string}      params.requestId - Request identifier.
 * @param {string}      params.url
 * @param {object}      params.initiator - Optional
 * @param {object} currentStatus
 * @param {object} responseData - Object for the output Json
 */
function webSocketCreated(params, currentStatus, responseData) {
        let webSocket = {};
        webSocket.requestId = params.requestId;
        webSocket.url = params.url;
        webSocket.initiator = params.initiator;

        //If the flag for a exhaustive result
        if(currentStatus.webSocketFlag){
                webSocket.frameReceivedNumber = 0;
                webSocket.frameSentNumber = 0;
                webSocket.frameReceived = [];
                webSocket.frameSent = [];

                responseData.webSockets[webSocket.requestId] =
                   ↪ webSocket;
        }else {
                webSocket.frameReceived = 0;
                webSocket.frameSent = 0;

                responseData.webSockets[webSocket.requestId] =
                   ↪ webSocket;
        }
}

// ----------------------------
// Network.webSocketFrameReceived
// ----------------------------
/**
 * webSocketFrameReceived
 *
 * @param {object} params - Received parameters from the event
 * @param {object}      params.requestId - Request identifier
    ↪ String.
 * @param {number}      params.timestamp - Monotonically increasing
    ↪  time in seconds since an arbitrary point in the past.
 * @param {object}      params.response - WebSocketFrame response
    ↪ data.
 * @param {object} currentStatus
```

```
 * @param {object} responseData - Object for the output Json
 */
function webSocketFrameReceived(params, currentStatus, responseData
    ↪ ) {

        //With the flag appears the content of the frames.
        if(currentStatus.webSocketFlag){
                let frame = {}
                frame.timestamp = params.timestamp;

                //Check if is base64 response
                let base64regex = /^([0-9a-zA-Z+/]{4})*(([0-9a-zA-Z
                    ↪ +/]{2}==)|([0-9a-zA-Z+/]{3}=))?$/;

                if(base64regex.test(params.response)){
                    let buff = new Buffer(params.response);
                    let text = buff.toString('ascii');
                    frame.response = text;
                }else{
                    frame.response = params.response;
                }

                responseData.webSockets[params.requestId].
                    ↪ frameReceivedNumber += 1;
                responseData.webSockets[params.requestId].
                    ↪ frameReceived.push(frame);

        }else {
                responseData.webSockets[params.requestId].
                    ↪ frameReceived += 1;

        }
}

// ----------------------------
// Network.webSocketFrameSent
// ----------------------------
/**
 * webSocketFrameSent
 *
 * @param {object} params - Received parameters from the event
 * @param {object}      params.requestId - Request identifier
    ↪ String.
 * @param {number}      params.timestamp - Monotonically increasing
    ↪  time in seconds since an arbitrary point in the past.
```

```
 * @param {object}      params.response - WebSocketFrame response
    ↪ data.
 * @param {object} currentStatus
 * @param {object} responseData - Object for the output Json
 */
function webSocketFrameSent(params, currentStatus, responseData) {
        //With the flag appears the content of the frames.
        if(currentStatus.webSocketFlag){
                let frame = {}
                frame.timestamp = params.timestamp;



                //Check if is base64 response
                let base64regex = /^([0-9a-zA-Z+/]{4})*(([0-9a-zA-Z
                   ↪ +/]{2}==)|([0-9a-zA-Z+/]{3}=))?$/;

                if(base64regex.test(params.response)){
                    let buff = new Buffer(params.response);
                    let text = buff.toString('ascii');
                    frame.response = text;
                }else{
                    frame.response = params.response;
                }

                responseData.webSockets[params.requestId].
                   ↪ frameSentNumber += 1;
                responseData.webSockets[params.requestId].frameSent
                   ↪ .push(frame);

        }else {
                responseData.webSockets[params.requestId].frameSent
                   ↪  += 1;
        }
}
```

Listing 4.1: Código añadido al crawler para registrar conexiones WebSocket.

La última adición al crawler, es que fuese capaz de recoger la información relativa a la perfilación del procesador en el navegador. Para ello se implementaron en el crawler los métodos que están explicados en la documentación del protocolo de depuración de Chrome [96]. Este método es de los implementados, uno de los que más entrelazado está con el código del crawler debido a que para habilitar el perfilado, darle comienzo y pararlo todo transcurre en momentos distintos del programa. Se habilita el perfilado en el mismo momento que se habilitan el resto de análisis que se van a realizar, se da comienzo al perfilado previo a navegar a la página y se da por terminado justo después de dar por finalizada la navegación de la página. Toda esta información se recopi-

la igual que en el caso de los websockets y que luego al dar por finalizado el crawling será exportada.

4.2.2. El sistema de análisis

Fig. 4.2: Logo del proyecto (IsKrawler Analyzer).

En esta sección se describirá el desarrollo relacionado con el analizador (IsKrawler Analyzer) y se desgranarán todos los tipos de análisis que el proyecto lleva a cabo, además del análisis global en el que se valoran los resultados de los distintos análisis llevados a cabo y un apartado final, en el que se explica de qué forma el IsKrawler Analyzer emite los resultados.

La estructura de la explicación de los analizadores está organizada por tecnologías o análisis comunes en los distintos módulos que componen un analizador para hacer más sencilla su comprensión, dicho de otro modo, el IsKrawler Analyzer se compone de distintos analizadores los cuales están formados por módulos en los que cada uno examina mediante una técnica distinta. En estos distintos apartados se ha tratado de explicar de la forma más concisa y legible el proyecto, tratando de alguna forma de convertir el código en párrafos de texto. Es importante también, que esta estructura utilizada está pensada para que una persona la cual no sepa como funciona el proyecto puede hacerse una idea fácilmente y a la vez para realizar un proyecto modular, en el que se puede ir ampliando y modificando de una forma sencilla para poder realizar un análisis lo más exhaustivo posible.

Por otra parte, este analizador ha sido desarrollado desde un sistema operativo GNU/Linux como es Debian y ha sido probado, además de Debian, en una máquina virtual en Ubuntu pero al estar haciendo uso de Python y de sus bibliotecas no tiene que haber ningún tipo de problema a la hora de ejecutarlo en otro tipo de entornos. Se ha utilizado Python en su versión 3.8.3, la cual es una de las más recientes. El IsKrawler Analyzer consta de alrededor de 6300 líneas y compuesto por alrededor de 50 ficheros. Se ha hecho uso de la herramienta Github para llevar un control de versiones y poder tener el repositorio en la nube de forma que en caso de tener algún problema o borrar algo que no debía, poder hacer uso de ella.

Es un analizador pensado para ser utilizado, tanto para una persona individual que quiera saber si una web es maligna, el cual sería el caso principal, como para que un desarrollador pueda buscar

algún tipo de fallo no esperado, para un investigador en el que realizando breves adiciones podría evaluar su teoría en la WWW o para una persona que está buscando posibles debilidades en la web. Por otro lado, el proyecto, una vez finalizada la entrega, se liberará dándole una licencia permisiva para que otras personas puedan entenderlo e incluso hacer aportaciones al mismo.

A la hora de ejecutar el IsKrawler Analyzer, nos permite seleccionar varios parámetros para elegir cómo queremos que sea nuestro análisis y cómo queremos que nos devuelva los resultados, para ello, se han desarrollado unas opciones en los que cada uno tiene una función y debemos de indicarlas al ejecutar el programa. Los parámetros son los siguientes:

- **Verbose** (`-v` / `--verbose`): Haciendo uso de esta opción nos sacará los resultados por pantalla.
- **Eliminar resultados anteriores** (`--rm` / `--rm-results`): Si utilizamos este parámetro borrara los resultados que pueda haber previamente en la carpeta resultados.
- **Analizar todos los resultados** (`--all` / `--analyze-all`): Indicándolo puedes lanzar el analizador sobre más de un crawling.
- **Deshabilitar el uso de VirusTotal** (`--dis-vt` / `--disable-vt`): Permite deshabilitar el uso de VirusTotal para aumentar la velocidad del análisis.
- **Exportar los resultados en formato HTML** (`--html`): Haciendo uso de este parámetro nos exportará los resultados en un HTML.
- **Exportar los resultados en formato HTML en la carpeta resultados y en /var/www/html/** (`--htmlvar`): Haciendo uso de este parámetro nos exportará los resultados en HTML en la carpeta resultados y en la carpeta predeterminada de apache2 para facilitar la visualización con el navegador.
- **Exportar los resultados a la base de datos** (`--db`): Si le indicamos esta opción guardará en la base de datos configurada los resultados.

4.2.2.1. Analizador de URL

En esta sección, se analizará toda la posible información que se puede sacar teniendo solo la url objetivo. La información que hoy en día podemos sacar teniendo solo la url es enorme, empezando por su fecha de creación, en donde se encuentra ubicada, que tipo de contenido tiene, es maliciosa según otros analizadores o en que puesto del ranking mundial se encuentra. Es por eso, que en este analizador, esta sección es una de las más importantes debido a toda esa información que nos ofrece teniendo solo ese simple dato. En los siguientes párrafos se explicarán los distintos módulos utilizados para analizar la url.

El primer módulo, posiblemente una de las técnicas más utilizadas, es el de hacer uso de la información que se encuentra mediante el protocolo WHOIS. Este es un protocolo TCP basado en petición/respuesta que se utiliza para efectuar consultas en una base de datos que permite determinar información sobre el propietario de un nombre de dominio o una dirección IP en Internet [97]. Para hacer uso de esta información desde el analizador, hacemos uso de la biblioteca para Python llamada pythonwhois [80], la cual nos permite recabar esta información de una manera sencilla.

Gracias a esta biblioteca, podemos obtener id del dominio de registro, la fecha de expiración del dominio, la fecha de creación del dominio, la fecha de actualización del dominio o el servidor WHOIS en el que se ha registrado. De toda esta información que nos ofrece el protocolo WHOIS, resulta de especial interés lo relacionado a las fechas, concretamente las fechas de creación y de vencimiento. Esto se debe a que en caso de ser un sitio web legítimo tiene una alta probabilidad de que haya sido creado hace varios años, esto es, resulta sospechoso si el dominio ha sido registrado el mismo año en el que se está analizando. Por otra lado, es común que al registrar un dominio, ya sea de una página personal o una empresa, se registre por un periodo superior a 1 año, teniendo en cuenta además las ofertas disponibles en los proveedores de registro de dominios. Entonces esta sería otra cosa a analizar. En la figura 4.3 se muestra una imagen de la información que nos muestra una simple búsqueda.

Fig. 4.3: Información WHOIS de una página web (Web: google.com).

Otro módulo muy importante, es el referido a la posición que ocupa a nivel mundial en el ranking de páginas web visitadas, que tipo de contenido por categoria contiene, cuantas visitas ha tenido el último mes. Para ello, en este analizador de hará uso de distintas formas, la primera mediante Alexa y la segunda mediante SimilarWeb. En la primera, se hace uso de Alexa, pero debido a que hace unos años cerraron la API pública y gratuita para hacer peticiones, se hará uso de la versión estática.

Alexa es una plataforma utilizada mundialmente que proporciona datos y análisis comerciales de tráfico web, su rango de tráfico global o ranking es una medida de cómo de popular es un sitio web en relación con todos los otros sitios en la web durante los últimos 3 meses. Este rango se calcula utilizando una metodología patentada que combina el promedio estimado de visitantes únicos diarios y su número estimado de páginas vistas durante los 3 últimos meses [98]. En nuestro caso, esta versión estática, es una API a la cual accedes y descargas un fichero “zip” con una ínfima cantidad de información comparada con el anterior servicio que nos ofrecía. Este fichero “zip” contiene un “csv” con la lista un millón de Alexa que es actualizada cada mes y medio más o menos. Esta lista simplemente contiene el ranking mundial del primer millón de páginas webs visitadas. Además para poder actualizar fácilmente esta información se ha creado un script en Bash el cual nos actualizaría

la lista con solo ejecutarlo.

```
#!/bin/bash

# Ruta al fichero
ALEXA=top-1m.csv

# Elimina el fichero antiguo y manda los errores al NULL
rm $ALEXA 2>/dev/null;
# Descarga el zip // Lo descomprime // Borra el zip
wget -q http://s3.amazonaws.com/alexa-static/top-1m.csv.zip;
   ↪ unzip top-1m.csv.zip; rm top-1m.csv.zip;
```

Listing 4.2: Script para actualizar Alexa Static [99]

Debido a que la información proporcionada por Alexa Static es poca y dado que se actualiza cada cierto tiempo, se decidió hacer uso también, de la información que nos ofrece la API de SimilarWeb [100]. SimilarWeb hace uso de cientos de fuentes y tipos de datos diversos, en los que se engloban cuatro grupos, los de medición directa, que son aquellos en los que los propietarios de sitios web y aplicaciones móviles comparten sus datos de primera parte con SimilarWeb directamente, los de las asociaciones, los cuales son datos de socios con millones de suscriptores que representan el mayor número de proveedores externos de cualquier persona en la industria, las fuentes de datos públicas, haciendo uso de tecnología patentada y motores de indexación que escanean continuamente los puntos de datos públicos, y por último, datos de comportamiento anónimos, basados en paneles agregados a nivel de sitio en lugar de a nivel individual, sin información de identificación personal. Es por ello, que para el proyecto hacemos uso de esta API para complementar la información del ranking de Alexa y aportar alguna información más como puede ser cuantas visitas tiene cada mes, la categoría y alguno más. A continuación podemos ver un ejemplo de json resultante, haciendo una búsqueda sobre el dominio “deusto.es”:


```
{
  "SiteName": "deusto.es",
  "TopCountryShares": [
    {
      "Value": 0.7471562092302807,
      "Country": 724
    },
    {
      "Value": 0.05860154199418899,
      "Country": 484
    },
    ...
  ],
```

```
  "Title": "universidad␣|␣deusto",
  "Engagments": {
    "BounceRate": "0.4465182195188896",
    "Month": "04",
    "Year": "2020",
    "PagePerVisit": "5.0744050285888544",
    "Visits": "862779.6313307486",
    "TimeOnSite": "334.63300764504356"
  },
  "EstimatedMonthlyVisits": {
    ...
    "2020-02-01": 581650,
    "2020-03-01": 812277,
    "2020-04-01": 862779
  },
  "GlobalRank": {
    "Rank": 54810
  },
  "CountryRank": {
    "Country": 724,
    "Rank": 1594
  },
  "Category": "Science_and_Education/Education",
  "CategoryRank": {
    "Rank": "2305",
    "Category": "Science_and_Education/Education"
  },
}
```

Listing 4.3: Ejemplo de resultado de SimilarWeb API.

Por lo tanto, este módulo, cumplimentando la información de los dos lugares, tanto de Alexa Static [99] como de SimilarWeb [100], nos permite completar mucha más información respecto al dominio y tener una visión todavía más amplia. Este módulo, es posiblemente una de las funcionalidades que más tiempo ha llevado, no por el desarrollo en sí, sino por la búsqueda de APIs o soluciones gratuitas y fiables. Hay que tener en cuenta que este puede ser uno de los módulos que más depende de fuentes externas, y sobretodo fuentes que en cualquier momento pueden dejar de dar este servicio. En caso de que alguna de las fuentes deje de dar este servicio, se buscarán alternativas para adaptar el proyecto.

Respecto a la categorización del sitio web, se hace uso, también, de otra API, para poder tener el resultado de dos lugares completamente distintos, y por otra parte, como hemos comentado anteriormente, puede que en algún momento alguna de estas falle y haya que buscarle un repuesto. En esta segunda categorización se hace uso una API de Symantec [101]. Esta API, nos ofrece la categorización de un sitio web con una simple petición POST incluyendo en el cuerpo de la petición la url, además de ser gratuita y pública.

Un módulo utilizado tanto en este analizador, como en alguno de los siguientes, es el de hacer uso de la API de VirusTotal [102], en este caso del apartado para la búsqueda de información respecto a una URL dada. Gracias a esta API, nos devuelve información de un total de 80 antivirus en los cuales te muestra si en alguno de ellos ha dado positivo o no, y en caso de dar positivo te devuelve la razón por la que da positivo.

Esta API de VirusTotal nos permite realizar 4 peticiones por minuto, por lo tanto, se hará el uso justo de la misma y se llevará una cuenta de las peticiones que se hacen. Es por ello, que en el proyecto se ha realizado para hacer uso de esta API de VirusTotal, además de los distintos métodos dependiendo si es una URL o un hash, un método para registrar la peticiones que se hacen, registrando el tiempo y guardándolas, comprobando cada vez que se quiera hacer una nueva petición si de alguna de las anteriores peticiones ha transcurrido ya el minuto, teniendo en todo momento la cantidad de peticiones que se están realizando por minuto, con un máximo de 4. En caso, de que las peticiones ya realizadas sea de 4, se irá comprobando a ver si alguna de las anteriores se hizo hace más de un minuto y en el momento que eso pase, se eliminará esa petición antigua y se realizará la nueva, añadiéndola a la vez a la lista.

Otro módulo incluido en esta sección, es el poder saber en qué parte del mundo está alojada esta página web, para ello, se hace uso de la biblioteca de Python [81], la cual añadiéndole los ficheros de información que se van a utilizar, en este caso en el proyecto se hace uso del fichero de sistemas autónomos (ASN) y del ficheros de ciudades, los cuales eran los únicos gratuitos en la web utilizada [103]. Haciendo uso de este módulo, obtenemos el id y la organización del sistema autónomo (ASN) que corresponda, se nos proporciona también el código ISO del país, el nombre de este, la latitud y la longitud, obviamente estos datos tienen un margen de exactitud de entorno 1000km. Este módulo está desarrollado porque normalmente, las webs maliciosas o de phising, suelen estar alojadas en países concretos debido a la flexibilidad que esto les permite a los atacantes. Esto no quiere decir, que todas las webs en países extraños sean maliciosas, ni el caso contrario, que todas las alojadas en países conocidos sean benignas, pero nos permite aportar más información al análisis.

A parte de todos los módulos anteriores, esta sección cuenta con otro apartado, en el que utiliza otro tipo de analizador web como es Webutation [104]. Webutation es una comunidad abierta sobre la reputación de sitios web que recopila los comentarios de los usuarios y la experiencia del cliente sobre los sitios web. Prueba sitios web contra spyware, spam y estafas con tecnología de escaneado inteligente en tiempo real. Webutation consulta información con Google Safebrowsing [105], antivirus webs que analizan sitios contra adware, spyware y virus, Web Of Trust (WOT) [106] que recoge revisiones sobre los sitios webs, Norton Safe Web [107], así como otros recursos de retroalimentación de sitios webs [108]. Webutation nos ofrece una puntuación sobre 100 que se obtiene del servicio, siendo las puntuaciones 90-100 indicadores de que es un sitio seguro, una puntuación de 70 significa que el servicio de Webutation no conoce el dominio consultado y una puntuación menor de 70 se debe a que el sitio es potencialmente peligroso. En la imagen 4.4 se muestra una consulta a esta web y el resultado que nos proporciona.

Otro tema que también se analiza en este apartado, es todo lo relacionado con las cabeceras, ya

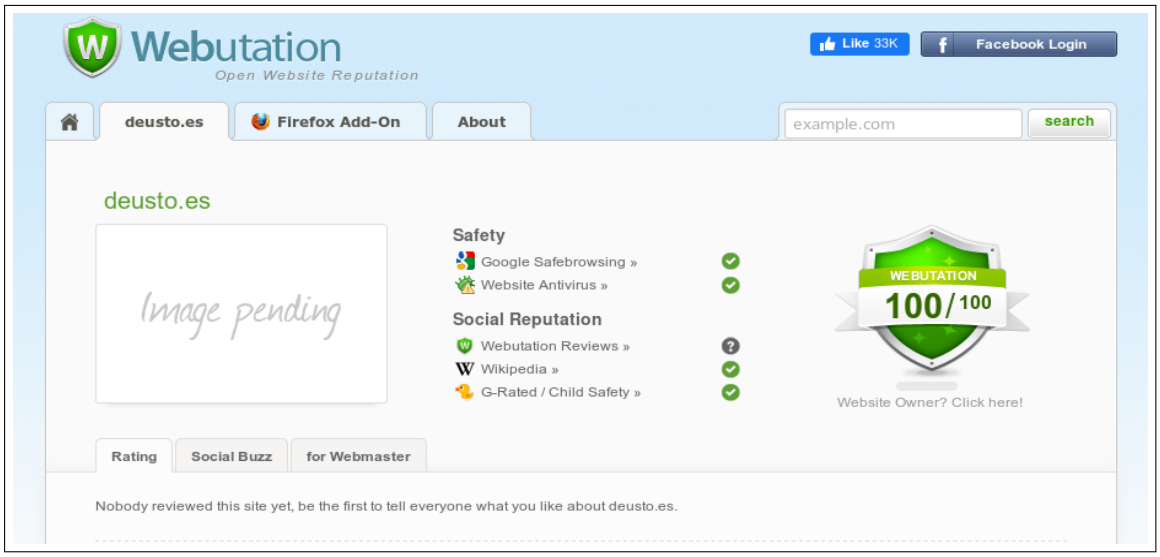
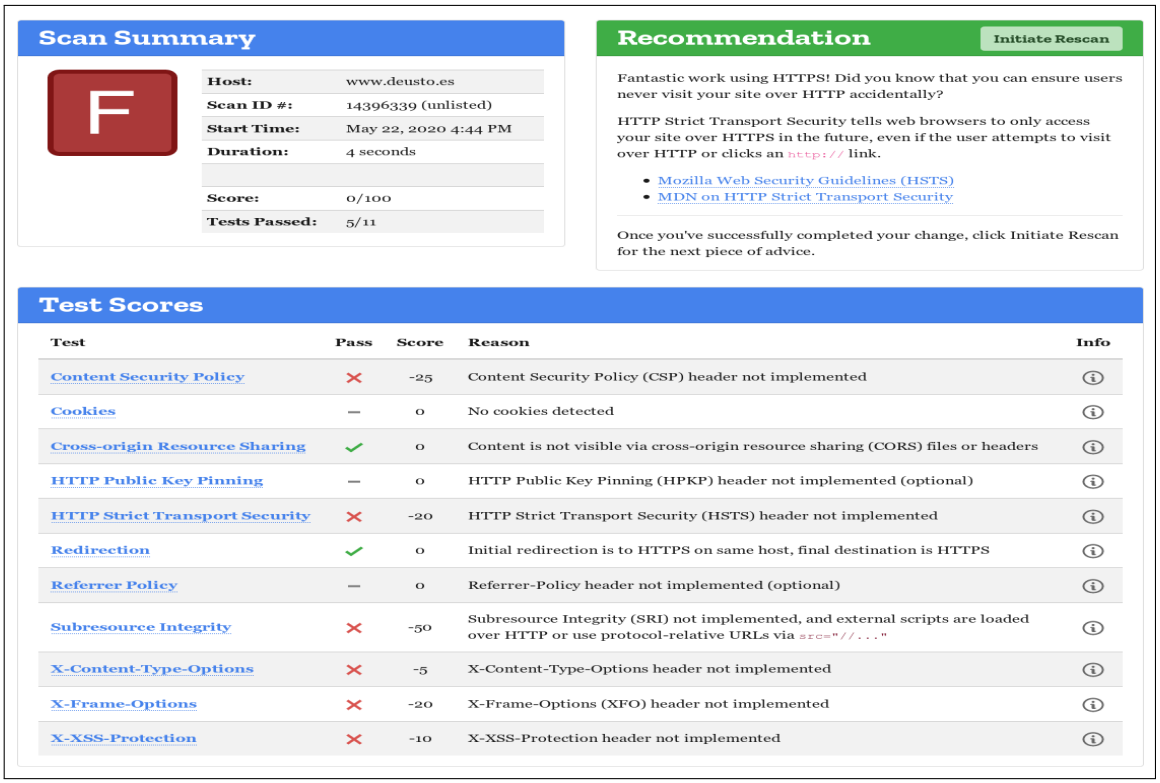


Fig. 4.4: Consulta en Webutation de una página web (Web: deusto.es).

sean las Content Security Policy (CSP) u otro tipo de protecciones para poder minimizar los vectores de ataque. Es por ello que el análisis no solo cubre esto desde un punto de vista de saber si una web es maliciosa, sino también el poder encontrar posibles fallas en las configuraciones de esta. Por todo esto, se hace uso de los proyectos de Mozilla Observatory [109], en el que uno está centrado en las cabeceras y el otro en configuraciones de TLS. En este caso, el analizador se conecta con las dos APIs del proyecto, una para el HTTP observatory y otra para el TLS observatory, y haría la petición recibiendo la respuesta con los resultados. Respecto al HTTP observatory nos ofrece un análisis sobre las CSP, las cookies, los Cross-origin Resource Sharing (CORS), el HTTP Public Key Pinning (HPKP), el HTTP Strict Transport Security (HSTS), las redirecciones, la cabecera Referrer-Policy, los Subresource Integrity (SRI), la cabecera X-Content-Type-Options, la cabecera X-Frame-Options y por último la cabecera de protección de XSS llamada X-XSS-Protection. Por lo tanto esto nos ofrece un gran análisis con sus descripciones y sus posibles soluciones. Para poder acompañar esta descripción del proyecto, podemos observar en la figura 4.5 una imagen de este análisis sobre la web de la universidad.

El otro análisis es relacionado a la seguridad en la capa de transporte en las conexiones HTTP, llamada comunmente TLS (En ocasiones se le llama SSL, pero realmente es el nombre de su versión antigua), en la que nos ofrece diversa información. Primeramente nos ofrece el nivel de compatibilidad del servidor, dicho de otro modo, si es compatible con versiones antiguas, si es flexible o si únicamente acepta las últimas versiones para el cifrado. Nos ofrece también para que dominios es utilizado, desde que fecha es válido, hasta que fecha es válido, que autoridad de certificación (CA) expidió el certificado y el algoritmo de firmado. Además, nos informa de que algoritmos de cifrado acepta el servidor incluyendo en una lista todos aquellos con los cuales podemos establecer una conexión con el servidor. Debido a la amplitud de la información que nos aporta no se adjunta una imagen descriptiva del análisis pero en cualquier caso la página web en la que hacer los análisis [109] y la API [110], la cual hace uso el proyecto para recabar la información, son públicas y de libre acceso.



Scan Summary

F

Host:	www.deusto.es
Scan ID #:	14396339 (unlisted)
Start Time:	May 22, 2020 4:44 PM
Duration:	4 seconds
Score:	0/100
Tests Passed:	5/11

Recommendation Initiate Rescan

Fantastic work using HTTPS! Did you know that you can ensure users never visit your site over HTTP accidentally?

HTTP Strict Transport Security tells web browsers to only access your site over HTTPS in the future, even if the user attempts to visit over HTTP or clicks an http:// link.

- Mozilla Web Security Guidelines (HSTS)
- MDN on HTTP Strict Transport Security

Once you've successfully completed your change, click Initiate Rescan for the next piece of advice.

Test Scores

Test	Pass	Score	Reason	Info
Content Security Policy	✗	-25	Content Security Policy (CSP) header not implemented	
Cookies	–	0	No cookies detected	
Cross-origin Resource Sharing	✓	0	Content is not visible via cross-origin resource sharing (CORS) files or headers	
HTTP Public Key Pinning	–	0	HTTP Public Key Pinning (HPKP) header not implemented (optional)	
HTTP Strict Transport Security	✗	-20	HTTP Strict Transport Security (HSTS) header not implemented	
Redirection	✓	0	Initial redirection is to HTTPS on same host, final destination is HTTPS	
Referrer Policy	–	0	Referrer-Policy header not implemented (optional)	
Subresource Integrity	✗	-50	Subresource Integrity (SRI) not implemented, and external scripts are loaded over HTTP or use protocol-relative URLs via src="//..."	
X-Content-Type-Options	✗	-5	X-Content-Type-Options header not implemented	
X-Frame-Options	✗	-20	X-Frame-Options (XFO) header not implemented	
X-XSS-Protection	✗	-10	X-XSS-Protection header not implemented	

Fig. 4.5: Consulta en Mozilla Observatory de una página web (Web: deusto.es).

En esta sección se ha tratado de realizar un estudio lo más completo posible haciendo uso únicamente del dato de la URL del sitio para que nos permita tener una visión más amplia haciendo uso, también, de otros analizadores webs los cuales nos aportan una información de gran valor para un investigación completa de un sitio web. Por otra parte, en este apartado hemos observado la gran cantidad de información que podemos obtener haciendo uso solo de la URL de la página, y cómo es un análisis indispensable a la hora de desarrollar una analizador lo más completo posible.

4.2.2.2. Analizador de ficheros

Otro tipo de información que recoge el crawler son los recursos que la página necesita, esto incluye los ficheros JavaScript, los ficheros HTML, los ficheros CSS o las imágenes, por lo tanto se explicará a continuación el análisis que se realiza sobre estos archivos. El análisis que se podría realizar sobre cada uno de estos ficheros seguramente podría dar para un proyecto entero, sobretodo los ficheros JavaScript, por lo tanto se intentará realizar un estudio sobre estos, lo más profundo posible pero a la vez acotado al tiempo límite que tiene el proyecto.

Este apartado daría comienzo recorriendo todos los archivos recogidos por el crawler y aplicando los distintos módulos según corresponda, dependiendo del tipo de fichero al que nos enfrentemos, en este caso casi todos los análisis estarán centrados en los ficheros JavaScript y en los scripts en línea dentro incrustados en el HTML.

El primer análisis que llevaríamos a cabo, en caso de que el fichero fuera un documento HTML

o un script en JS, sería el de buscar código JavaScript ofuscado con la curiosa técnica conocida como JSFuck. Este código se puede encontrar tanto en los ficheros JavaScript como en los script en línea de HTML. Para este módulo, y para los posteriores en los que se requiera sacar los scripts en línea incrustados en el HTML, se hace uso de la biblioteca de Python Beautiful Soup [82], que nos permite parsear de forma sencilla el lenguaje de marcado HTML. Seguidamente podemos ver un sencillo caso de este tipo de ofuscación, creado con la página web jsfuck [111].

```
// Un ejemplo de alert(1) ofuscado con JSFuck sin hacer uso de eval
   ↪ ()
(![]+[])[+!+[]]+(![]+[])[!+[]+!+[]]+(!![]+[])
   ↪ [!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]+(!![]+[])
   ↪ [+[]]+(![]+[][(![]+[])[+[]]+([![]]+[][[]])
   ↪ [+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])
   ↪ [!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])
   ↪ [!+[]+!+[]+[+[]]]+[+!+[]]+(!![]+[][(![]+[])
   ↪ [+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])
   ↪ [!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])
   ↪ [!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])[!+[]+!+[]+[+[]]]
```

Listing 4.4: "alert(1)" ofuscado con JSFuck.

Primeramente utilizaremos una expresión regular hecha para el proyecto, para ser capaces, de entre tantos caracteres e información,reconocer este tipo de técnica. Para crear esta expresión regular, hemos hecho mucho uso de la página web [112] que permitió poder hacerla y ajustarla de una forma sencilla. Por lo tanto, hemos desarrollado y usado la siguiente expresión regular para detectar este tipo de ofuscación:

```
# Variable en la que guardamos la regexp de JSFuck
JSFUCK_REGEX = re.compile('(\[*\]*\(*\)*\+*\!*)+', re.DOTALL)
```

Listing 4.5: Expresión regular para encontrar texto ofuscado con JSFuck.

En caso de encontrar algún script ofuscado con esta técnica, y en caso de encontrarlo daríamos paso a desofuscarlo. En en el proyecto hemos hecho uso de un proyecto, ya creado, de Github que se llama AntiJSFuck [113]. Este proyecto nos permite desofuscarlo de una forma sencilla, dando igual si hace uso de la función "eval()" para la ofuscación o no. Por lo tanto, en caso de encontrar algún script ofuscado con JSFuck el analizador será capaz de detectarlo y desofuscarlo.

El segundo módulo, también se utiliza en caso de que el fichero sea un HTML o un script, y en este caso es un estudio relacionado con la actividad maliciosa del cryptojacking. Es el encargado de buscar algún patrón conocido como puede ser, el del llamado orquestador, refiriendose a las sentencias en JavaScript encargadas de empezar el proceso del cryptojacking. Este apartado, se lleva a cabo en ficheros HTML y en JavaScript, debido a que aunque normalmente es usado en scripts JS en línea, puede darse el caso de que esté en un fichero propio. Este tipo de análisis está basado en el paper presentado en ACM Computer and Communications Security (2018) llamado

"MineSweeper: An In-depth Look into Drive-by Cryptocurrency Mining and Its Defense" [12] el cual, además de en esta parte, se ha utilizado para la creación de otras partes del IsKrawler Analyzer. En la tabla 4.1 se encuentran los distintos tipos de patrones que busca este módulo y los cuales indicarían una actividad maliciosa.

Servicio de minería	Patrón — Palabras clave
coinhive	new CoinHive.Anonymous — coinhive.com/lib/coinhive.min.js — authedmine.com/lib/
cryptonoter	minercry.pt/processor.js — .User(addr
nfwebminer	new NFMiner — nfwebminer.com/lib/
jsecoin	load.jsecoin.com/load
webmine	webmine.cz/miner
cryptoloot	CRLT.anonymous — webmine.pro/lib/crlt.js — verifier.live/lib/crypta.js
coinimp	www.coinimp.com/scripts — new CoinImp.Anonymous — new Client.Anonymous — freecontent.stream — freecontent.data — freecontent.date
deepminer	new deepMiner.Anonymous — deepMiner.js
monerise	apin.monerise.com — monerise_builder
coinhave	minescripts.info
cpufun	snipli.com/[A-Za-z]+"data-id=
minr	abc.pema.cl — metrika.ron.si — cdn.rove.cl — host.dns.ga — static.hk.rs — hallaert.online — st.kjli.fi — minr.pw — cnt.statistic.date — cdn.static-cnt.bid — ad.g-content.bid — cdn.jquery-uim.download
mineralt	ecart.html?bdata= — /amo.js» — mepirtedic.com

Tabla 4.1: Patrones de los distintos tipos de orquestadores de los servicios de cryptojacking. (Fuente: Publicaciones científicas [12] [15])

Otro módulo relativo a esta parte, será el de hacer uso de la biblioteca JS Beutifier [83], que nos permite reformatear los scripts JavaScript, re-indentarlos, desempacarlos en caso de estar empacados mediante el popular empacador de Dean Edward's, además de tratar de deofuscarlo en caso de ser una técnica de ofuscación conocida. Haciendo uso de esta biblioteca, podremos realizar todo lo anterior comentado sobre los ficheros script en línea de HTML y sobre los JavaScript, dejando en la carpeta de resultados el script resultante. Esta utilidad es muy interesante no solo para los scripts ofuscados, dado que todos los JavaScript que solemos encontrar en las páginas están minificados y tratados de reducir al máximo, incluso los nombres de variables, tratando así de darle más rendimiento a la página. A continuación podemos observar un código JS ofuscado, creado con la web [114], y el resultado que sacaría después de analizarlo, gracias a la biblioteca comentada.

```
// JS ofuscado
eval(function(p,a,c,k,e,d){e=function(c){return c};if(!''.replace
    ↪ (/^/,String)){while(c--){d[c]=k[c]||c}k=[function(e){return d
    ↪ [e]}];e=function(){return'\\w+'};c=1};while(c--){if(k[c]){p=p
    ↪ .replace(new RegExp('\\b'+e(c)+'\\b','g'),k[c])}}return p}('5
    ↪ 0(){4.3("2 1!")}0();',6,6,'hi|World|Hello|log|console|
    ↪ function'.split('|'),0,{}))

// JS desofuscado tras pasar por el analizador
function hi() {
  console.log("Hello World!");
}
hi();
```

Listing 4.6: Ejemplo sencillo de código JS ofuscado y su desofuscación.

Además se ha incluido en este analizador, una expresión regular la cual tratará de buscar en los scripts, tanto en línea como los ficheros JS, que se haga uso de la función "eval()". Esta función evalúa o ejecuta el argumento que le pasamos, es decir, si el argumento es una expresión, "eval()" evalúa la expresión. En caso, de que el argumento sea una o más declaraciones de JavaScript, "eval()" ejecuta las declaraciones. Esta función es muy conocida, además de poder a llegar a ser peligrosa, por ser usada en técnicas de ofuscación como hemos visto en el código anterior. La expresión regular que se utiliza y que es la encargada de buscar esta función es la siguiente:

```
# Variable en la que guardamos la regexp de eval()
JSEVAL_REGEX = re.compile('eval\(.+\)', re.M)
```

Listing 4.7: Expresión regular para encontrar si se hace uso de la función eval.

El último módulo desarrollado en este apartado, ha sido un buscador de links para los scripts, tanto en línea como ficheros propios. Esta es una técnica muy utilizada en el Bug Bounty, es decir, en la búsqueda de bugs de páginas webs, debido a que en ocasiones podemos encontrarnos con lugares, los cuales no deberían de estar abiertos a la red y que ofrecen todo tipo de información privada o que llevan consigo algún otro problema mayor.

Es por ello, que es una técnica de análisis interesante que se ha añadido al proyecto. Para llevar a cabo este estudio, y muy parecido a otros, se hace uso de una expresión regular, sacada del proyecto [115], que se podrá ver a continuación. Se entiende que el link se encontrará incluido entre comillas simples o dobles debido a que está en código JS y debe ser del tipo "String". Dentro de estas comillas se buscará distintos tipos de casos. A continuación se puede ver la estructura principal de la expresión regular:

```
# Busca el inicio del delimitador unas comillas dobles o simples
("|')
  (
```

```
        # CASO 1
    |
        # CASO 2
    |
        # CASO 3
    |
        # CASO 4
    |
        # CASO 5
  )
#Delimitador de las comillas simples o dobles
("|')
```

Listing 4.8: Expresión regular para encontrar links en código JavaScript.

Conociendo ya la estructura principal, la cual busca que el texto comience y termine con comillas simples o dobles. Dentro de toda la cantidad de textos que podríamos encontrar con esta simple premisa, debemos acotar mucho más nuestra búsqueda y para ello hay distintos casos que se evaluaran y en caso de que algún texto encaje con alguno de ellos se reportará y guardará ese texto.

```
# CASO 1
    # Ejemplo 1: "http://example.com/pathone/pathtwo"
    # Ejemplo 2: "ftp://example.com/pathone/pathtwo"

# Busca el esquema [a-Z]*1-10// <-> //
(([a-zA-Z]{1,10}://|//)
# Busca el nombre de dominio -> cualquiera (menos "'/) + un punto
[^"'/]{1,}\.
# La extension del dominio (min 2) y en caso de que haya, la ruta
[a-zA-Z]{2,}[^"']{0,})
```

Listing 4.9: Expresión regular para encontrar links en código JavaScript.

```
# CASO 2
        # Ejemplo 1: "../pathone/pathtwo"
        # Ejemplo 2: "../pathone/pathtwo"

# Empieza con / <-> ../ <-> ./
((/|\.\./|\./)
# El siguiente caracter no puede ser ninguno de los siguientes
[^"'><,;| *()(%%$^/\\\[\]]
# El resto de caracteres que no puede ser
[^"'><,;|()]{1,})
```

Listing 4.10: Expresión regular para encontrar links en código JavaScript.

```
# CASO 3
    # Ejemplo: "api/private/info.action?id="

# Posible endpoint con /
([a-zA-Z0-9_\-/]{1,}/
# Nombre del recurso
[a-zA-Z0-9_\-/]{1,}
# Rest + extension (dimension 1-4 O action)
\.([a-zA-Z]{1,4}|action)
# ? O # con parametros
([\?|#][^"|']{0,}|))
```

Listing 4.11: Expresión regular para encontrar links en código JavaScript.

```
# CASO 4
    # Ejemplo: "api/dev?id="

# REST API (sin extension) con /
([a-zA-Z0-9_\-/]{1,}/
# Normalmente los endpoints REST tienen 3+ caracteres
[a-zA-Z0-9_\-/]{3,}
# ? O # con parametros
([\?|#][^"|']{0,}|))
```

Listing 4.12: Expresión regular para encontrar links en código JavaScript.

```
# CASO 5
    "api_internal.php?id=5"

#  Fichero
([a-zA-Z0-9_\-]{1,}
# punto + extension (Ej: .php)
\.(php|asp|aspx|jsp|json|
     action|html|js|txt|xml)
# ? O # con parametros
([\?|#][^"|']{0,}|))
```

Listing 4.13: Expresión regular para encontrar links en código JavaScript.

Con todos estos módulos, se trata de hacer un análisis lo más completo posible de los ficheros que el crawler es capaz de recolectar de una página web. Esta sección se ha centrado sobre todo en los scripts JavaScript, dado que es código que se ejecuta en el navegador, por lo tanto, a la hora de estudiar la seguridad de una web, es de las partes más importantes. Al igual que el resto de apartados del proyecto, son muy fácilmente extensibles para ir añadiendo otros tipos de módulos con vistas a hacer más completo el IsKrawler Analyzer.//

4.2.2.3. Analizador de cookies

En este apartado se explicará el analizador de cookies que se ha implementado para el proyecto. Para llevarlo a cabo, se ha hecho uso de la información del repositorio de Github [116] en el que nos encontramos con una gran información que contiene sobre distintos tipos de cookies. Las cookies se crearon entorno a 1994 debido a que el protocolo HTTP es un tipo de protocolo considerado "stateless", es decir, que no guarda estado, por lo tanto se necesitaba alguna forma de identificar a los usuarios o clientes, como por ejemplo, para el caso de los comercios electrónicos.

Desde entonces, el uso de las cookies ha crecido mucho y actualmente su uso no se limita al de guardar las sesiones de los usuarios con sus carritos de compra, si no que se están utilizando para otro tipo de fines como pueden ser el tracking de los usuarios a través de la red, el de fines analíticos como por ejemplo para el análisis estadístico del tráfico de un sitio web o el de marketing de anuncios, en que poder adaptar los anuncios en base a criterios geográficos y de otra índole.

Es por ello, que en este proyecto no se quería dejar fuera este tema, aunque muchas de las soluciones que se podían encontrar en internet, eran de pago, como puede ser el caso de la web cookiedatabase [117].

El análisis primeramente realiza una búsqueda en base a los resultados del crawler recogiendo cada una de las cookies encontradas con los siguientes datos: el nombre de la cookie, su valor, el dominio, la ruta, cuando expira esta, el tamaño, si está activo los parámetros "httpOnly", "secure" o "session", el atributo SameSite y el de la prioridad. Posteriormente, se sanitizan los datos como el de la expiración en la que el dato en segundos se pasa a una fecha concreta y más visible.

Finalmente, se hace uso del proyecto comentado para completar la información ya obtenida por el crawler con datos como la plataforma a quien pertenece este tipo de cookie, la categoría de esta (marketing, analytics...) o la descripción de su utilidad. Al igual que el resto del proyecto, está pensado para poder fácilmente añadir más tipos de análisis de una forma sencilla. En la tabla 4.2 se encuentra un ejemplo sobre todos los datos recopilados respecto a una cookie.

4.2.2.4. Analizador de los perfiles del procesador en el navegador

Respecto a este apartado, explicitar que este análisis se puede llevar a cabo debido a las adiciones realizadas en el crawler. El estudio que se llevará a cabo en este apartado está basado en el paper [13] y en su proyecto [66], presentados en la conferencia de ACM CCS (Computer and Communications Security) en 2018 [118]. La funcionalidad se dividirá en dos apartados diferenciados unas enfocado a las operaciones de hash en el procesador y otro entorno a la actividad de la pila del navegador. Este análisis que se realiza es muy dependiente de la cantidad de segundos que se esté en la página, al periodo de iniciación que pueden tener algunos procesos, etc.

El primer módulo, primeramente, buscará en los nodos de las funciones que hemos obtenido con el crawler y en los nombres, para comprobar si alguno de ellos corresponde con alguna de las librerías de hash más utilizadas como pueden ser "cryptonight", "hash", "sha256", o "crypto". Otro análisis, explicado muy extensamente en el artículo, es el del tiempo que pasan las páginas web realizando operaciones de hash, para ello primero, se guarda un recuento total de usos de las

Tipo	Valor
Nombre	_gcl_au
Valor	1.1.397761823.1589900139
Expira en	2020-08-17 16:55:38
Plataforma	Google
Tipos de dominios	Advertiser's website domain (1st party)
Categoría	Marketing
Descripción	Used by Google AdSense for experimenting with advertisement efficiency across websites using their services.
Dominio	.deusto.es
Ruta	/
Tamaño	31
Parámetro httpOnly	False
Parámetro secure	False
Parámetro session	False
Parámetro priority	Medium

Tabla 4.2: Ejemplo de información obtenida de una cookie en la web www.deusto.es.

distintas funciones, quitando por supuesto la de "(idle)", la cual corresponde a que el navegador está ocioso.
Por otra parte, para poder diferenciar los distintos tipos de operaciones se hace un recorrido de estas, se compara que su nombre de función no corresponda a ninguno de los siguientes:"(idle)", "(program)", "(root)", "(garbage collector)" y por otro lado se comprueba que el recuento de usos de esa función, al ser de hash en actividades de cryptojacking, sea muy alto, por ejemplo, superior 500 usos. Dentro de este bucle, en la diferenciación de los distintos tipos de operaciones, se iría recogiendo dentro de todas las diferenciadas, únicamente la que más usos ha tenido como función.

Después de seleccionar la función que más usos ha tenido, se comprobará que la misma es periódica, es decir, que en la información que nos recoge el crawler sobre la pila, esta función aparece repetidamente. Finalmente, se comprobará aproximadamente si el tiempo utilizado para esta operación, y entendiendo que es una operación de hash al tener muchos usos y ser repetida en el tiempo, es mayor de lo normal y en caso de ser positivo se reportará como actividad maliciosa.

En la segunda parte del análisis, es decir, en el segundo módulo se realizará un análisis sobre la pila del navegador, en la que si se repite una cadena de llamadas periódicamente (en un tiempo fijo), y además, la cadena de llamadas ocupa más del 30 % del tiempo de ejecución se entenderá que esta realizando actividades maliciosas de cryptojacking. En caso de querer saber más sobre estos dos módulos, me remito al paper arriba nombrado y citado, en el cual se pueden encontrar el "pdf", el proyecto en Github y el vídeo de la presentación en la conferencia de ACM.

Estos dos módulos nos permiten tener una forma más de detectar actividades maliciosas y en un futuro, muy sencillamente, se podría hacer uso de esta información extraída para otros tipos de análisis. Finalmente, se crea un diagrama, haciendo uso de la librería graphviz [85] de Python, con las distintas funciones de los perfiles del procesador en el navegador, en el que se incluye el nombre, las veces que ha sido usado y un identificador. En la figura 4.6 podemos observar un ejemplo de este diagrama el cual nos facilita el análisis.

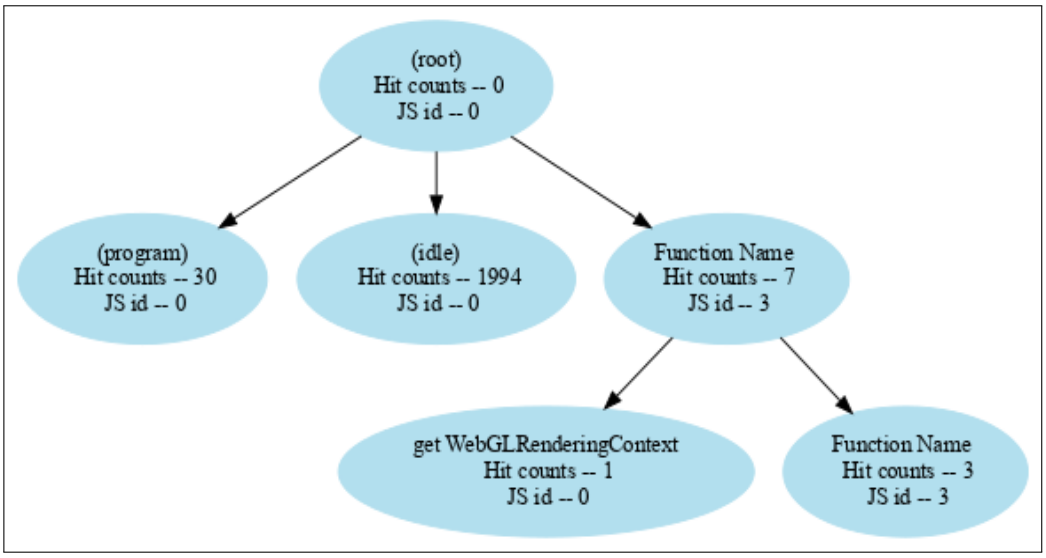


Fig. 4.6: Funciones procesadas en el navegador.

4.2.2.5. Analizador de comunicaciones mediante WebSockets

En este apartado se explicará que tipo de estudio se realiza para esta tecnología novedosa de comunicaciones. Primeramente recordar que este tipo de análisis se puede realizar gracias a las adiciones y modificaciones realizadas en el crawler, como se ha explicado anteriormente, las cuales nos permiten recoger todo tipo de información respecto a estas comunicaciones. Este apartado se basará en la búsqueda de conexiones utilizadas en el cryptojacking, basadas en artículos científicos presentadas en congresos como son el de "MineSweeper: An In-depth Look into Drive-byCryptocurrency Mining and Its Defense" [12] o el de "End-to-End Analysis of In-Browser Cryptojacking" [119], pero en un futuro se podrá extender muy sencillamente para reconocer otro tipo de actividades maliciosas.

Primeramente se comprobará si en el resultado del crawler ha habido alguna conexión de este tipo. En caso de ser afirmativo, daría comienzo el análisis y el primer punto sería el de hacer uso de la API de VirusTotal [102] para contrastar la información que tienen respecto a la url con la que se conecta en la comunicación, dándonos información de si algún antivirus la reconoce como maliciosa.

Hay casos en los que aún habiendo creado la conexión mediante WebSocket, puede ser que no se hayan enviado ningún tipo de frame por eso debemos de hacer una previa comprobación. Dentro de todos los frames que podemos encontrar en la comunicación haremos una observación diferente para los recibidos por nosotros como los enviados por nosotros. En los dos análisis será una técnica similar debido a que en el cryptojacking se utiliza el protocolo Stratum, el cual va normalmente en texto plano, para no reducir el rendimiento, y del que se conocen las formas mayoritarias en las que se comunican.

Respecto a los frames recibidos, primeramente buscaremos si alguna palabra clave de las conexiones conocidas se encuentra en los frames, como pueden ser "type", "command" o "identifier". En caso de haber encontrado alguna de las palabras claves iremos recorriendo los frames comparando si es de autenticación aceptada, de haber recibido un trabajo o de haber aceptado una solución que hayamos enviado. En la tabla 4.3 se encuentran las palabras claves que se buscan en el análisis de los frames recibidos.

Tipo	Palabras clave
Autenticación aceptada	authed — work
Trabajo recibido	job — work — get_job — set_job
Hash/Trabajo aceptado	hash_accepted — accepted

Tabla 4.3: Protocolo Stratum y sus palabras claves en los frames recibidos.

Respecto a los frames enviados, primeramente buscaremos al igual que en los recibidos por las palabras claves, como son "type", "command" o "identifier". En caso de haber encontrado alguna de las palabras claves iremos recorriendo los frames comparando si es para autenticarnos o de enviar la solución de algún trabajo. En la tabla 4.4 se encuentran las palabras claves que se buscan en el análisis de los frames enviados.

Tipo	Palabras clave
Autenticación	auth — connect — handshake — info
Enviar solución de un trabajo	submit — share

Tabla 4.4: Protocolo Stratum y sus palabras claves en los frames enviados.

En el caso de que se hayan ido reconociendo los frames con la técnica nombrada, además de ir recogiendo que tipos de acciones se han estado realizando, en algunos casos también se puede recoger otro tipo de información. Por ejemplo, en el caso de hacer una petición de autenticación se recogerá también el tipo de moneda a minar y la clave del sitio, el cual es un identificador del cliente y es muy útil para detectar el tipo de campaña al que pertenece.

Finalmente, y tras realizar las búsquedas y el análisis de todas las palabras claves tanto en los frames enviados como en los frames recibidos en las comunicaciones encontradas, se creará un gráfico para poder visualizar de forma sencilla las comunicaciones habidas, haciendo uso de la librería graphviz [85] de python. En la figura 4.7 tenemos uno de los gráficos resultantes de uno de los análisis realizados en el proyecto.

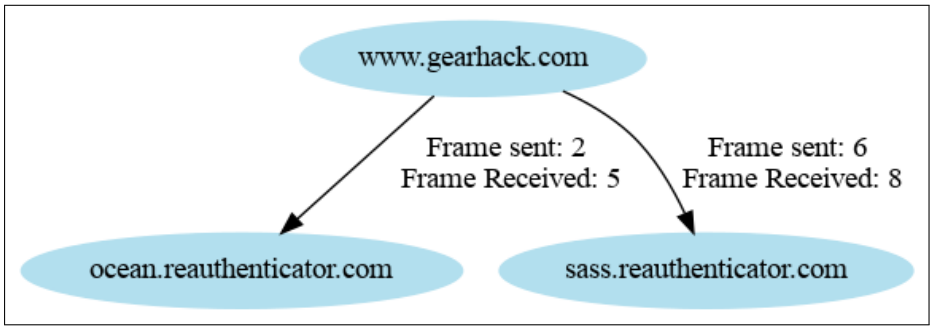


Fig. 4.7: Visualización de las conexiones mediante WebSocket.

4.2.2.6. Analizador de ficheros creados en la tecnología WebAssembly

Respecto al Analizador de WebAssembly (Wasm) nos encontramos con dos módulos bien diferenciados en el proyecto. Estos módulos funcionarán en los casos en los que la página web en cuestión haga uso de al menos un fichero de WebAssembly. En el analizador se ha implementado, debido a la falta de librerías, una forma de reconocer este tipo de ficheros dentro de todos los recogidos por el crawler, para ello el analizador busca los denominados "Magic Number", los cuales son unos caracteres alfanuméricos al principio de los archivos que nos permiten reconocer que tipo de fichero es, y que en caso de ser binarios WebAssembly tendrían el valor "00 61 73 6d". En el caso de que se haya recogido algún WebAssembly (wasm) y antes de dar comienzo a los análisis con los módulos, se saca el hash de este archivo y es comparado mediante una API con VirusTotal [102], permitiéndonos saber si algún antivirus lo reconoce como maligno. Además de los dos módulos se ha integrado una solución de blacklisting, es decir, se ha recopilado una lista de hashes de ficheros wasm malignos, con los cuales poder comparar los encontrados en la determinada página web a

analizar.

Por otra parte, y antes de entrar a la explicación de los módulos, explicaré el caso curioso que ocurre con los WebAssembly a la hora de reconocerlos con el crawler. Los ficheros WebAssembly (Wasm), se recogen gracias a unas modificaciones que se realizaron en el crawler para este proyecto, siempre se nombran como "Fetch" junto a unos números. Este nombre, como se ha comentado en las modificaciones realizadas en el crawler, se debe al tipo de recurso percibido por el motor de renderizado, en este caso al ser Chrome, se llama Blink. Los tipos de recursos percibidos y renderizados por el motor se explicitan en la documentación del protocolo de depuración de Chrome [93].

En el primer módulo haremos uso del proyecto Octopus [14]. Haciendo uso de este proyecto seremos capaces de desensamblar los binarios dejando el fichero resultante en la carpeta de resultados, es decir, seremos capaces de convertir el fichero binario, que normalmente tiene una extensión de "wasm", en su interpretación en formato texto o ensamblador con extensión "wat". Este proyecto también nos permite crear el control flow graph (CFG) del fichero que hemos desemsamblado anteriormente. Este control flow graph (CFG) del programa nos permite representar o visualizar todos los posibles caminos que puede tener en ejecución, por esta razón es una técnica muy utilizada en el área de seguridad del análisis de programas. Además, esta funcionalidad, al igual que en las posteriores en las cuales se hace uso de este proyecto, nos permite exportarla en diversos tipos de fichero como pueden ser "jpeg" o "pdf". En la figura 4.8 podemos visualizar un ejemplo del resultado de esta funcionalidad.

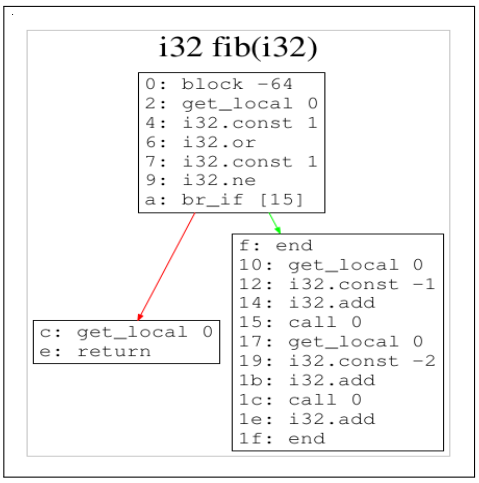


Fig. 4.8: Wasm CFG by Octopus Project (Fuente: Octopus Github [14]).

Otra funcionalidad que este proyecto nos ofrece de manera sencilla, es la construcción del call graph. Este call graph nos permite representar y visualizar las llamadas entre subrutinas o funciones de un programa. Como en el anterior caso, podemos exportar el resultado en diversos tipos de fichero como pueden ser "jpeg" o "pdf". Como explica el autor en sus charlas [120], los call graph de los ficheros nos permite detectar visualmente si se realiza algún tipo de cryptojacking utilizando el wasm debido a que en la representación nos encontramos con una gran cantidad de llamadas estáticas a funciones locales y por otra parte, porque suele haber muchas referencias cruzadas (XREFs) a una función determinada. En la imagen 4.9 podemos ver un ejemplo del resultado de esta funcionalidad.

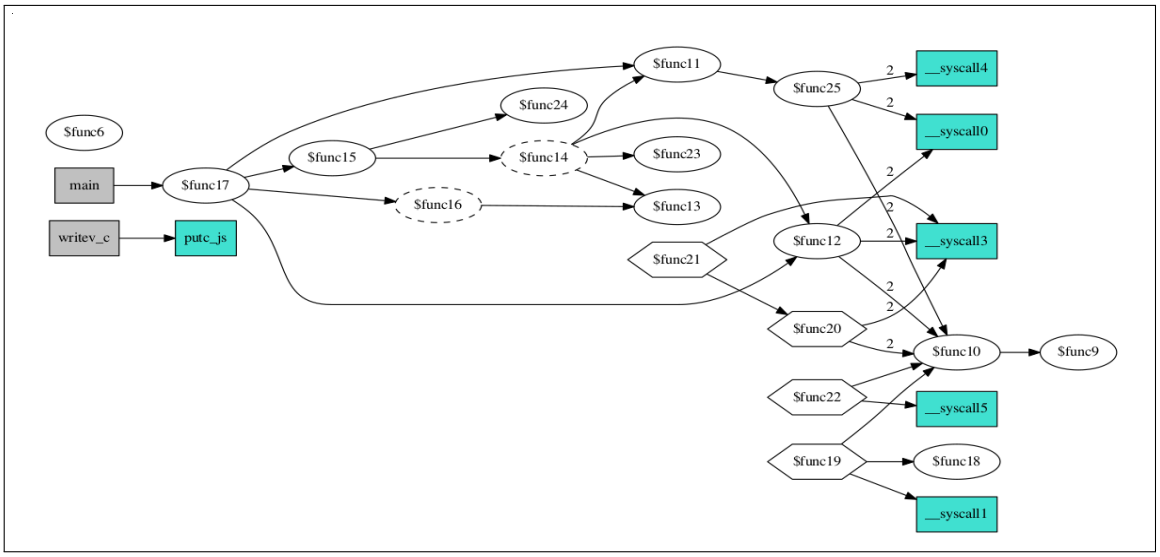


Fig. 4.9: Wasm Call Flow by Octopus Project (Fuente: Octopus Github [14]).

Por último, este proyecto nos ofrece otro funcionalidad muy interesante como es el análisis de las instrucciones de las funciones. En esta funcionalidad nos analiza las funciones representando y pudiendo visualizar por cada función que tipo de instrucciones tiene, sean de Control (Branches, blocks...), lógicas, aritméticas o de memoria. Al igual que el call graph, esta funcionalidad también nos permite detectar visualmente la actividad maliciosa llamada cryptojacking. Alguno de los puntos a tener en cuenta, es que suele haber funciones las cuales no tienen casi operadores de control y por otra parte, que estas funciones hacen uso de una gran cantidad de variables. El autor estima que estas funciones suelen tener alrededor de un 30 % más de operadores de constantes y alrededor de un 30 % más de operaciones aritméticas/bitwise. Se muestra en la figura 4.10 un ejemplo de esta funcionalidad.

Respecto al segundo módulo para analizar wasm, al igual que en el primero, lo primera será desemsamblar el fichero “wasm” a “wat”. En este caso para desemsamblarlo haremos uso del proyecto oficial WebAssembly Binary Toolkit (WABT) [67], el cual desemsablará el fichero y dejará el fichero resultado en las carpeta de resultados. En este caso haremos uso de esta utilidad lanzando un subproceso utilizando la librería de Python subprocess [121] dado que son binarios compilados de C++ los cuales no tienen ningún tipo de soporte para Python. Por otra parte, se hace uso de este proyecto debido a que es el oficial del proyecto encargado de esta tecnología y por otro parte, porque nos crea el desemsamblado con alguna diferencia respecto al utilizado en el anterior módulo. Este segundo modulo se utilizará en caso de indicar a la hora de lanzar el análisis la opción correspondiente, debido a que se necesita tener el binario o ejecutable WABT para poder funcionar.

Este segundo módulo ha sido creado e implementado para el proyecto haciendo uso del proyecto Minesweeper del grupo VUSec tanto del paper [12] presentado en ACM como del proyecto [16] el cual es bastante complejo, está desarrollado en Python pero en la versión 2 por lo que habrá que adaptarla a la versión 3 del proyecto y del cual solo nos interesan ciertas partes. La implantación de este segundo modulo es importante dado que desarrollamos una versión propia para al analizador

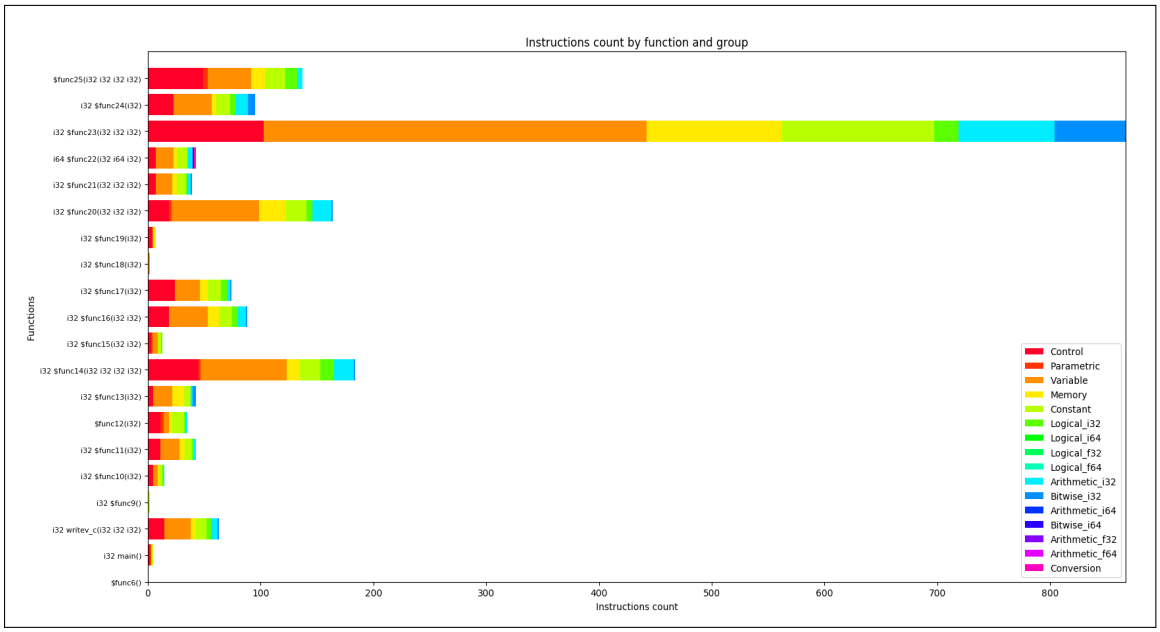


Fig. 4.10: Wasm Function Analytics by Octopus Project (Fuente: Octopus Github [14]).

el cual no depende de otros proyectos y nos permite sencillamente en un futuro ir mejorándolo e ir añadiéndole mas funcionalidades.

Lo primero que realiza este módulo es mediante expresiones regulares recoger las funciones que este fichero contiene y por cada función que tiene este fichero recoger las operaciones que tiene para su uso posterior. Lo siguiente que realiza, y gracias al paso anterior, va recorriendo el código de cada una de las funciones que tiene el fichero y haciendo uso de expresiones regulares, encuentra y recoge por cada función llamadas a otras funciones. Este segundo paso, nos permite crear nuestro Call Graph para el fichero que estamos analizando. El tercer paso que realiza este módulo es recorrer nuevamente las funciones que contiene el fichero "wasm" e ir contando el total por cada tipo de operación. A continuación, el módulo recorre las funciones recogiendo los bucles o loops que contiene y recogiendo también su contenido, pudiendo a su vez, contener otro bucle dentro de este. Además por cada bucle cuenta el total por cada tipo de operación que contiene. Este paso es un poco complicado debido a que se necesita reconocer donde comienza y finaliza el bucle, y por ejemplo, los bucles al igual que las condiciones finalizan utilizando la palabra clave "end" por lo tanto debemos tener cuidado al encontrar esa finalización y saber a qué tipo se está refiriendo.

A continuación se describirán las dos partes más complejas del módulo y las cuales se han desarrollado modificando el proyecto Minesweeper [16] comentado anteriormente. La primera parte será un acercamiento a detectar el algoritmo CryptoNight muy utilizado en el cryptojacking y el segundo será un detector de un algoritmo genérico de criptografía en cual de detectará operaciones de criptografía.

En el estudio comentado, se dieron cuenta que la totalidad de páginas web que realizaban cryptojacking haciendo uso de la tecnología WebAssembly utilizaban el algoritmo CryptoNight por lo tanto crearon una caracterización de este algoritmo, en el que se incluyen otro tipo de algoritmos

como son Keccak, Blake, Skein, Groestl o AES. Entonces, nosotros haremos uso de este fichero de caracterización que denominaron cn_fingerprint para poder detectar de la mejor manera este tipo de actividad maliciosa. Este fichero, básicamente tiene los distintos algoritmos que he comentado y por cada uno de ellos, tenemos las distintas operaciones en WebAssembly que utilizan, y su correspondiente cantidad. Teniendo esta caracterización y habiendo ya recopilado las funciones del programa, debemos para la detección reconocer para cada uno de los algoritmos de CryptoNight definir cual es la función del programa que más se asemeja a este. Para hacer esta búsqueda debemos recorrer por cada algoritmo las distintas funciones del programa comparando los siguientes parámetros; primero la cantidad del total de operaciones del algoritmo que se han encontrado en esa función; segundo con otra variable debemos de valorar de esas operaciones la diferencia que hay entre las cantidades de la caracterización y de la función; por último, debemos de ir recogiendo cual es la función que más encaja por cada algoritmo de la caracterización, por lo tanto, cada vez que se evalúa una función, con los parámetros comentados anteriormente, se debe valorar si encaja mejor que el resto, entonces se comparará si tiene más operaciones encontradas o una menor diferencia de cantidades con la "mejor" función hasta el momento.

Finalmente, se valorará de toda esta información recopilada si realmente realiza todos los algoritmos y en que medida encajan con la caracterización. Para ello, primero se dividen las funciones que mejor han encajado en tres grupos; el primero es el que encaja exactamente con la caracterización, es decir, que la diferencia de cantidades entre el de la función y la caracterización es cero y que además contiene todas las operaciones; el segundo caso es el de funciones que encajan bien, es decir, se reduce la necesidad de que estén todas las operaciones y la necesidad de que la diferencia sea tan exacta; por último, están las operaciones que realmente no cumplen con eso requisitos y por lo tanto se toma que no son utilizadas para el algoritmo CryptoNight. Después de haber realizado esta diferenciación, se chequea que tiene la cantidad mínima de los algoritmos necesarios para realizar CryptoNight y nos proporcionará un booleano indicándonos el resultado de la detección, siendo esta positiva o negativa.

En esta segunda parte se trata de detectar un algoritmo de criptografía genérico. Para ello se recorrerán las funciones y dentro de estas, se buscarán en los bucles, dado que es como se realizan los algoritmos criptográficos, operaciones en WebAssembly de criptografía, como por ejemplo "i32.xor", "i32.rotl" o "i32.rotr". Recogiendo las cantidades de operaciones de hash dentro de una función y la cantidad de operaciones de hash por bucle se evalúa que si estas superan una cantidad podría se sospechosa de realizar operaciones de cryptojacking.

En conclusión, gracias a estos dos módulos, en el cual, el primero hace un análisis exportando imágenes con los resultados, pudiéndolos visualizarlo de una forma sencilla, y en cual, el segundo detecta, haciendo uso de distintos estudios, si realiza operaciones de hash dando como resultado un positivo o un negativo. Estos dos apartados, nos permiten realizar conjuntamente un análisis amplio e íntegro de los programas desarrollados en la tecnología WebAssembly. Por otra parte, el tener un módulo el cual únicamente depende de los binarios oficiales para el desensamblado, es una gran ventaja a la hora de poder realizar en un futuro adaptaciones propias para otros tipos de proyectos, como por ejemplo, poder detectar otro tipo de actividades maliciosas.

4.2.2.7. Analizador de otros tipos de información

En esta sección, se recopilarán los análisis que por tiempo no se han podido completar lo suficiente como para abarcar una sección, o bien porque la información que obtiene el crawler respecto a eso no es demasiado extensa. Todos estos análisis, y muchos más, respecto a información que reporta el crawler pero que no se ha podido tratar en este proyecto, se podrán extender fácilmente en un futuro para completarlo y poder realizar un análisis cada vez más exhaustivo e íntegro de una página web.

El primer módulo de esta sección, es el de analizar los services workers, también denominados web workers, de la página web. La información que recopila el crawler respecto a estos trabajadores, encargados de operaciones en segundo plano de la página, es el ID del trabajador, el título, la url y si se ha clicado en algún lugar de la página para que se cree. Sabiendo esto, podemos darnos cuenta que no es una gran información y por lo tanto es difícil de saber que poder analizar. Aún así, en algún estudio sobre cryptojacking se nombra que normalmente las páginas benignas utilizan menos de 3 service workers [122] [16], por lo tanto, se recogerá toda la información y también se contará el número total. Otro módulo incluido en este apartado, es el de análisis de frames de una página web. Para ello, se recorre la información aportada por el crawler y recogemos la relacionada con los frames construyendo un árbol de jerarquías entre ellos. Posteriormente, se hace uso de la biblioteca de Python graphviz [85] para poder crear el árbol de forma visual mostrado en la figura 4.11.

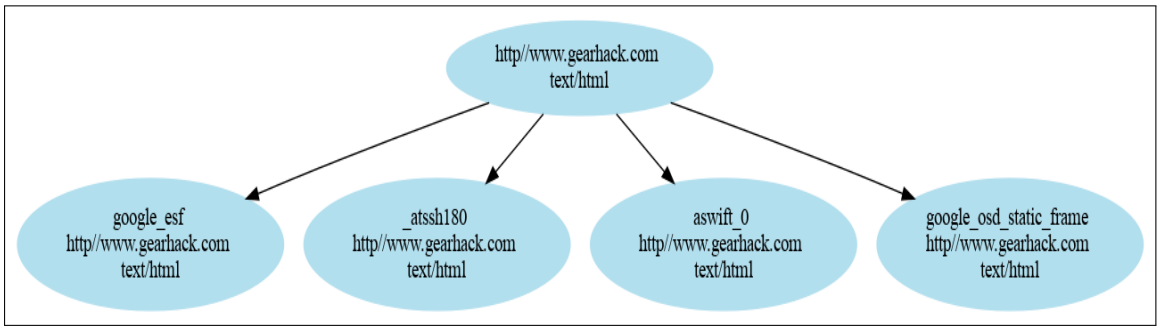


Fig. 4.11: Frames de una página web (Web: gearhack.com).

4.2.2.8. Análisis general

En este apartado, se explica como haciendo uso de todos los análisis anteriores, que realiza el IsKrawler Analyzer, se saca una valoración general. Para ello, hemos optado por usar la técnica de un sistema de evaluación o "scoring" en el que se toman los indicadores, resultado en los distintos módulos, para determinar una conclusión. Los indicadores que se hacen uso, en muchos casos no representan información relevante individualmente, por lo tanto, es el conjunto de varios de ellos lo que nos haría sospechar de una web. Por lo tanto, cada uno de los indicadores nos hará sumar o restar puntos de peligrosidad de la web.

El primer indicador hace uso de los datos WHOIS [80] extraídos en la sección 4.2.2.1. Se comprueba si la web se ha creado en un tiempo menor a un año y en caso de que sea así se sumaría 5 puntos al scoring. Otra medida es que si a la fecha de creación le restamos la de expiración y nos da un

tiempo menor a 1 año, le sumaríamos 10 puntos al resultado total. Siguiendo con indicadores de la URL, en caso de no encontrar información del dominio en SimilarWeb [100] o en Alexa Static [99], se sumaría un punto no encontrarse en alguna de las dos. Respecto a la información que nos ofrece Webutation [104], en caso de devolvernos un resultado menor a cincuenta, se sumarán 15 puntos. En caso, de que la web no haga uso de TLS sumaremos 5 puntos al resultado. Por último, dependiendo del resultado de Mozilla observatory header [109], se sumará un punto en caso de ser menor de 70, 5 puntos en caso de ser menor de cincuenta y diez puntos en caso de ser menor de veinticinco.

En caso de que se haya encontrado un director o "orchestrator" de cryptojacking se sumarán 25 puntos al scoring del sitio web. Por otra parte, en caso de haber encontrado código JavaScript ofuscado con la técnica JSFuck [111] se sumarán 15 puntos. Si en el análisis del perfilado del procesador en el navegador 4.2.2.4 se ha encontrado alguna función de tipo hash se sumarán 5 puntos, en caso de que el tiempo de hash sea mayor a lo normal se sumarán 25 puntos y si el tiempo de pila es mayor a lo normal se incrementará en 10 el resultado de la evaluación. Si el analisis de WebSocket4.2.2.5 ha encontrado comunicaciones de cryptojacking por el protocolo Stratum se sumarán 20 puntos. Respecto a la api de VirusTotal [102], en caso de ser positivo en alguna de sus consultas, ya sea en la URL, en las conexiones mediante WebSocket o de alguno de los ficheros Wasm, se incrementarán 30 puntos al resultado. Por último en caso de dar positivo en el algoritmo de CryptoNight o alguno genérico con el módulo de analizar ficheros wasm 4.2.2.6, se sumará 25 y 20 puntos, respectivamente.

En este sistema de evaluación, una web finalmente tendrá un resultado numérico que nos dirá si la web es maliciosa, si puede ser una web sospechosa o si realmente es una web benigna. Los rangos para estas tres categorías son, en caso, de ser superior a 50 puntos es maliciosa con total certeza, en caso de ser superior a 35 y menor de 50 sería sospechoso, y finalmente si es menor de 35 diríamos que es una web sin ningún tipo de actividad negativa para el visitante.

4.2.2.9. Resultados del analizador

En esta sección, se expone de que manera el IsKrawler Analyzer es capaz de exportar los resultados de todos los análisis. Como se ha ido explicando durante la memoria, el analizador tiene 4 formas de dar los resultados, haciendo uso de la pantalla, guardando un JSON con toda la información, guardarla en una base de datos (BD) o exportar un HTML que se puede observar con el navegador. Aun así. en las cuatro opciones, cuando ejecutamos el programa nos visualizará por pantalla el nombre del programa y otras variables. Por defecto, y la opción que siempre ocurre es la de exportarlo a un JSON a la carpeta de resultados.

Previamente a dar comienzo a las distintas opciones de exportación, se explicará en que lugar deja el analizador todos los resultados que va emitiendo. El analizador crea, si no está creada, una carpeta results dentro del proyecto y va guardando dentro todos los resultados de los distintos análisis. El esquema que sigue el analizador para guardar los resultados es el siguiente `results/URL/0..9999/` y normalmente dentro de esa carpeta se crearan las carpetas para los análisis de WebAssembly (carpeta /wasm) y de JavaScript (carpeta /js). Este esquema pretende ser sencillo, intuitivo y ordenado para los resultados. En la figura 4.12 se puede observar un ejemplo de la ejecución.

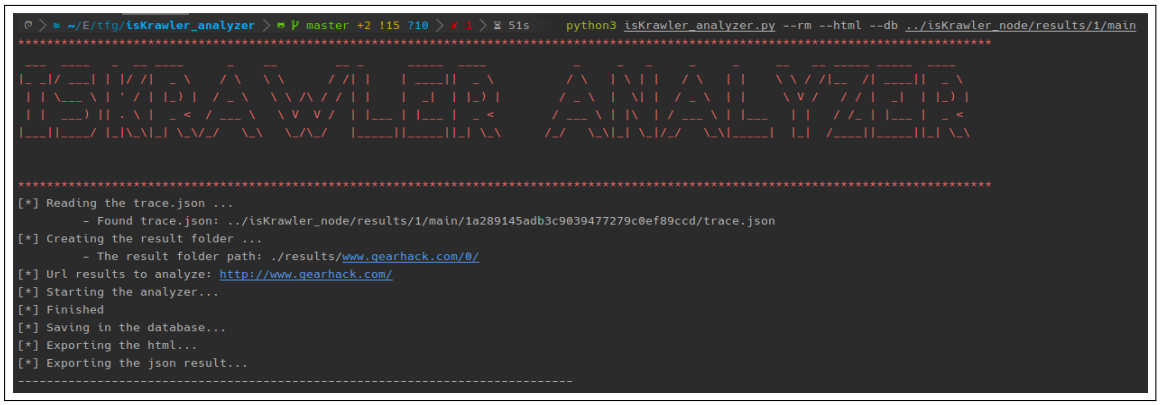


Fig. 4.12: Ejemplo de ejecución.

La primera opción que se va a describir es la de sacar el resultado por pantalla. Esta opción, al igual que la de HTML, es una de las opciones más difíciles de mantener, debido a que cualquier cambio en alguno de los módulos, ya sea de adición, de modificación o de eliminación, nos haría tener que cambiar el resultado que vería el usuario. Esta opción nos permite rápidamente ver los resultados, sin tener que acceder a otros sitios, justo tras haber lanzado el programa. Se puede utilizar cuando buscas determinada información de forma rápida. Haciendo uso del símbolo "¿" podríamos exportarlo a un fichero que quisiéramos. Está planteado que se imprima por pantalla por niveles, permitiéndonos fácilmente buscar la información que queremos. No es una solución en caso de queramos añadirlo a un reporte o en caso de que queramos realizarlo sobre más de una página. Por otra parte, no se incluyen en este método los diagramas creados con el analizador.

La segunda opción es la más sencilla de implementar y es la de exportar un JSON. En este caso, debido a las funcionalidades que nos aporta Python, el poder transformar toda la información de clases, listas y diccionarios a un json es muy simple. Esta es la opción por defecto y siempre se realiza. Esta opción es la más utilizada en caso de querer buscar un texto o análisis en concreto, como puede ser haciendo uso de herramientas como jq [123] y luego de grep [124], o en caso de querer hacer uso de esta información para pasársela a otro programa. Es una opción sencilla de mantener y realmente útil como forma de exportar los resultados del programa. Este fichero se guardaría en la carpeta de resultado que crea el analizador.

Otra opción es la de exportar el resultado en un HTML más agradable visualmente y que incluye los diagramas. Para ello, se hace uso de la biblioteca de Python llamada Jinja2, la cual nos permite generar el HTML. Esta biblioteca nos permite generar un HTML dinámico, es decir, que dependiendo de las variables que le insertemos nos cambiará el resultado. También nos permite añadirle listas y recorrerlas en la plantilla de HTML. Esta opción nos permite visualizar los resultados de todos los análisis de forma simple para todo tipo de persona. Al igual que la opción de ver los resultados por pantalla, es una opción difícil de mantener y que con cualquier cambio en el programa debemos de realizar lo mismo en la plantilla. Es una elección muy utilizada para poder añadir los resultados a un documento de reporte en caso de que alguien lo necesite. En las figuras 4.13 y 4.14 se puede observar un pequeño ejemplo del resultado que produce esta opción, obviamente solo se incluyen estas dos figuras tratando de enseñar cómo funciona esta opción sin ocupar la memoria de figuras. Además se ha añadido una función especial a esta forma de emitir los resultados, que se

utiliza si se indica el parámetro a la hora de ejecutar el programa, y hace que el HTML resultante se guarde en la carpeta resultado, como en la anterior opción, y en la carpeta predeterminada de apache2 /var/www/html/, la cual nos facilitaría para visualizarlo desde el navegador, habiendo levantado el servicio de apache2, indicando simplemente localhost.

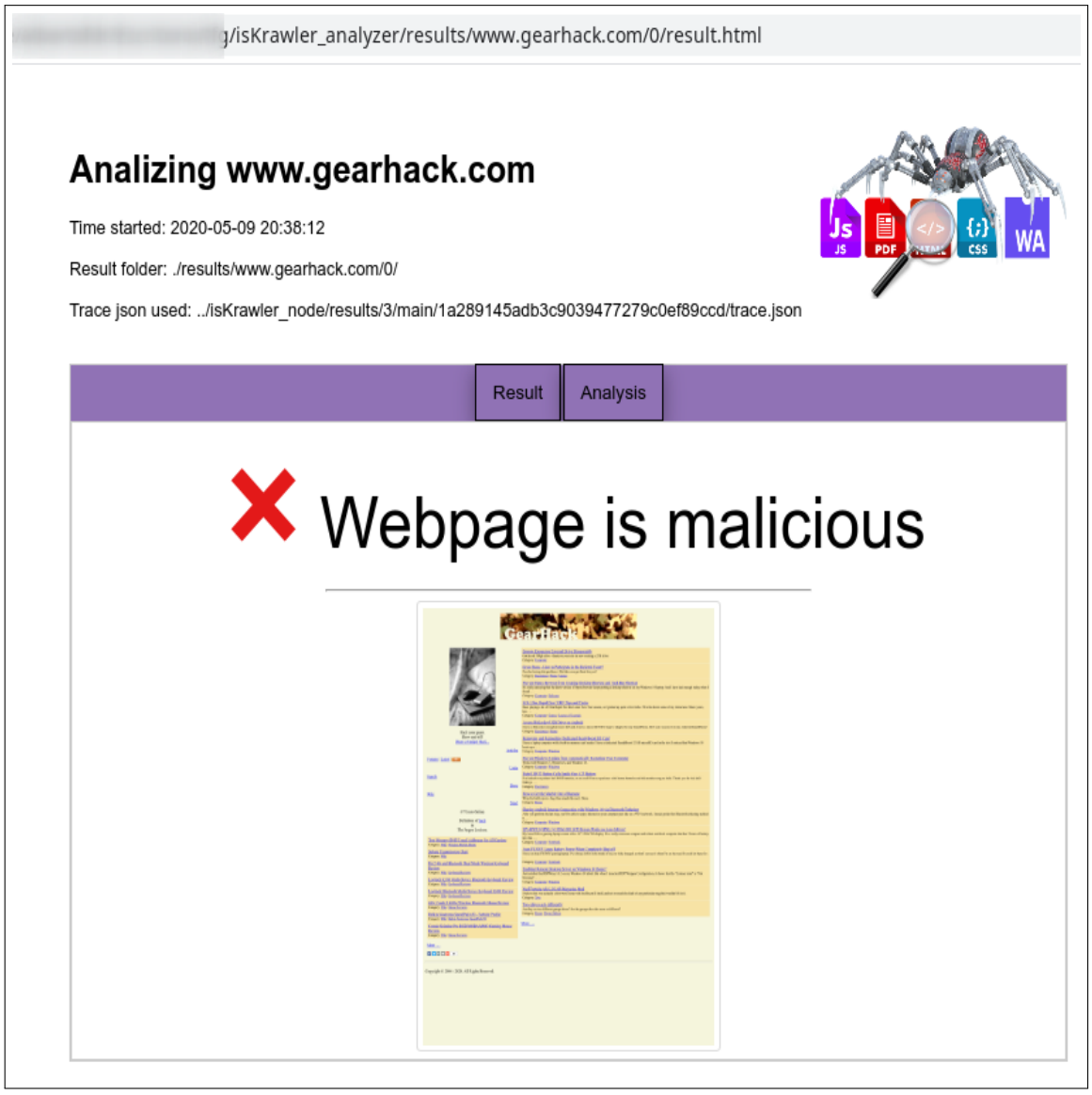


Fig. 4.13: Ejemplo de la exportación HTML (Web: gearhack.com).

Por último, tenemos la opción de guardar los resultado en un base de datos. Esta base de datos puede estar en local o en Heroku [92] como está planteado el proyecto. Esta base de datos nos permite guardar información hasta cierta cantidad siendo gratuita. Para realizar esto, hacemos uso de la biblioteca de Python SQLAlchemy [86] que nos permite realizar esta interacción con la base de datos. Esta opción es la más útil para la investigación y para la recopilación una gran cantidad de resultados, nos permite lanzar el analizador sobre una gran cantidad de datos recopilados anteriormente y que se vayan guardando en la base de datos automáticamente para después revisar alguna característica en concreto. Está es una opción que no es difícil de mantener pero que no

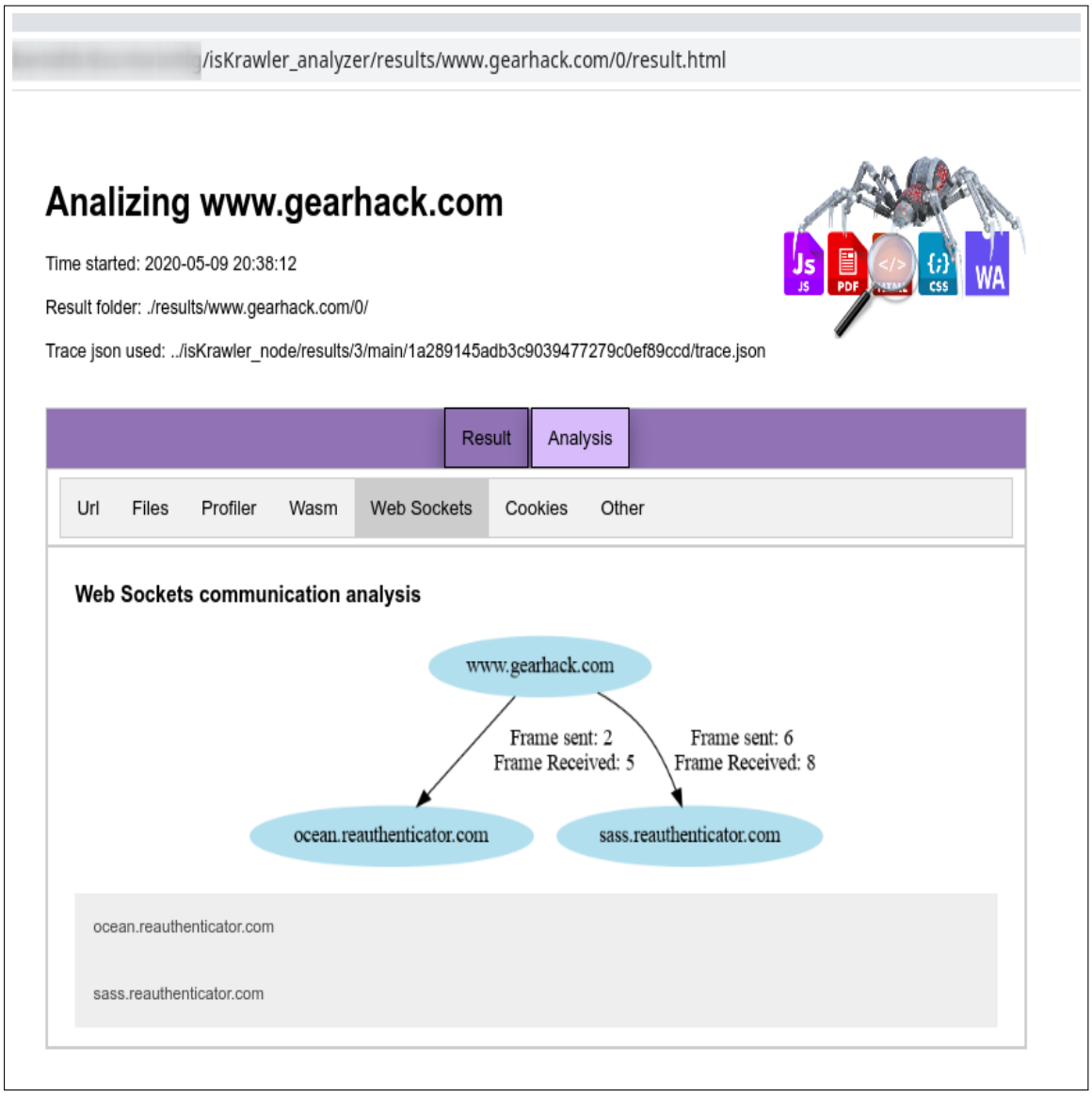


Fig. 4.14: Segundo ejemplo de la exportación HTML (Web: gearhack.com).

abarca toda la información que recopila el analizador, como pueden ser los diagramas o ficheros analizados resultantes. En la figura 4.15 podemos ver como ha sido configurado para este proyecto y para la recopilación de los datos.

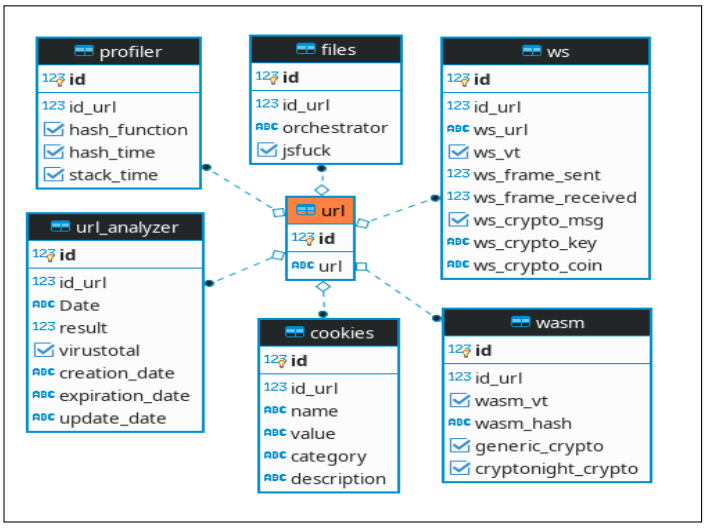


Fig. 4.15: Diagrama Entidad-Relación.

En esta sección, se ha explicado todas las opciones que nos permite el IsKrawler Analyzer para exportar los resultados. Con todas estas opciones, lo que se pretende es que el analizador sea lo más polivalente posible adaptándose a las necesidades del usuario. Según la forma en la que lo queramos usar y dependiendo del contexto, se podrá hacer uso de una o de otra opción. También dependiendo del fin que le queramos dar a los resultados optaremos por una o por otra según nos convenga.

4.2.3. Evaluación de los análisis

Se tratará en este apartado las pruebas realizadas de la herramienta en un entorno real. Para llevar a cabo esto, se han realizado tres tipos de prueba distinta, la primera se realizará sobre sitios web de Alexa Top [4.2.3.1], la segunda sobre páginas web relacionadas con el SARS-CoV-2 [4.2.3.2] y por último sobre sitios web recopilados por el proyecto Minesweeper [16], en el que podemos encontrar una serie de URL de páginas que realizaban cryptojacking [4.2.3.3].

La metodología que se ha utilizado para esta sección, ha sido la siguiente, primero se lanzó el IsKrawler sobre cada una de las listas de los sitios web, recorriéndola y recopilando los resultados navegación por cada URL. Esta etapa fue realizada la primera semana de noviembre como está indicado en los diagramas de Gantt [5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 5.10]. Posteriormente, sobre estos resultados se lanzará el IsKrawler Analyzer, el cual, los analizará, aportándonos un resultado exhaustivo en formato “HTML” y los más importante, guardando lo más relevante del análisis, como se puede ver en el esquema 4.15, en la base de datos remota de Heroku [92]. Esta segunda etapa se ha realizado entorno a principios de junio como indica el diagrama Gantt [5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 5.10].

Como forma de simplificar y automatizar esta evaluación, compuesta por dos secciones, se realizaron dos scripts sencillos en Bash. Esta automatización se podía haber realizado con funcionalidades tanto del IsKrawler como del analizador, pero debido a la facilidad, se optó por implementar estos pequeños programas. El primer script realizado para el crawler, consta de dos modos, uno para leer líneas aleatorias de una lista, como es el caso de Alexa Top, y en el otro para leerlas una a una, como en los otros dos casos. A continuación, se encuentran estos scripts.

```
#!/bin/bash

# FICHERO A LEER
input="../top-1m.csv"
# NOS PERMITE LLEVAR LA CUENTA DE LOS SITIOS WEB RECOPILADOS
num=1
# NOS PERMITE INDICARLE EL ID DEL CRAWL
crawl=1
# INDICAMOS EL MAXIMO DE CRAWLS HA REALIZAR (En base a los ID)
maxcrawl=10
# INDICAMOS EL MODO
mode=1

# DOS MODOS
#   - Primero: Recorre lineas aleatorias
#   - Segundo: Recorre lineas seguidamente

if [ $mode -eq 1 ]; then
    # MODO UNO (RANDOM)
    while true;
    do
        # Numero de linea aleatorio
        line=$(shuf -n 1 $input);

        # Obtener la URL de la linea
        URL=$(echo $line | awk -F, '{print␣$2}');

        # Print para ir observando el avance
        echo "␣␣[*]␣Crawling␣'$URL'␣---␣ID:␣'$crawl'";

        # Lanzar el IsKrawler con los parametros y la URL aleatoria
        ./isKrawler -C -b google-chrome --ws -t $crawl "http://$URL
            ↪ ";

        # Aumentar el ID del crawl
        crawl=$((crawl + 1));
```

```
        # Sacar por pantalla un aviso, esperar unos segundos para
            ↪ no sobresaturar y limpiar la pantalla
        echo "␣[*]␣Finalizado...";
        sleep 3;
        clear;

        # Comprobar si hemos alcanzado nuestro maximo
        if [ $crawl -eq $maxcrawl ]; then
            # Salir
            exit 0;
        fi
    done

else
    # MODO DOS (Lineas seguidas)
    # Contar total de lineas
    lines=$(wc -l $input);

    # Realizar los crawls leyendo las lineas una por una
    while IFS= read -r line;
    do
        # Printear un aviso
        echo "␣␣[*]␣Crawling␣'$line'␣--␣'$num'/'$lines'";

        # Lanzar el crawler
        ./isKrawler -C -b google-chrome --ws -t $crawl "http://
            ↪ $line/";

        # Aumentar el ID, el numero de crawls y limpiar la pantalla
        crawl=$((crawl + 1));
        num=$((num+1));
        clear;

        # Comprobar si hemos alcanzado nuestro maximo
        if [ $crawl -eq $maxcrawl ]; then
            # Salir
            exit 0;
        fi
    done < $input
fi
```

Listing 4.14: Script para la automatización de la evaluación del IsKrawler.

Por otra parte, para la automatización del analizador, el script es muy similar, simplemente en vez de lanzar el crawler, lanza el IsKrawler Analyzer sobre el conjunto de resultados, del IsKrawler, que le indiquemos. A continuación se muestra el código utilizado para la segunda parte de la evaluación.

```
#!/bin/bash

# ID DESDE EL QUE SE VA A EMPEZAR A COGER LOS RESULTADOS
NUM=1
# ID MAXIMO AL QUE LLEGAR
MAXNUM=1000

# Recorrer y analizar
while true;
do
    # Comprobacion para llegar hasta cierto resultado
    if (($NUM > $MAXNUM)) ;
    then
        exit 0;
    fi

    # Aviso por pantalla
    echo "[*] Escaneo number: '$NUM'"

    # Analisis con los parametros, ademas imprime el banner y el
       ↪ estado del analisis
    python3 isKrawler_analyzer.py --db --html ../isKrawler_node/
       ↪ results/$NUM/main/;

    # Aumentar el numero del analisis, esperar para no sobresaturar
       ↪  y limpiar la terminal
    NUM=$(($NUM+1));
    sleep 2;
    clear;
done
```

Listing 4.15: Script para la automatización de la evaluación del IsKrawler Analyzer.

4.2.3.1. Evaluación en el Top de Alexa (400 Sitios web)

En este apartado de evaluación se ha navegado y recopilado información sobre 400 sitios web aleatorios del top de Alexa [99]. Los datos recopilados, como se ha comentado anteriormente, se realizó entorno a la primera semana de noviembre como está indicado en los diagramas de Gantt [5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 5.10].

Se han incorporado gráficos y tablas para representar los resultados de la forma más limpia posible. En el gráfico 4.16, podemos ver los resultados finales obtenidos de esta evaluación, con casi la totalidad de sitios seguros. Además en la tabla 4.5, se encuentran los datos más relevantes encontrados

en la base de datos de Heroku [92] tras el análisis de todos las páginas web. Por último, podemos ver las razones por las que una de ellas ha dado de resultado "Sospechoso" en la tabla 4.6.

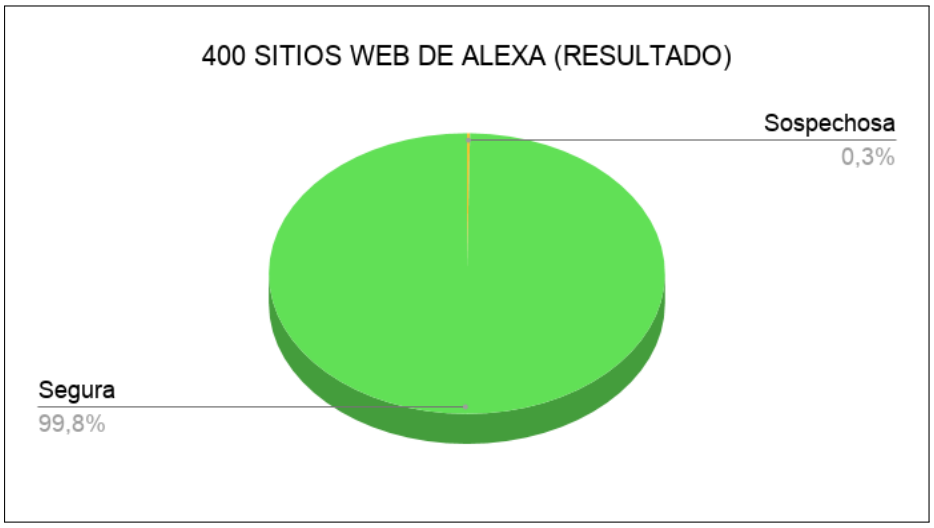


Fig. 4.16: Resultados de Alexa (IsKrawler Analyzer).

	Descripción
Ficheros WebAssembly	0
Conexiones WebSockets	Se han encontrado 29, de las cuales, ninguna positiva en VirusTotal o en nuestro análisis [4.2.2.5]. La mayoría contienen la palabra "Chat".
Análisis del perfilado de la CPU [4.2.2.4]	0
Análisis de ficheros [4.2.2.2]	No se ha encontrado ningún JavaScript con el la técnica JSFuck [111]. Tampoco se ha encontrado ningún orquestador de cryptojacking [4.1].
Cookies	Se han encontrado 4029 cookies, de las cuales 1066 son únicas. Además, de estas 1066 únicas, se han encontrado 148 en el proyecto [116] utilizado en el analizador.
URL positivas en VirusTotal	6

Tabla 4.5: Datos relevantes de la evaluación en el Top Alexa.

URL	**Resultado**	**Motivos**
https://moneysyst.biz	47 (Sospechosa)	Positivo en VirusTotal (1/66) — Resultado en Webutation de 40 — Mozilla header resultado muy bajo.

Tabla 4.6: URL sospechosas del análisis en el Top Alexa.

4.2.3.2. Evaluación en sitios web del SARS-CoV-2(5 Sitios web)

Se han recopilado 5 sitios web relacionados con el SARS-CoV-2, dado que al ser sitios web tan recientes, no es de extrañar, y se han visto casos [40], en los que estas web se aprovechan de los usuarios realizando cualquier tipo de actividad maliciosa, como por ejemplo, fraudes. Por ello, se recopilaron manualmente 5 sitios web relacionados respecto a ello.

Los resultados de esta evaluación, se han expuesto en tablas y gráficos para facilitar la comprensión del lector. En la gráfica 4.17 podemos observar, como una de las web nos ha dado como resultado "Sospechosa". Por otro lado, en la tabla 4.8 podemos observar el resultado numérico y las razones por las que ha dado ese resultado. Por último, en la tabla 4.7 podemos observar los datos más relevantes extraídos del análisis.

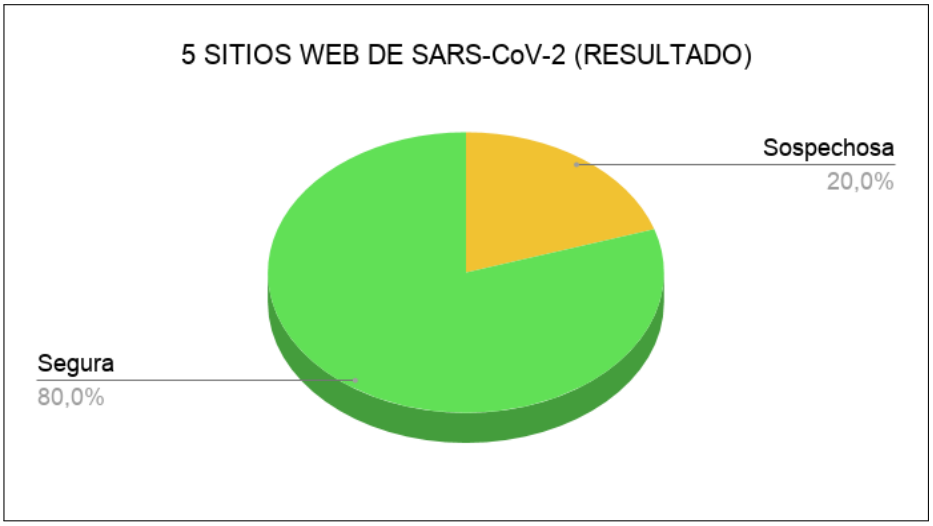


Fig. 4.17: Resultados de los sitios web del SARS-CoV-2 (IsKrawler Analyzer).

	Descripción
Ficheros WebAssembly	0
Conexiones WebSockets	0
Análisis del perfilado de la CPU [4.2.2.4]	0
Análisis de ficheros [4.2.2.2]	No se ha encontrado ningún JavaScript con el la técnica JSFuck [111]. Tampoco se ha encontrado ningún orquestador de cryptojacking [4.1].
Cookies	Se han encontrado 18 cookies, de las cuales 17 son únicas. Además, de estas 17 únicas, se han encontrado 4 en el proyecto [116] utilizado en el analizador.
URL positivas en VirusTotal	3

Tabla 4.7: Datos relevantes de la evaluación de sitios web relacionados con SARS-CoV-2.

URL	Resultado	Motivos
http://2020covid19.best/	37 (Sospechosa)	Positivo en VirusTotal (2/80) — Creada este año (Como es obvio) — No tiene certificado TLS.

Tabla 4.8: URL sospechosas del análisis en el Top Alexa.

4.2.3.3. Evaluación en sitios web que realizaban cryptojacking (830 Sitios web)

En este apartado se ha realizado el análisis más amplio, debido a que nos facilita el encontrar sitios web que hayan sido sospechosos de realizar actividades maliciosas para poder comprobar nuestros resultados, sobretodo ante nuestra imposibilidad de realizar un análisis a gran escala. Debemos tener en cuenta, que esta lista fue creada en el proyecto [16] y publicación científica [12] "MineSweeper: An In-depth Look into Drive-by Cryptocurrency Mining and Its Defense", por lo tanto, la lista es de 2018, por lo que, muchos de los sitios habrán dejado de realizar esas actividades malignas que realizaban, ya sea porque incrustaban recursos de terceros que estaban modificados maliciosamente o porque algún atacante aprovechando alguna vulnerabilidad, lo había incrustado en su momento.

En el gráfico 4.18, podemos observar que alrededor del 97 % de los sitios web, ya no realizan ningún tipo de actividad maliciosa, seguramente por el factor que habíamos comentado. Aún así, se ha obtenido que hay un 1,4 % que dan como resultado que son maliciosas. Se han incluido algunas de estas en la tabla 4.10, en el que se explicitan los motivos y resultados de los análisis. Por otra parte, en la tabla 4.9, podemos encontrar los datos más relevantes respecto al análisis completo.

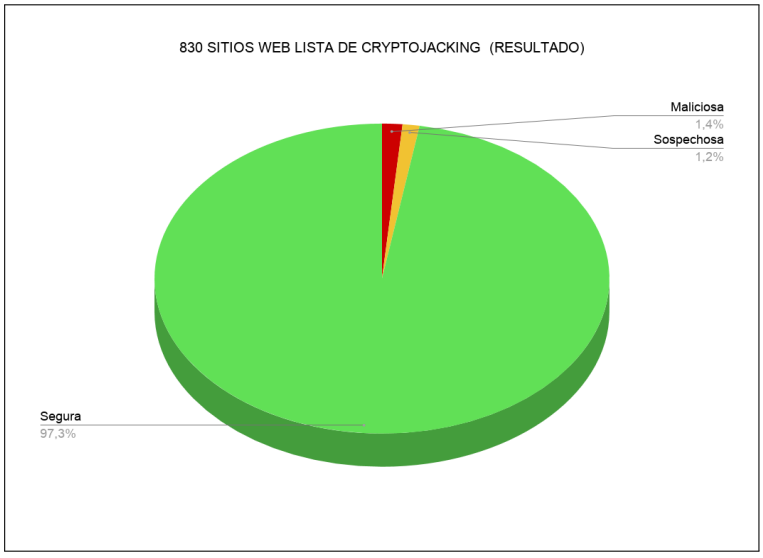


Fig. 4.18: Resultados de los sitios web de la lista que realizaban cryptojacking (IsKrawler Analyzer).

	Descripción
Ficheros WebAssembly	0
Conexiones WebSockets	Se han encontrado 170, de las cuales, 3 positivas tanto en VirusTotal como en nuestro análisis [4.2.2.5]. En una de ellas se ha podido sacar el "site-key" o "id-cliente" y la moneda "xmr".
Análisis del perfilado de la CPU [4.2.2.4]	0
Análisis de ficheros [4.2.2.2]	No se ha encontrado ningún JavaScript con el la técnica JSFuck [111]. Sí se han encontrado 10 orquestadores de cryptojacking [4.1].
Cookies	Se han encontrado 12.921 cookies, de las cuales 2.458 son únicas. Además, de estas 2.458 únicas, se han encontrado 143 en el proyecto [116] utilizado en el analizador.
URL positivas en VirusTotal	235

Tabla 4.9: Datos relevantes de la evaluación de sitios web que realizaban cryptojacking (Fuente: Proyecto Minesweeper [16]).

URL	Resultado	Motivos
http://happyjav.com/	111 (Maliciosa)	Positivo en VirusTotal (1/71) — Contiene orquestador [4.1] — Conexión WebSocket que da positivo en VirusTotal y en nuestro análisis [4.2.2.5] — Contiene 8 Service Workers.
http://sorok-nedel.ru/	62 (Maliciosa)	Positivo en VirusTotal (1/71) — Contiene orquestador [4.1]
http://fonts4web.ru/	57 (Maliciosa)	Positivo en VirusTotal (4/79) — Contiene orquestador [4.1]

Tabla 4.10: URL sospechosas del análisis en el Top Alexa.

Capítulo 5

Metodología y planificación

En este capítulo, se tratará todo lo relacionado con la metodología y planificación seguida en el proyecto. La metodología, al igual que la sección de planificación, se dividirá en distintos apartados para abarcar toda la información posible y poder tratarlo todo organizadamente.

5.1. Metodología

En este proyecto, para realizar un buen desempeño y elegido por las condiciones del mismo se han utilizado dos metodología, una para las adiciones en el crawler y otra diferente para el analizador. Respecto a la metodología seguida para las modificaciones y añadidos en el IsKrawler se ha hecho uso de la metodología de desarrollo en espiral, el cual es una mezcla en del modelo en cascada y el modelo iterativo.

Esta metodología se definió por primera vez en 1986 por Barry Boehm y se ha utilizado enormemente en lo que conocemos como ingeniería del software [125]. Se ha utilizado debido a tener que hacer adiciones sobre un proyecto tan grande y largo como es el crawler, en el que cualquier adición sin haberlo planificado podría desembocar en grandes problemas en otros módulos del mismo. Al tener que realizar diversas adiciones en este proyecto se optó por este modelo en el que existe una fase de análisis de los riesgos a la hora de añadir nuevos desarrollos evitando futuros problemas en el desarrollo.

Las ventajas que tiene esta metodología frente a otros tipos de desarrollos, es que está pensada para proyectos grandes que están sujetos a riesgos, para proyectos técnicos, proyectos que se desarrollan incrementalmente y constantemente, y una de las más importantes respecto a otros modelos es que incorpora una fase explícita para el análisis de riesgos. Esta metodología permitió el poder tener bien definidas las fases a realizar a la hora de añadir cada uno de los nuevos módulos. Las cuatros fases bien definidas en esta metodología son:

- **Planificación**: En esta fase se determina el objetivo y alcance haciendo una investigación previa. Con cada iteración que se realice, esta fase será más complicada debido a que ira aumentando el tamaño del proyecto y sus funcionalidades.

- **Análisis de Riesgo**: En este momento, se examina y evalúa todo tipo de variantes que puedan afectar al proyecto, es decir, se evaluarán todos los módulos que pudieran ser afectados por el desarrollo.
- **Implementación**: En este instante, se desarrolla y valida el código según lo acordado en las fases anteriores teniendo en cuenta, sobretodo, el análisis de riesgo a la hora de realizarlo.
- **Evaluación**: Antes de que podamos continuar con las iteraciones y planificar otro desarrollo, debemos analizar lo ocurrido en esta iteración, examinar los riesgos que habíamos detectado y de qué forma les hemos puesto remedio, así como determinar cual será la próxima iteración.

En el caso del analizador, se ha hecho uso de la metodología Kanban. El término Kanban proviene del japonés y traducida, significa tarjeta de signos o señal visual [126], lo cual es bastante explicatorio respecto a la metodología. Es una metodología muy visual y sencilla de utilizar, que básicamente consiste en colocar las tareas en distintas columnas dependiendo de su estado. Esta metodología cuenta con distintas fases explicadas a continuación:

- **Por hacer (To-do)**: En esta fase se encuentras las tareas que por prioridad o por alguna otra razón están sin hacer.
- **En progreso**: En este momento, las tarjetas que se encuentran en este apartado, es que se han dado comienzo a realizarlas y se encuentran realizándose.
- **A probar**: Aquí se encuentran las tareas o tarjetas que están desarrolladas pero que faltan de probar su perfecto funcionamiento.
- **Realizadas**: En esta sección se encuentran todas las tareas que ya han sido implementadas y comprobadas para su correcto funcionamiento, dado fin a su ciclo de vida.

Esta metodología nos hace aumentar el rendimiento, monitorizando las tareas y realizando los ajustes apropiados en cada momento para poder aumentar nuestra eficiencia. Otra ventaja de esta metodología es la facilidad para aprender a utilizarla y lo visual que es. Nos permite también tener en todo momento en qué estado se encuentra cada tarea y tratando de eliminar los problemas implícitos a realizar multitareas, permitiéndonos focalizarnos en una única tarea. Para realizar este diagrama, se comenzó realizándolo en papel y posteriormente se decidió seguir la metodología con alguna herramienta. Para ello, y tras unas búsquedas, se optó que la mejor solución, sabiendo que es gratuita y su facilidad, y la más lógica, dado que el proyecto se encuentra en Github, era hacer uso de Github projects [127]. Esta herramienta nos permite realizar un Kanban asignado a un repositorio organizándonos todo de forma sencilla. En la figura 5.1 se muestra el ejemplo de cómo se ha realizado esta metodología y sus tareas.

5.2. Planificación

En esta sección se tratará la planificación que se ha llevado a cabo para realizar el proyecto. Se abordarán todos los puntos relacionados con la planificación, aportando en alguno de ellos diagramas y figuras explicativas. Se ha dividido en distintas secciones para poder tratar en amplitud el tema.

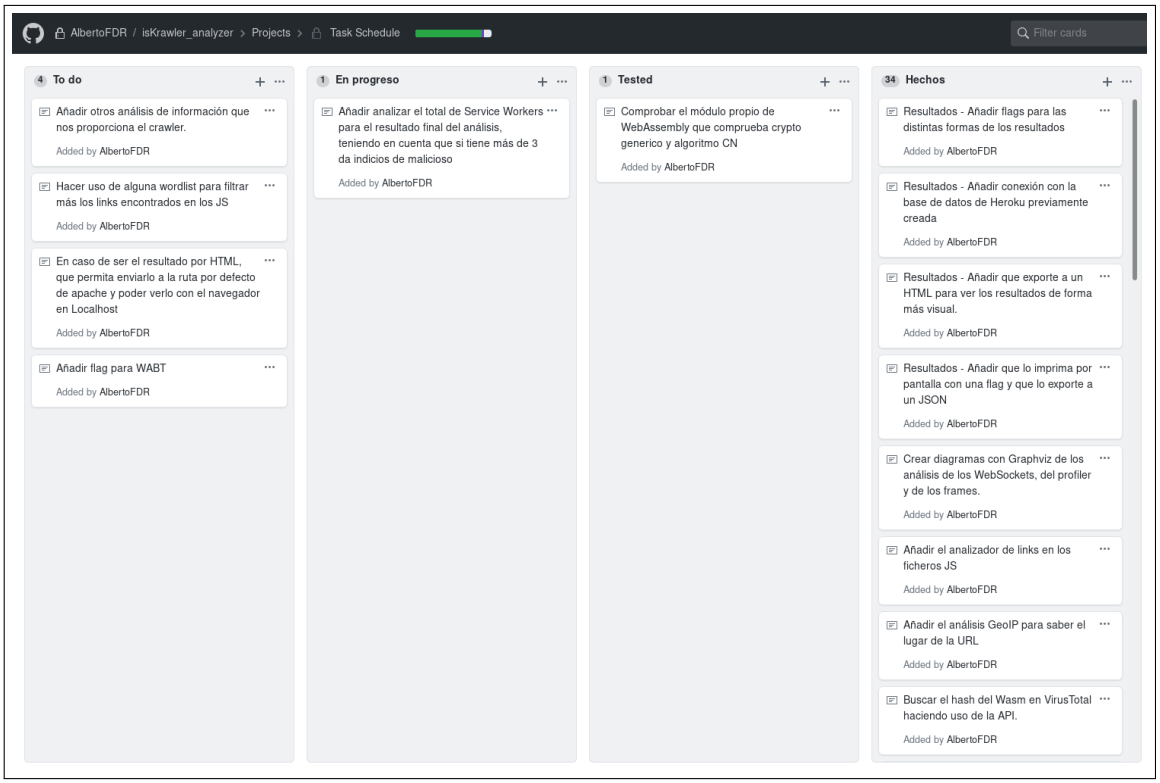


Fig. 5.1: Analizador metodología Kanban.

5.2.1. Fases del proyecto

Las fases del proyecto estarán divididas en 3 grandes bloques. El primero abarca la investigación y formación necesaria para realizar el proyecto. El segundo nos permitirá mejorar el crawler existente y el último, contendrá la mayor parte del proyecto, siendo el diseño y desarrollo del analizador. Se puede observar en la tablas 5.1 la duración de cada una de ellas que se profundizará en sus correspondientes apartados.

Nombre	Duración (días)
Estudios de las conferencias de seguridad web	71
Análisis del crawler existente (IsKrawler)	21
IsKrawler Analyzer	108

Tabla 5.1: Tareas del proyecto.

5.2.1.1. Estudios de las conferencias de seguridad web

Estudios de las conferencias de seguridad web	Fecha de inicio	Fecha de fin
Planeado	10/10/2019	16/01/2020
Real	12/10/2019	14/01/2020

Tabla 5.2: Fechas de la fase de estudios de las conferencias de seguridad web.

En esta fase se componen todas las tareas de investigación y formación para la realización del proyecto. El proyecto necesitará unas tareas previas de formación y de lectura para su realización, tanto para la parte del crawler como para el desarrollo del analizador. Esta fase estará divido en las siguientes tareas:

1. **Recopilación de estudios científicos sobre la seguridad web**: Esta tarea consiste en recorrer las conferencias de seguridad web más importante e ir recopilando los estudios que tuvieran algún tipo de relación con este proyecto y que fueran interesantes. El criterio más determinante para esta recopilación es que fuese un proyecto similar del que se pudieran extraer ideas.

2. **Recopilación de estudios sobre el cryptojacking**: Un tema interesante sobre de la seguridad web, que tuvo su auge en 2018, era el cryptojacking y en muchos aspectos muy relacionada con nuestro proyecto. Por otra parte, era una temática de bastante interés para implementar en el proyecto y es por eso por lo que se incluye esta tarea, en la que se recopilan estudios de las conferencias sobre este tema.

3. **Síntesis de los estudios y recopilación de ideas para el proyecto**: En este momento, debemos leer todos los estudios recopilados e ir sintetizandolos para.

4. **Análisis de los proyectos utilizados en los estudios en caso de estar liberados**: En esta tarea, se debe analizar en detalle el trabajo que han realizado para llevar a cabo el estudio, dicho de otro modo, con el documento de la conferencia nos podemos hacer una idea abstracta pero necesitamos observar como han implementado esa idea en código.

5.2.1.2. Análisis del crawler existente (IsKrawler)

Análisis del crawler existente	Fecha de inicio	Fecha de fin
Planeado	10/10/2019	07/11/2019
Real	5/10/2019	17/11/2019

Tabla 5.3: Fechas de la fase de análisis del crawler existente.

En esta sección, se explicará todas las tareas relacionadas con el IsKrawler. Dado que necesitamos hacer modificaciones y pruebas del crawler para poder recopilar la máxima información posible de la página a analizar para después desarrollar el isKrawler Analyzer. La tareas que componen esta fase son las siguientes:

5. **Implementación de la tecnología WebSocket en el crawler**: El crawler no tenía la capacidad de recopilar estas comunicaciones TCP, entendiendo, que en el momento que se desarrolló todavía no era una tecnología en uso. Por ello, en esta tarea se implantará los métodos necesarios para que el IsKrawler sea capaz de recoger con que sitio se está comunicando, además de todos los frames enviados y recibidos.

6. **Implementación de la recogida de ficheros WebAssembly (Wasm)**: Esta tarea consistirá en la modificación del IsKrawler para que por defecto sea capaz también de recopilar los ficheros Wasm. Para ello deberemos. primero, identificar de qué forma los reconoce el crawler haciendo uso del protocolo de depuración de Chrome, y posteriormente hacer las modificaciones pertinentes en el mismo para que sea capaz de descargarlos.

7. **Implementación del perfilado del procesador en el navegador**: Se añadirán las funciones necesarias para hacer el perfilado del procesador, haciendo uso de la documentación del protocolo de depuración [96]. Esto consistirá en habilitar el perfilado, indicarle que comience antes de navegar sobre la página y que finalice, devolviendo los resultados, tras la misma.

8. **Pruebas del perfecto funcionamiento**: En esta tarea, el desempeño tratará en hacer uso del IsKrawler sobre webs que sabemos que hacen uso de alguna de estas tecnologías. Después, debemos analizar los resultados, examinando que todas las implementaciones realizan perfectamente su trabajo.

9. **Período de crawling web para obtener datos**: Como forma de probar el proyecto completo, y ateniéndonos a los tiempos por eso se coloca en este grupo, esta fase consistirá en lanzar el IsKrawler durante determinado tiempo recorriendo webs. Para ello, y por seguridad, todo el crawling será hecho desde una máquina virtual.

5.2.1.3. isKrawler Analyzer

isKrawler Analyzer	Fecha de inicio	Fecha de fin
Planeado	17/01/2020	16/06/2020
Real	25/01/2020	12/06/2020

Tabla 5.4: Fechas de la fase del analizador isKrawler Analyzer.

Este apartado contendrá la mayor cantidad de tareas y de tiempo. En esta sección, se engloban todas labores relacionadas con el analizador de los resultados del IsKrawler. Dicho de otro modo, el cometido de este grupo es realizar el IsKrawler Analyzer. Todas las labores que componen este apartado, estarán supeditadas a la finalización de la etapa los estudios de las conferencias [5.2.1.1] y a las pruebas del perfecto funcionamiento del crawler [8]. Esta fase contendrá las siguientes actividades:

10. **Creación del proyecto**: Crear el proyecto de forma local, configurando el entorno de trabajo y el entorno de desarrollo para el mismo. También se añade en Github un repositorio para poder ir teniendo versiones y backups de lo realizado.

11. **Diseño de la estructura**: Esta tarea consistirá en el diseño general del IsKrawler Analyzer, diseñando una desarrollo modular pensando en posibles modificaciones, eliminaciones o adiciones del mismo.

12. **Diseño e implementación de la entrada de datos y de los posibles argumentos**: Boceto y planificación de la entrada de los distintos parámetros y su posterior desarrollo.

13. **Diseño e implementación de los modelos a usar**: Planificación del modelo de datos a usar para poder interpretar los resultados del IsKrawler y de los resultados del analizador. Posteriormente, realizar la codificación de lo planteado.

14. **Diseño y desarrollo del análisis de la URL**: Esta tarea se basará en la planificación del módulo mencionado. También se incluye en esta fase el desarrollo del mismo.

15. **Diseño y desarrollo del análisis de los ficheros Wasm (WebAssembly)**: Se diseña e implementan los módulos referidos a este tipo de tecnología.

16. **Diseño e implementación del análisis de ficheros**: Se planea y desarrolla el módulo encargado de los recursos de la web, que son descargados por el IsKrawler.

17. **Diseño e implementación del análisis de la perfilación del procesador**: Esta tarea consistirá en el diseño y posterior desarrollo de la observación de los resultados emitidos por el crawler respecto al procesador.

18. **Diseño y desarrollo del análisis de las conexiones mediante WebSockets**: En esta etapa se planificará y desarrollará el módulo que desarrollará la examinación de todas las comunicaciones sucedidas mediante WebSockets.

19. **Implementación de APIs y de otras bibliotecas**: Esta tarea se basará en la implementación y gestión de todas las APIs y otro tipo de bibliotecas de Python que se haga uso. Esta es una etapa que estará estrechamente relacionada con el desarrollo de cada uno de los módulos.

20. **Implementación de la salida de los resultados (JSON, HTML, BD y Verbose)**: La última etapa de desarrollo y diseña será la de implementar diversas formas de proporcionar los resultados dependiendo de los parámetros utilizados a la hora de ejecutar el programa. Esta tarea englobará las cuatro formas pero luego cada una de ellas es completamente distinta al resto.

21. **Pruebas del perfecto funcionamiento**: Esta tarea consistirá en comprobar que cada una de las etapas anteriores y de análisis realiza lo que se espera de él. Estas pruebas ya se han realizado individualmente a la hora de realizar cada módulo pero no se han llegado a probar conjuntamente.

22. **Prueba del funcionamiento sobre los datos anteriormente recopilados**: Finalmente, se hará uso de los datos recopilados en la etapa del crawler existente [8] para analizar los distintos resultados. Esta labor, al igual que la del crawling, se llevará a cabo en la máquina virtual que habíamos mencionado.

5.2.2. EDT

En este apartado, y dado que anteriormente ya se han definido todas las tareas del proyecto, crearemos el diagrama de la estructura de descomposición del trabajo (EDT). Es una herramienta fundamental que se basa en la descomposición jerárquica de las tareas. El propósito de esta técnica es organizar el alcance del proyecto y representar gráficamente el proyecto de forma muy descriptiva. Para ello hemos hecho uso del sitio web lucidchart [128]. En la figura 5.2 se muestra el dicho diagrama.

5.2.3. Organización y perfiles

Este trabajo, es un Proyecto de Fin de Grado individual, por lo tanto, está formado, principalmente por el estudiante, autor del presente documento, y por el director del proyecto. Este estudiante será el encargado de realizar todos los perfiles descritos y necesarios, salvo el de director. Se describen, a continuación, los distintos perfiles que se han llevado a cabo para realizar el proyecto:

1. **Desarrollador**: Este perfil ha sido necesario para desarrollar una gran parte del proyecto. En este perfil se necesitaba tener conocimientos en Python (IsKrawler Analyzer) y en JavaScript (IsKrawler) para desarrollarlos. Es una de las tareas fundamentales del proyecto debido a toda la gestión de ficheros, bibliotecas, etc... que se hace dentro del proyecto, además de todos los análisis.

2. **Investigador**: El perfil de investigador a medida que avanzaba el proyecto perdía la importancia en el mismo. La función de este papel ha sido la de leer documentos y proyectos de conferencias y analizarlos detenidamente. Además se ha realizado una gran labor a la hora de buscar en todos los congresos, cuales nos podían servir de referencia.

3. **Director**: Este perfil, es el único no realizado por el estudiante dado que ha sido el director del proyector el encargado esta función, en este caso, David Buján. La función, ha sido la de aconsejar, acompañar o dirigir el proyecto para que llegase a buen puerto, ayudando al estudiante en todo lo necesario, sobretodo, en el momento de poder enfocar de una manera buena el proyecto.

4. **Gestor del proyecto**: Este rol es el encargado, además del director, de gestionar el proyecto desde la fase inicial hasta la final. Se asegura de que todas las tareas estén planificadas y se van realizando en los tiempos estimados. Por otra parte, es el perfil encargado de elegir la metodología adecuada para cada proyecto.

5. **Ingeniero del software**: La función de este papel, por una parte, ha sido la del análisis de un sistema existente como ha sido el IsKrawler y por la otra, ha sido la de diseñar un analizador desde cero. También, dentro de este rol, es encargado de que se vaya realizando todo a su tiempo.

6. **Desarrollador web**: Este es el perfil necesario para añadir la plantilla de Jinja2 [88], la cual es una plantilla de HTML, CSS y JavaScript. Se necesita para poder añadir al analizador la opción de exportar el resultado en HTML. Se necesitará conocimientos en HTML, Python, CSS y JavaScript, para realizar una plantilla dinámica e intuitiva para el usuario y que permita visualizar todos los resultados, desde los datos hasta los diagramas. Sin ser una

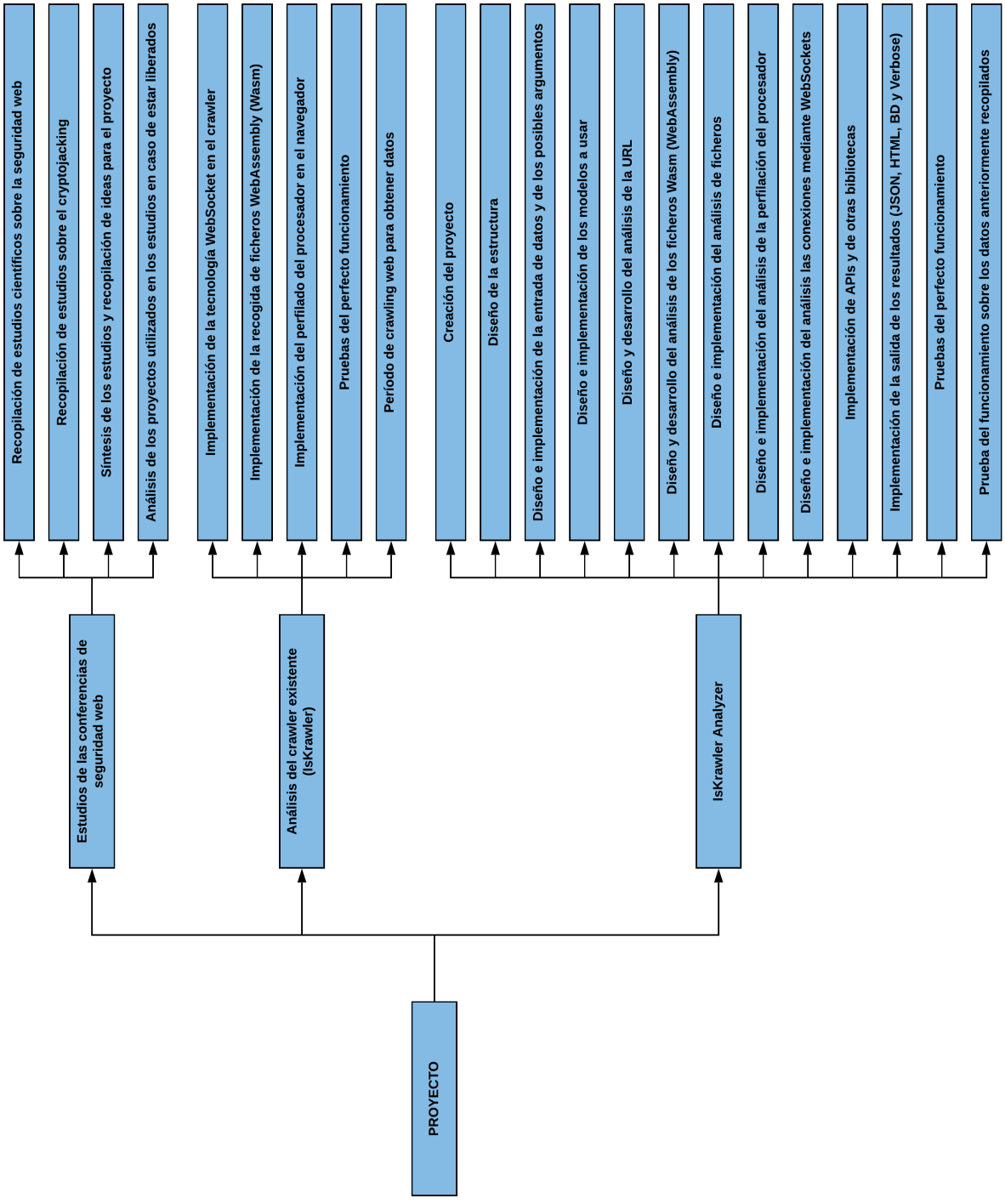


Fig. 5.2: Diagrama EDT.

parte fundamental o central del proyecto, ha sido uno de los perfiles más recurridos en la realización del mismo.

7. **Desarrollador de bases de datos**: Esta función ha sido una de las menos importantes del proyecto, dado que solo se utiliza en el caso de exportarlo a una base de datos como es la de Heroku [92]. Aunque las conexiones con la base de datos se hayan gestionado con un ORM como es SQLAlchemy [86], se han necesitado nociones para ir visualizando y comprobando lo realizado en la base de datos con programas cliente como DBeaver [129].

8. **Administrador de sistemas**: Por último, se han necesitado unos conocimientos mínimos para gestionar tanto el desarrollo del proyecto completo como la puesta en marcha de un entorno de pruebas. Aún así, ha sido el perfil que menos tiempo ha requerido de todos los roles nombrados anteriormente.

5.2.4. Condiciones de ejecución

En este apartado se describe las condiciones en las que se ha desarrollado el proyecto. El lugar de trabajo habitual han sido, en primera instancia, el domicilio del desarrollador del proyecto y en segunda instancia, las instalaciones del laboratorio de investigación de Deusto (DeustoTech).

El horario que se ha utilizado para el proyecto, y las fechas en las que se ha trabajado en el mismo, han ido variando enormemente dependiendo del momento. En algunas ocasiones el estudiante se ha podido dedicar intensamente al proyecto, realizando un horario de mañana y uno de tarde, mientras que en otras épocas del año no ha podido trabajar en el mismo. Las mayor razón por la que no se ha podido realizar un trabajo continuado ha sido por los quehaceres del resto de asignaturas de la universidad. En términos generales, se ha tratado de planificar lo máximo el proyecto y de realizar un trabajo continuado a lo largo de la duración del mismo.

Otro aspecto a tener en cuenta en este apartado, han sido el hardware y software utilizado para el proyecto. Sin importar el lugar de trabajo siempre se ha hecho uso del portátil del estudiante para realizarlo, tanto para el desarrollo de todo el proyecto como para las pruebas del mismo. Por otra parte, en el resto de casos, tanto de software utilizado como de la base de datos, se ha optado por soluciones gratuitas dado que el proyecto no tenía un presupuesto real para financiarlo. Aún así, en la mayoría de casos no ha sido ninguna perdida no poder utilizar opciones pago dado que siempre suele haber alternativas libres de muy buena calidad.

5.2.5. Diagrama de precedencias

En este apartado, y dado que anteriormente ya se han definido todas las tareas del proyecto, damos paso a entender las precedencias de cada tarea. Para ello hemos hecho uso del sitio web lucidchart [128]. En la figura 5.3 se muestra las precedencias de cada tarea de una forma visual y simple.

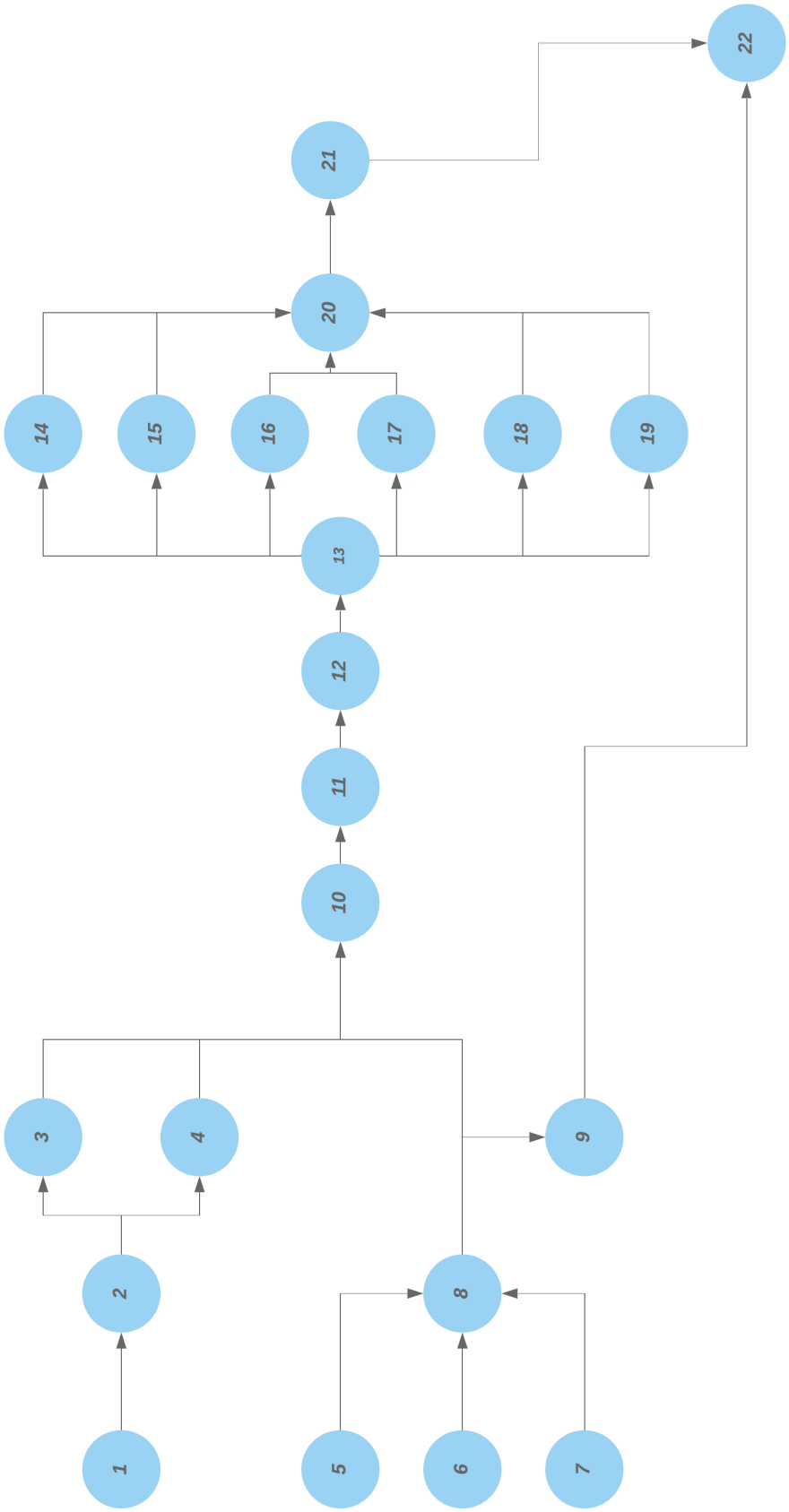
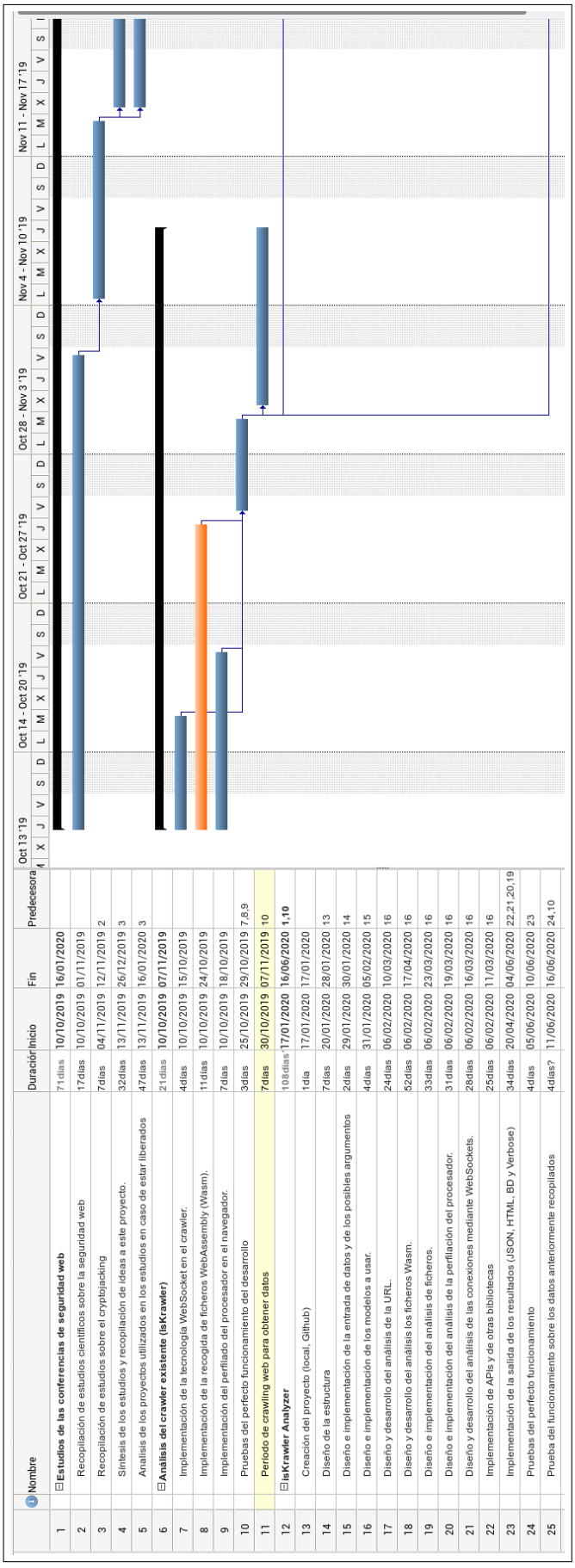


Fig. 5.3: Diagrama de precedencias para este proyecto.

5.2.6. Diagrama de Gantt

Los diagramas Gantt nos permite visualmente exponer el tiempo de dedicación para diferentes tareas, en este caso, el desarrollo del proyecto. Los diagramas para este proyecto se muestran en las figuras 5.4, 5.5, 5.6, 5.7, 5.8, 5.9 y 5.10. En este caso debemos incluir una gran cantidad de figuras sobre el diagrama debido a que ha sido un proyecto que dio comienzo antes de dar comienzo la asignatura del "Proyecto fin de grado". Para realizar estos gráficos se ha hecho uso del sitio web online llamado Gantter [130].



	Nombre	Duración	Inicio	Fin	Predecesora
1	**Estudios de las conferencias de seguridad web**	71días	10/10/2019	16/01/2020	
2	Recopilación de estudios científicos sobre la seguridad web	17días	10/10/2019	01/11/2019	
3	Recopilación de estudios sobre el cryptojacking	7días	04/11/2019	12/11/2019	2
4	Síntesis de los estudios y recopilación de ideas a este proyecto.	32días	13/11/2019	26/12/2019	3
5	Análisis de los proyectos utilizados en los estudios en caso de estar liberados	47días	13/11/2019	16/01/2020	3
6	**Análisis del crawler existente (isKrawler)**	21días	10/10/2019	07/11/2019	
7	Implementación de la tecnología WebSocket en el crawler.	4días	10/10/2019	15/10/2019	
8	Implementación de la recogida de ficheros WebAssembly (Wasm).	11días	10/10/2019	24/10/2019	
9	Implementación del perfilado del procesador en el navegador.	7días	10/10/2019	18/10/2019	
10	Pruebas del perfecto funcionamiento del desarrollo	3días	25/10/2019	29/10/2019	7,8,9
11	Período de crawling web para obtener datos	7días	30/10/2019	07/11/2019	10
12	**isKrawler Analyzer**	108días	17/01/2020	16/06/2020	**1,10**
13	Creación del proyecto (local, Github)	1día	17/01/2020	17/01/2020	
14	Diseño de la estructura	7días	20/01/2020	28/01/2020	13
15	Diseño e implementación de la entrada de datos y de los posibles argumentos	2días	29/01/2020	30/01/2020	14
16	Diseño e implementación de los modelos a usar.	4días	31/01/2020	05/02/2020	15
17	Diseño y desarrollo del análisis de la URL.	24días	06/02/2020	10/03/2020	16
18	Diseño y desarrollo del análisis los ficheros Wasm.	52días	06/02/2020	17/04/2020	16
19	Diseño e implementación del análisis de ficheros.	33días	06/02/2020	23/03/2020	16
20	Diseño e implementación del análisis de la perfilación del procesador.	31días	06/02/2020	19/03/2020	16
21	Diseño y desarrollo del análisis de las conexiones mediante WebSockets.	28días	06/02/2020	16/03/2020	16
22	Implementación de APIs y de otras bibliotecas	25días	06/02/2020	11/03/2020	16
23	Implementación de la salida de los resultados (JSON, HTML, BD y Verbose)	34días	20/04/2020	04/06/2020	22,21,20,19
24	Pruebas del perfecto funcionamiento	4días	05/06/2020	10/06/2020	23
25	Prueba del funcionamiento sobre los datos anteriormente recopilados.	4días?	11/06/2020	16/06/2020	24,10

Fig. 5.4: Diagrama de Gantt para este proyecto 1/7.

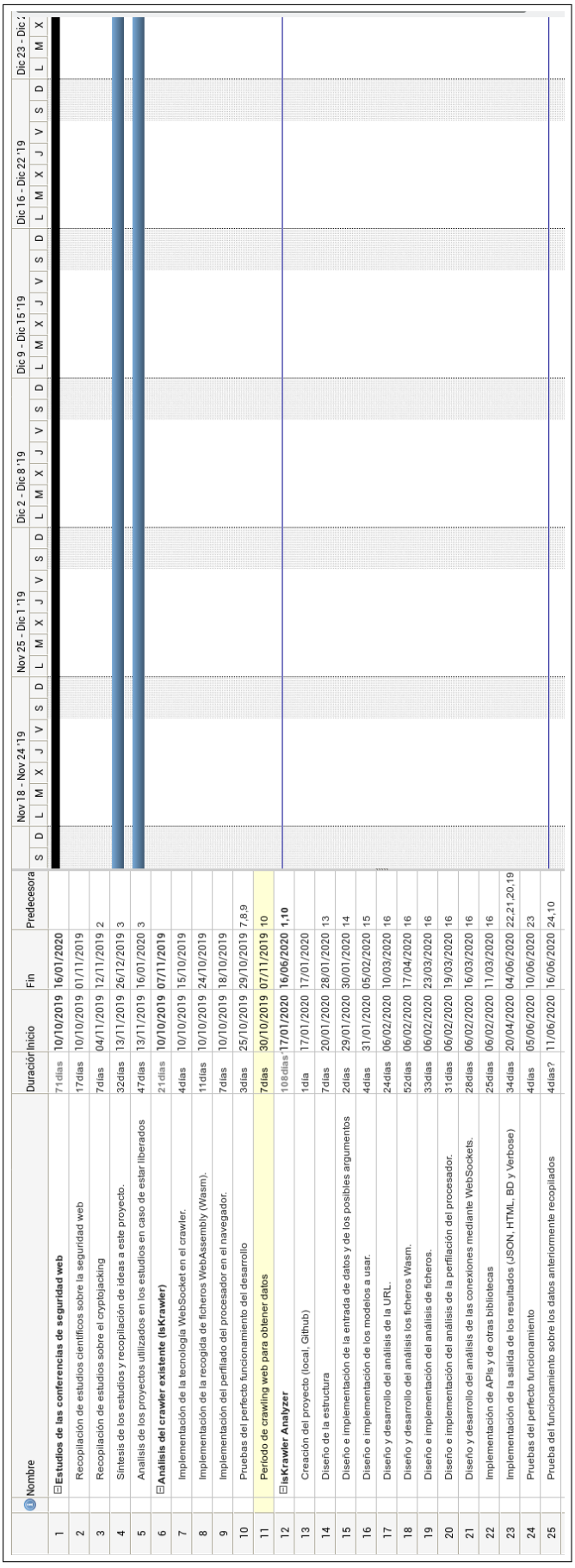
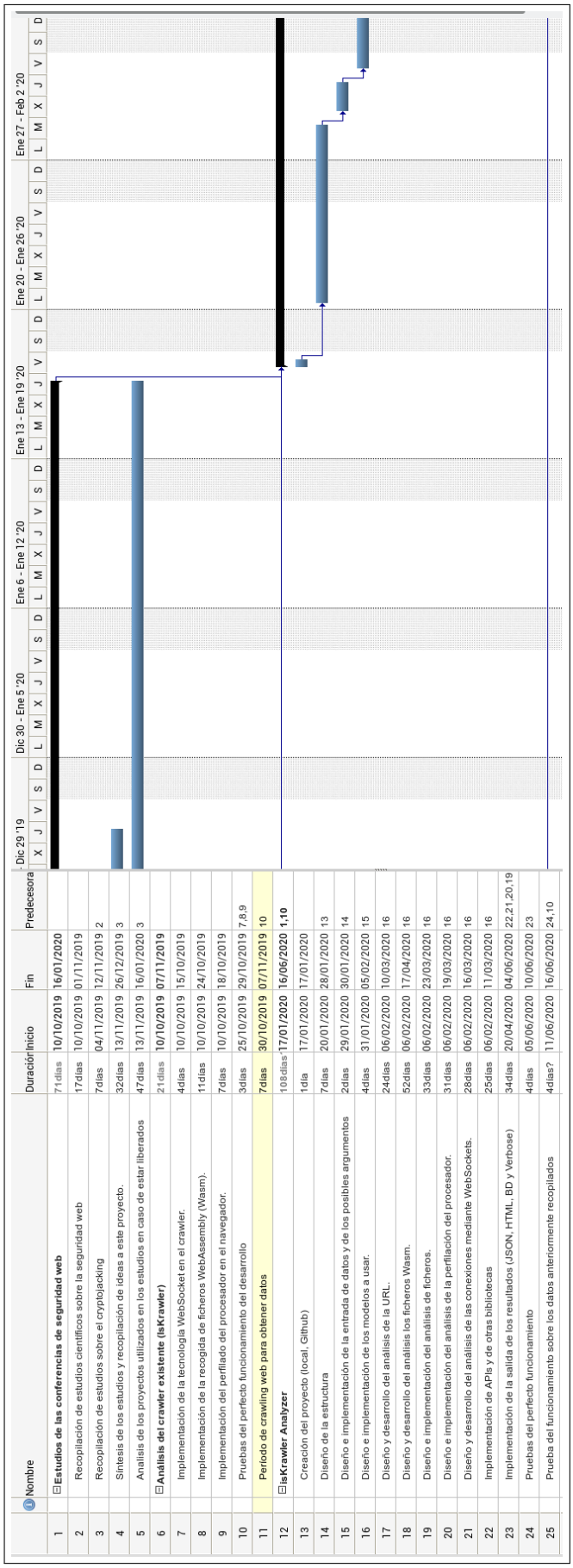
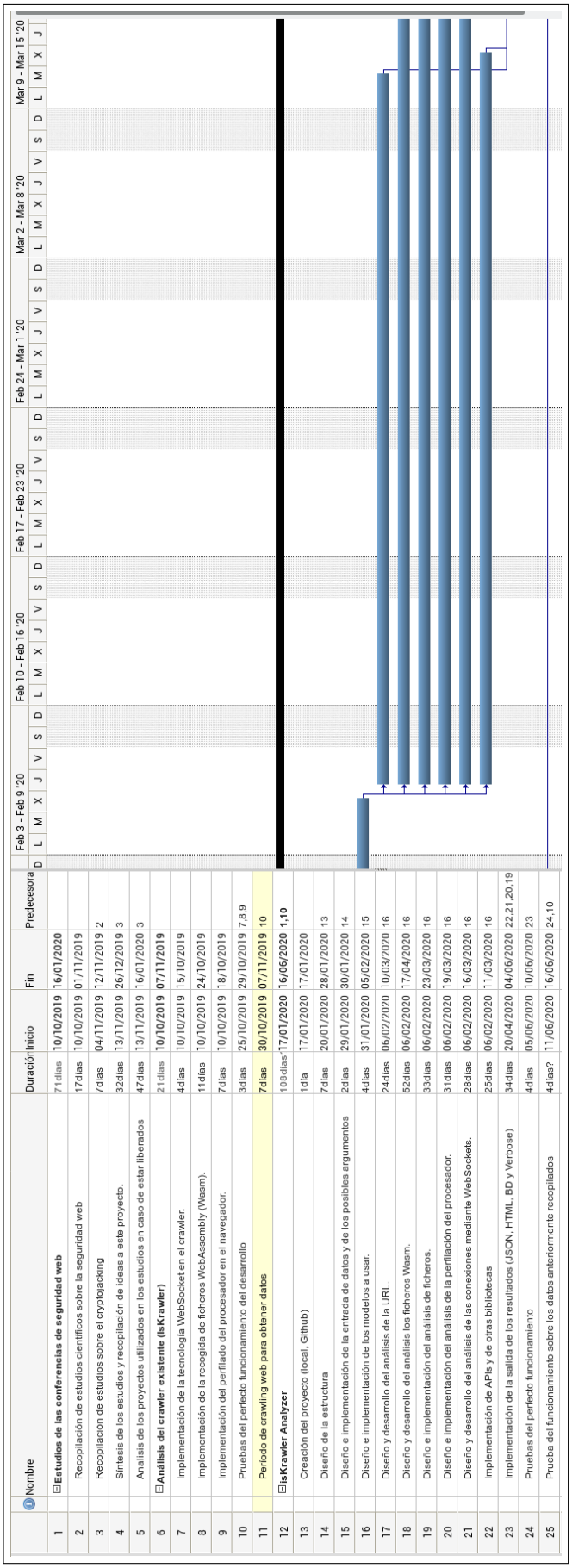


Fig. 5.5: Diagrama de Gantt para este proyecto 2/7.



	Nombre	Duración	Inicio	Fin	Predecesora
1	**Estudios de las conferencias de seguridad web**	71días	10/10/2019	16/01/2020	
2	Recopilación de estudios científicos sobre la seguridad web	17días	10/10/2019	01/11/2019	
3	Recopilación de estudios sobre el cryptojacking	7días	04/11/2019	12/11/2019	2
4	Síntesis de los estudios y recopilación de ideas a este proyecto.	32días	13/11/2019	26/12/2019	3
5	Análisis de los proyectos utilizados en los estudios en caso de estar liberados	47días	13/11/2019	16/01/2020	3
6	**Análisis del crawler existente (IsKrawler)**	21días	10/10/2019	07/11/2019	
7	Implementación de la tecnología WebSocket en el crawler.	4días	10/10/2019	15/10/2019	
8	Implementación de la recogida de ficheros WebAssembly (Wasm).	11días	10/10/2019	24/10/2019	
9	Implementación del perfilado del procesador en el navegador.	7días	10/10/2019	18/10/2019	
10	Pruebas del perfecto funcionamiento del desarrollo	3días	25/10/2019	29/10/2019	7,8,9
11	**Período de crawling web para obtener datos**	7días	30/10/2019	07/11/2019	10
12	**IsKrawler Analyzer**	108días	17/01/2020	16/06/2020	1,10
13	Creación del proyecto (local, Github)	1día	17/01/2020	17/01/2020	
14	Diseño de la estructura	7días	20/01/2020	28/01/2020	13
15	Diseño e implementación de la entrada de datos y de los posibles argumentos	2días	29/01/2020	30/01/2020	14
16	Diseño e implementación de los modelos a usar.	4días	31/01/2020	05/02/2020	15
17	Diseño y desarrollo del análisis de la URL.	24días	06/02/2020	10/03/2020	16
18	Diseño y desarrollo del análisis los ficheros Wasm.	52días	06/02/2020	17/04/2020	16
19	Diseño e implementación del análisis de ficheros.	33días	06/02/2020	23/03/2020	16
20	Diseño e implementación del análisis de la perfilación del procesador.	31días	06/02/2020	19/03/2020	16
21	Diseño y desarrollo del análisis de las conexiones mediante WebSockets.	28días	06/02/2020	16/03/2020	16
22	Implementación de APIs y de otras bibliotecas	25días	06/02/2020	11/03/2020	16
23	Implementación de la salida de los resultados (JSON, HTML, BD y Verbose)	34días	20/04/2020	04/06/2020	22,21,20,19
24	Pruebas del perfecto funcionamiento	4días	05/06/2020	10/06/2020	23
25	Prueba del funcionamiento sobre los datos anteriormente recopilados	4días?	11/06/2020	16/06/2020	24,10

Fig. 5.6: Diagrama de Gantt para este proyecto 3/7.



	Nombre	Duración	Inicio	Fin	Predecesora
1	**Estudios de las conferencias de seguridad web**	71días	10/10/2019	16/01/2020	
2	Recopilación de estudios científicos sobre la seguridad web	17días	10/10/2019	01/11/2019	
3	Recopilación de estudios sobre el cryptojacking	7días	04/11/2019	12/11/2019	2
4	Síntesis de los estudios y recopilación de ideas a este proyecto.	32días	13/11/2019	26/12/2019	3
5	Análisis de los proyectos utilizados en los estudios en caso de estar liberados	47días	13/11/2019	16/01/2020	3
6	**Análisis del crawler existente (IsKrawler)**	21días	10/10/2019	07/11/2019	
7	Implementación de la tecnología WebSocket en el crawler.	4días	10/10/2019	15/10/2019	
8	Implementación de la recogida de ficheros WebAssembly (Wasm).	11días	10/10/2019	24/10/2019	
9	Implementación del perfilado del procesador en el navegador.	7días	10/10/2019	18/10/2019	
10	Pruebas del perfecto funcionamiento del desarrollo	3días	25/10/2019	29/10/2019	7,8,9
11	**Período de crawling web para obtener datos**	7días	30/10/2019	07/11/2019	10
12	**IsKrawler Analyzer**	108días	17/01/2020	16/06/2020	**1,10**
13	Creación del proyecto (local, Github)	1día	17/01/2020	17/01/2020	
14	Diseño de la estructura	7días	20/01/2020	28/01/2020	13
15	Diseño e implementación de la entrada de datos y de los posibles argumentos	2días	29/01/2020	30/01/2020	14
16	Diseño e implementación de los modelos a usar.	4días	31/01/2020	05/02/2020	15
17	Diseño y desarrollo del análisis de la URL.	24días	06/02/2020	10/03/2020	16
18	Diseño y desarrollo del análisis los ficheros Wasm.	52días	06/02/2020	17/04/2020	16
19	Diseño e implementación del análisis de ficheros.	33días	06/02/2020	23/03/2020	16
20	Diseño e implementación del análisis de la perfilación del procesador.	31días	06/02/2020	19/03/2020	16
21	Diseño y desarrollo del análisis de las conexiones mediante WebSockets.	28días	06/02/2020	16/03/2020	16
22	Implementación de APIs y de otras bibliotecas	25días	06/02/2020	11/03/2020	16
23	Implementación de la salida de los resultados (JSON, HTML, BD y Verbose)	34días	20/04/2020	04/06/2020	22,21,20,19
24	Pruebas del perfecto funcionamiento	4días	05/06/2020	10/06/2020	23
25	Prueba del funcionamiento sobre los datos anteriormente recopilados	4días?	11/06/2020	16/06/2020	24,10

Fig. 5.7: Diagrama de Gantt para este proyecto 4/7.

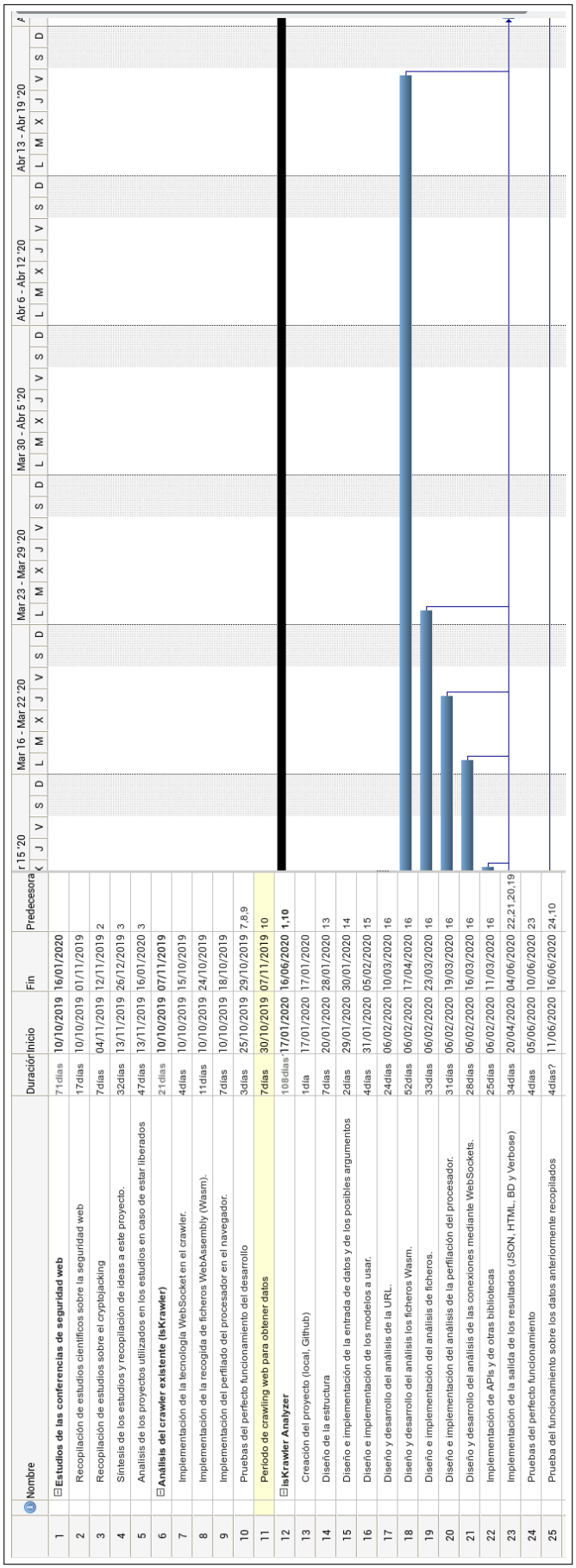
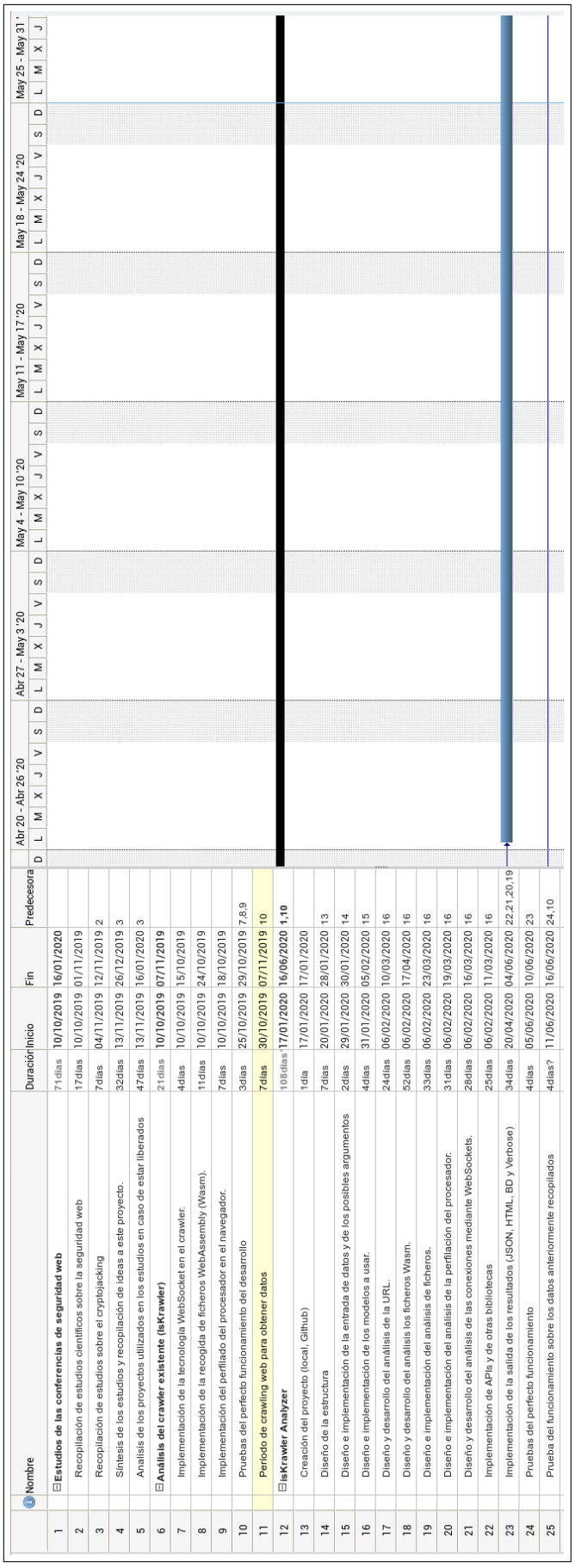
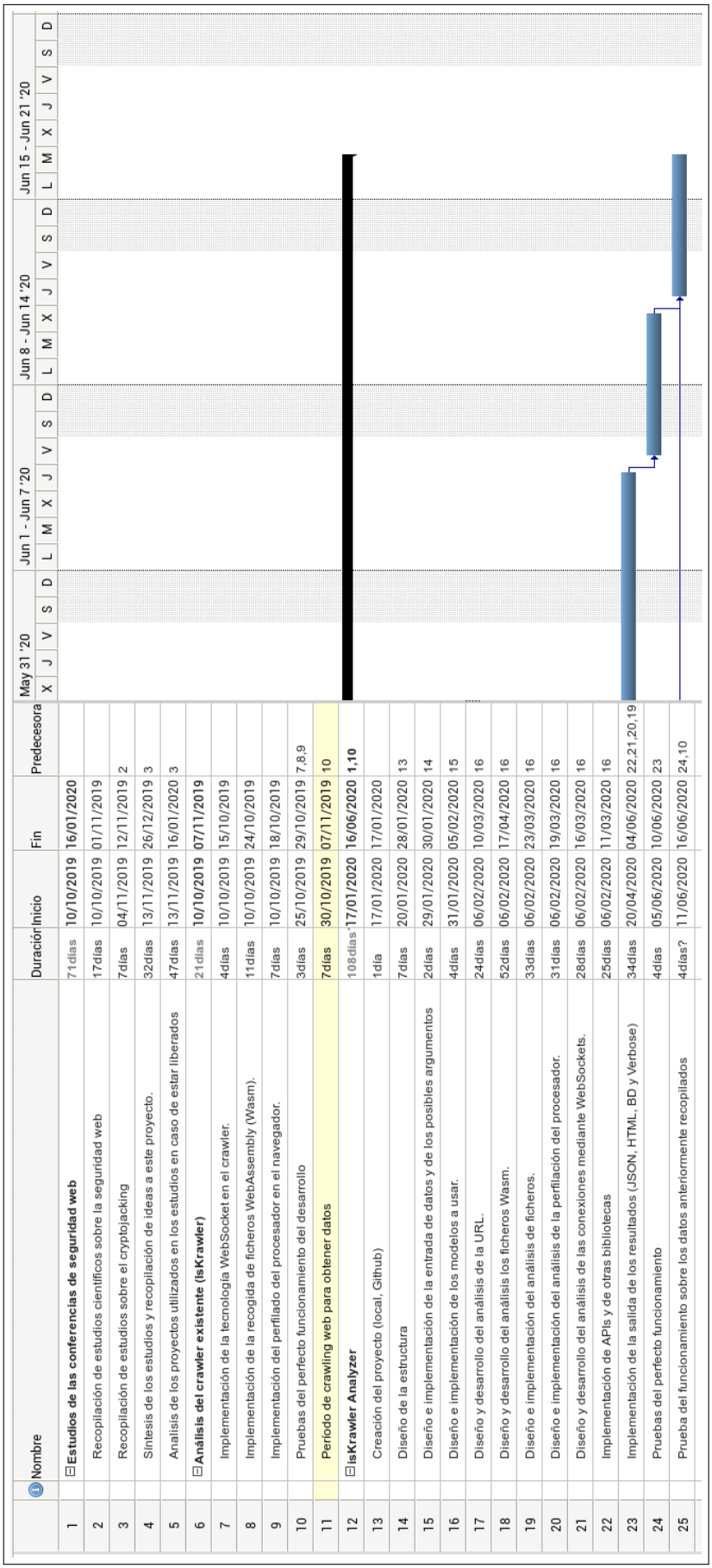


Fig. 5.8: Diagrama de Gantt para este proyecto 5/7.



	Nombre	Duración	Inicio	Fin	Predecesora
1	**Estudios de las conferencias de seguridad web**	71días	10/10/2019	16/01/2020	
2	Recopilación de estudios científicos sobre la seguridad web	17días	10/10/2019	01/11/2019	
3	Recopilación de estudios sobre el cryptojacking	7días	04/11/2019	12/11/2019	2
4	Síntesis de los estudios y recopilación de ideas a este proyecto.	32días	13/11/2019	26/12/2019	3
5	Análisis de los proyectos utilizados en los estudios en caso de estar liberados	47días	13/11/2019	16/01/2020	3
6	**Análisis del crawler existente (IsKrawler)**	21días	10/10/2019	07/11/2019	
7	Implementación de la tecnología WebSocket en el crawler.	4días	10/10/2019	15/10/2019	
8	Implementación de la recogida de ficheros WebAssembly (Wasm).	11días	10/10/2019	24/10/2019	
9	Implementación del perfilado del procesador en el navegador.	7días	10/10/2019	18/10/2019	
10	Pruebas del perfecto funcionamiento del desarrollo	3días	25/10/2019	29/10/2019	7,8,9
11	**Período de crawling web para obtener datos**	7días	30/10/2019	07/11/2019	10
12	**IsKrawler Analyzer**	108días	17/01/2020	16/06/2020	**1,10**
13	Creación del proyecto (local, Github)	1día	17/01/2020	17/01/2020	
14	Diseño de la estructura	7días	20/01/2020	28/01/2020	13
15	Diseño e implementación de la entrada de datos y de los posibles argumentos	2días	29/01/2020	30/01/2020	14
16	Diseño e implementación de los modelos a usar.	4días	31/01/2020	05/02/2020	15
17	Diseño y desarrollo del análisis de la URL.	24días	06/02/2020	10/03/2020	16
18	Diseño y desarrollo del análisis los ficheros Wasm.	52días	06/02/2020	17/04/2020	16
19	Diseño e implementación del análisis de ficheros.	33días	06/02/2020	23/03/2020	16
20	Diseño e implementación del análisis de la perfilación del procesador.	31días	06/02/2020	19/03/2020	16
21	Diseño y desarrollo del análisis de las conexiones mediante WebSockets.	28días	06/02/2020	16/03/2020	16
22	Implementación de APIs y de otras bibliotecas	25días	06/02/2020	11/03/2020	16
23	Implementación de la salida de los resultados (JSON, HTML, BD y Verbose)	34días	20/04/2020	04/06/2020	22,21,20,19
24	Pruebas del perfecto funcionamiento	4días	05/06/2020	10/06/2020	23
25	Prueba del funcionamiento sobre los datos anteriormente recopilados	4días?	11/06/2020	16/06/2020	24,10

Fig. 5.9: Diagrama de Gantt para este proyecto 6/7.



	Nombre	Duración	Inicio	Fin	Predecesora
1	**Estudios de las conferencias de seguridad web**	71días	10/10/2019	16/01/2020	
2	Recopilación de estudios científicos sobre la seguridad web	17días	10/10/2019	01/11/2019	
3	Recopilación de estudios sobre el cryptojacking	7días	04/11/2019	12/11/2019	2
4	Síntesis de los estudios y recopilación de ideas a este proyecto.	32días	13/11/2019	26/12/2019	3
5	Analisis de los proyectos utilizados en los estudios en caso de estar liberados	47días	13/11/2019	16/01/2020	3
6	**Análisis del crawler existente (IsKrawler)**	21días	10/10/2019	07/11/2019	
7	Implementación de la tecnología WebSocket en el crawler.	4días	10/10/2019	15/10/2019	
8	Implementación de la recogida de ficheros WebAssembly (Wasm).	11días	10/10/2019	24/10/2019	
9	Implementación del perfilado del procesador en el navegador.	7días	10/10/2019	18/10/2019	
10	Pruebas del perfecto funcionamiento del desarrollo	3días	25/10/2019	29/10/2019	7,8,9
11	Período de crawling web para obtener datos	7días	30/10/2019	07/11/2019	10
12	**IsKrawler Analyzer**	108días	17/01/2020	16/06/2020	**1,10**
13	Creación del proyecto (local, Github)	1día	17/01/2020	17/01/2020	
14	Diseño de la estructura	7días	20/01/2020	28/01/2020	13
15	Diseño e implementación de la entrada de datos y de los posibles argumentos	2días	29/01/2020	30/01/2020	14
16	Diseño e implementación de los modelos a usar.	4días	31/01/2020	05/02/2020	15
17	Diseño y desarrollo del análisis de la URL.	24días	06/02/2020	10/03/2020	16
18	Diseño y desarrollo del análisis los ficheros Wasm.	52días	06/02/2020	17/04/2020	16
19	Diseño e implementación del análisis de ficheros.	33días	06/02/2020	23/03/2020	16
20	Diseño e implementación del análisis de la perfilación del procesador.	31días	06/02/2020	19/03/2020	16
21	Diseño y desarrollo del análisis de las conexiones mediante WebSockets.	28días	06/02/2020	16/03/2020	16
22	Implementación de APIs y de otras bibliotecas	25días	06/02/2020	11/03/2020	16
23	Implementación de la salida de los resultados (JSON, HTML, BD y Verbose)	34días	20/04/2020	04/06/2020	22,21,20,19
24	Pruebas del perfecto funcionamiento	4días	05/06/2020	10/06/2020	23
25	Prueba del funcionamiento sobre los datos anteriormente recopilados	4días?	11/06/2020	16/06/2020	24,10

Fig. 5.10: Diagrama de Gantt para este proyecto 7/7.

Capítulo 6

Presupuesto

El coste de este proyecto ha sido relativamente cero, debido a que el software o APIs de terceros que se ha utilizado, además de los proyectos y documentos de conferencias han sido gratuitos. El entorno de desarrollo que se ha utilizado nos ha permitido hacer uso de nuestra cuenta de estudiante para que fuese gratuito. Teniendo en cuenta todo esto, y que el proyecto se ha desarrollado y probado en el ordenador del estudiante en cuestión, no ha habido ningún coste material en el proyecto. El único coste económico presupuestado del proyecto (el cual se muestra en la tabla 6.1) será el relativo al tiempo, dicho de otro modo, a las horas invertidas por el estudiante trabajando como diversos perfiles en el desarrollo del mismo.

	Precio (euro/hora)	**Horas estimadas**	**Total estimado (euros)**
Desarrollador principal	35	300	10,500.00 €
Investigador	40	130	5,200.00 €
Gestor del proyecto	45	85	3,825.00 €
Ingeniero del software	35	100	3,500.00 €
Desarrollador web	30	50	1,500.00 €
Desarrollador de Bases de datos	30	30	900.00 €
Administrador de sistemas	35	20	700.00 €
TOTAL		715	26,125.00 €

Tabla 6.1: Presupuesto del proyecto.

Capítulo 7

Conclusión

En este capítulo se procede a exponer las distintas conclusiones a las que se han llegado con el desarrollo y la finalización de este proyecto. En este proyecto se ha implementado una herramienta, que complementándose con el crawler ya existente e incluso añadiéndole métodos, es capaz de de analizar la seguridad de una página web.

Para comprobar el éxito del proyecto se van a revisar las metas que se establecieron al comienzo, examinando con el resultado del proyecto si esos objetivos del capítulo 3 han sido cumplidos y de qué forma.

Respecto a los objetivos principales, se puede afirmar que se han cumplido con creces y de una forma mejor a la esperada. Se ha conseguido desarrollar esa herramienta funcional que, complementándose con el IsKrawler, fuese capaz de formar una arquitectura para el análisis de páginas web, proporcionando resultados exhaustivos para cada uno de los análisis. Además, el IsKrawler Analyzer no solo es capaz de examinar los resultados y ficheros emitidos por el crawler, sino que además recopila información que no se puede obtener simplemente navegando por la página, aportándonos una visión más amplia y con más indicadores a valorar.

Cabe destacar que la herramienta desarrollada es intuitiva y fácil de usar, tanto a la hora de ejecutar un análisis sobre un objetivo, como a la hora de ofrecernos los resultados con diagramas incorporados que amenizan la visualización de los mismos. Esto permite que personas que no tengan muchos conocimientos de la materia puedan hacer uso de ella y contemplar superficialmente los resultados que presenta. Además, para estas personas se ha desarrollado un sistema de evaluación o "scoring" en el que se sintetizan todos los análisis en tres posibles estados, indicando al usuario si el resultado es benigno, sospechoso o malicioso.

El analizador ha sido desarrollado con un diseño modular, permitiendo así que tanto el desarrollo que se ha llevado a cabo como las futuras implementaciones que se realicen, resulten sencillas de aplicar. Dicho de otro modo, este diseño nos permite que la modificación, eliminación o adición de módulos sea sencilla. También permite que estos módulos puedan ser exportados a otros desarrollos de una forma simple.

Otro objetivo que se ha cumplido es el de ser una herramienta funcional y capaz de ser desplegada

en un entorno de producción. Esta herramienta, tal y como se ha podido demostrar en el apartado de evaluación 4.2.3, es perfectamente funcional y se podría poner en producción, pasando el proyecto a una escala mayor en cualquier momento, incluso pudiendo ser desplegado en una arquitectura más amplia. Además, el IsKrawler Analyzer es capaz de realizar todo el proceso de ejecución en un tiempo menor al minuto, mostrando así que es eficiente y rápido en un entorno de producción.

Otro detalle a resaltar es que se ofrecen una gran cantidad de opciones de parametrización de los análisis, permitiendo que no solo sirva para usuarios finales, sino también para futuras investigaciones científicas o incluso para exportar unos resultados que serán posteriormente usados por otros programas. Todo ello gracias a los distintos parámetros de los que se puede hacer uso para el análisis y presentación de los resultados.

Este proyecto ha permitido al estudiante desarrollarse enormemente a nivel personal y profesional, debido a que todas las responsabilidades y tareas recaían sobre él. Además se ha dado cuenta de todo lo que implica desarrollar un proyecto completo desde cero y también le ha permitido adquirir conocimientos sobre estudios científicos relevantes en el área del proyecto, presentados en algunos de los mejores congresos internacionales sobre seguridad informática.

Como se ha comentado anteriormente, se han podido mantener los ritmos de trabajo cumpliendo así con los tiempos desarrollados en los diagramas de Gantt5.4. Esto ha permitido realizar buenas pruebas de evaluación 4.2.3, dando lugar a un gran volumen de análisis sobre una ingente cantidad de sitios web, comprobando así que la propia arquitectura, formada por el IsKrawler y el analizador, es realmente funcional en entornos reales, tal y como se comentaba anteriormente.

En cuanto a las posibles aplicaciones del sistema, la principal y más directa se daría en el caso de que un usuario deseara comprobar si una web es maligna antes de acceder a ella. Pero no se descarta que, en el caso de un usuario experimentado que domina la materia y tiene conocimientos del tema, este sistema se pueda utilizar para poder extraer análisis sobre una web como etapa previa a examinar detenidamente y manualmente cierta cuestión de seguridad o cualquier otro tema de interés relacionado. Por último, otro uso interesante también podría ser el llevado a cabo por "bug hunters", personas que se dedican a participar en plataformas para la búsqueda de errores en determinadas webs por las que son recompensadas con dinero.

Es probable que alguno de los análisis que permite el proyecto todavía no sea muy utilizado debido a que se están utilizando tecnologías muy recientes, como es el caso del WebAssembly y de las comunicaciones mediante WebSockets. Aún así, se prevé que en un futuro muy cercano el uso de estas tecnología se extenderá a la mayoría de sitios web. Es por ello que, aunque actualmente los análisis que provee la herramienta no se pueden realizar a gran escala porque no hay muchas webs que utilicen las tecnologías mencionadas, es previsible que en poco tiempo los desarrollos llevados a cabo en este proyecto serán de gran utilidad y habrá sido un acierto haberlos desarrollado con suficiente anticipación como para poder ir ampliando sus funcionalidades cuando estas tecnologías se popularicen.

También es de recibo mencionar que, al ser la seguridad web un tema tan amplio y con muchas implicaciones en cualquier sistema, en el proyecto se ha tratado de alcanzar la máxima funciona-

lidad posible, pero especializándose en unos pocos análisis. Ello ha provocado que en el presente trabajo se haya hecho uso de muchos proyectos y bibliotecas de terceros sin poder profundizar al máximo en cada uno de ellos. Aún así, se puede concluir que el proyecto ha sido un éxito, ya que se han completado satisfactoriamente los objetivos propuestos inicialmente y se ha finalizado en los plazos previstos en la planificación.

7.1. Trabajo futuro

En este apartado se mencionarán posibles líneas de trabajo para el futuro, que no tiene porque ser lejano. Dicho de otro modo, se describirán distintas posibilidades que se deberían abordar tras este proyecto para que siguiera teniendo validez y aceptación en entornos reales.

La línea más previsible es la de seguir añadiendo funcionalidades en los análisis del IsKrawler Analyzer, convirtiéndola con cada una de ellas en una herramienta más completa y mejorada por los resultados que sería capaz de presentar. Unos ejemplos interesantes serían poder realizar análisis sobre las redirecciones de una web o sobre sus ficheros CSS y HTML.

Respecto a los análisis ya desarrollados, se puede seguir ahondando y profundizando en cada uno de ellos para ser capaces de reconocer de forma más sencilla qué tipo de actividades realiza y determinar con más fiabilidad aún si un resultado es benigno o no. Como ya se ha mencionado, los ficheros WebAssembly y las comunicaciones mediante WebSockets serán dos de las tecnologías que a día de hoy son novedosas y que en poco tiempo serán habituales en el desarrollo web. Por ello, ahondar en el estudio de estas dos tecnologías, tanto ampliando desarrollos en el analizador como revisando posibles nuevos artículos sobre las mismas en las conferencias de seguridad web, podría ser otra gran línea de trabajo a partir de los resultados de este proyecto fin de grado.

Asimismo, se puede tratar de minimizar el uso de bibliotecas y programas externos que se utilizan actualmente, añadiendo esos análisis de forma manual en el propio analizador, haciendo que no dependa de terceros.

No se debe olvidar tampoco que otra línea natural de trabajo futuro, menos interesante a priori, pero importante igualmente, sería la del mantenimiento de la propia herramienta. Mantener el proyecto eliminando módulos que empiecen a presentar un mal funcionamiento o proceder a arreglarlos para que sigan teniendo un buen desempeño puede ser una tarea desafiante. También se deberían mantener las diferentes formas de presentar los resultados que dependen fuertemente de los distintos módulos que haya en cada momento y los análisis que se realicen.

Otra línea de trabajo interesante podría ser la dockerización completa de esta herramienta, tanto el IsKrawler como el IsKrawler Analyzer, para construir una arquitectura completa y facilitar el despliegue de la misma. Incluso la posibilidad de usar Vagrant [131] para preparar una máquina virtual lista para correr análisis con solo dos comandos poidría ser una línea de trabajo igualmente interesante.

Por último, otra línea de trabajo muy clara, es la de darle más accesibilidad a la herramienta. Este

enfoque se puede observar en la siguiente sección 7.2, donde se pueden ver soluciones que suelen tener una web accesible desde la cual hacer los análisis y recibir los resultados, pudiendo añadir una gestión de usuarios y de distintos tipos de licencias.

7.2. Proyectos similares

Actualmente existe una gran cantidad de soluciones para escanear una web, ofreciendo distintos tipos de servicios que normalmente suelen ser de pago. En este apartado, se van a mostrar una serie de ejemplos de herramientas que realizan análisis similares al que realiza nuestro proyecto, pudiendo realizar una comparación con los mismos para poner de relieve que estos proyectos similares no suelen tener análisis profundos en algún aspecto, como puede ser el uso de la tecnología WebAssembly, y que en nuestro caso sí está presente en el módulo de Wasm 4.2.2.6.

Se ha incluido esta sección fundamentalmente para poder realizar una comparativa real entre nuestro proyecto y las soluciones reales que existen a día de hoy por Internet y que constituyen verdaderos modelos de negocio para algunas empresas. En las herramientas que se van a nombrar se probará su uso con una URL encontrada en el transcurso del proyecto y que realiza cryptojacking.

7.2.1. VirusTotal

El primer ejemplo es el popular sitio web VirusTotal [132], que nos ofrece un portal con una gran cantidad de funcionalidad, cada una para analizar distintos elementos. Entre ellas, podemos analizar un sitio web indicándole la URL. En la figura 7.1 se muestra una imagen de ejemplo de este análisis. Observando el resultado, podemos darnos cuenta como el análisis más importante que realiza, dado que es por lo que es popular, es a comparar en los distintos antivirus la web. Esto no nos permite realmente realizar un análisis técnico ni extenso de la web, pero es interesante para un analizador más amplio tener esta funcionalidad y por eso se decidió implementar en el IsKrawler Analyzer.

7.2.2. Webutation

Otro proyecto similar es el de Webutation, comentado anteriormente porque se hace uso de él en los distintos módulos. Anteriormente se ha incluido una imagen 4.4 de los resultados que este emite. Por lo tanto, en caso de querer examinarla, se debe acudir a la sección en cuestión. Como también se explicaba en el transcurso del proyecto, se trata de una comunidad abierta sobre la reputación de sitios web que recopila comentarios de los usuarios y experiencia de los clientes. Hace uso de la información recopilada de lugares como Google SAfe Browsing [105], antivirus webs, Web Of Trust (WOT) [106], Norton Safe Web [107] y otros. Nos permite sencillamente buscar una URL y nos emitiría un resultado entre cero y cien, siendo 70 que no tiene información del lugar.

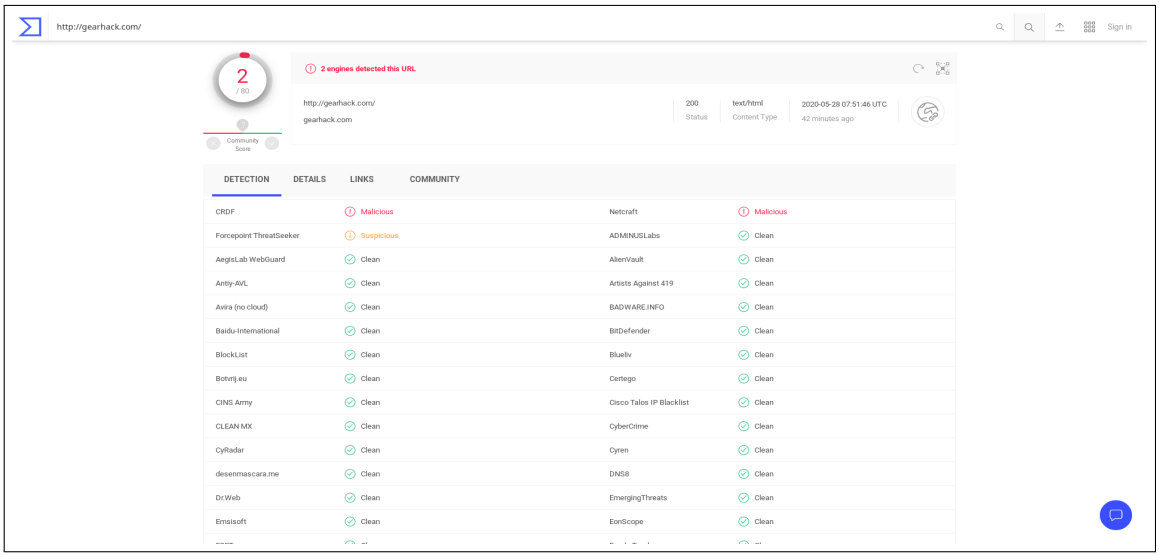


Fig. 7.1: Consulta en VirusTotal de una página web (Web: gearhack.com).

7.2.3. URLVoid

URLVoid [133] es un servicio gratuito desarrollado por NoVirusThanks [134] que permite a los usuarios escanear una dirección web con varios motores de reputación de sitios web y listas negras de dominios para facilitar la detección de posibles sitios web peligrosos, utilizados con actividades fraudulentas, malware...

En la imagen 7.2 podemos observar como clasifica una web y los distintos resultados que nos permite visualizar. Como podemos ver, algunos de los resultados que emite, son los mismos análisis que realiza nuestra herramienta, pero por otra parte, no hace uso de análisis sobre las comunicaciones mediante WebSockets o comprueba si existen ficheros Wasm malignos.

7.2.4. Zulu URL Risk Analyzer

Este sitio web [135] es otra opción a la hora de analizar una web. Al igual que nuestra herramienta utiliza un "scoring" el cual determina si es una web maliciosa o no. El problema de esta web, es que en muchos de los resultados que realiza, no nos ofrecen una gran información técnica, no pudiendo así saber el por qué ha resultado ser maligna. En la figura 7.3 se puede observar un ejemplo de este análisis.

7.2.5. Sucuri Sitecheck

Otro ejemplo de este tipo de proyectos es el de Sucuri Sitecheck [136]. Este sitio web comprobará, dada una URL, si contiene algún virus conocido, si está incluída en alguna lista negra, si el sitio web contiene algún tipo de error, si contiene software desactualizado o si tiene incrustado código malicioso. Se incluye una imagen 7.4 en la que podemos observar un ejemplo de esta herramienta con una URL que realiza cryptojacking.

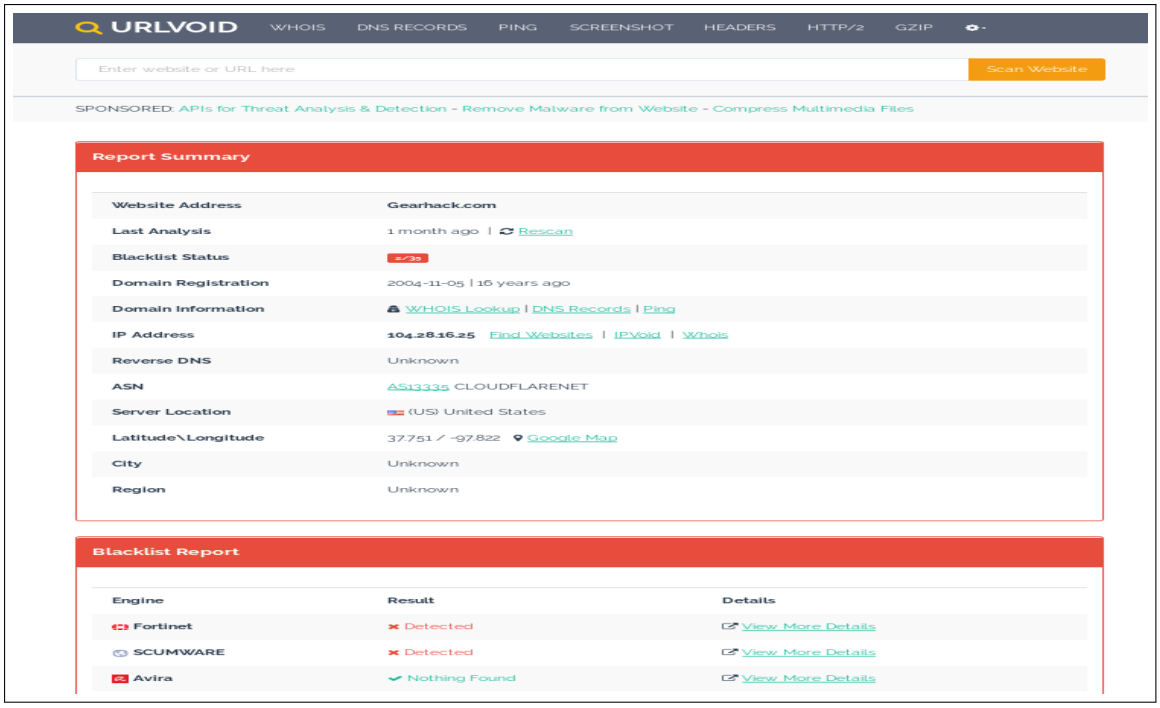


Fig. 7.2: Consulta en URLVoid de una página web (Web: gearhack.com).

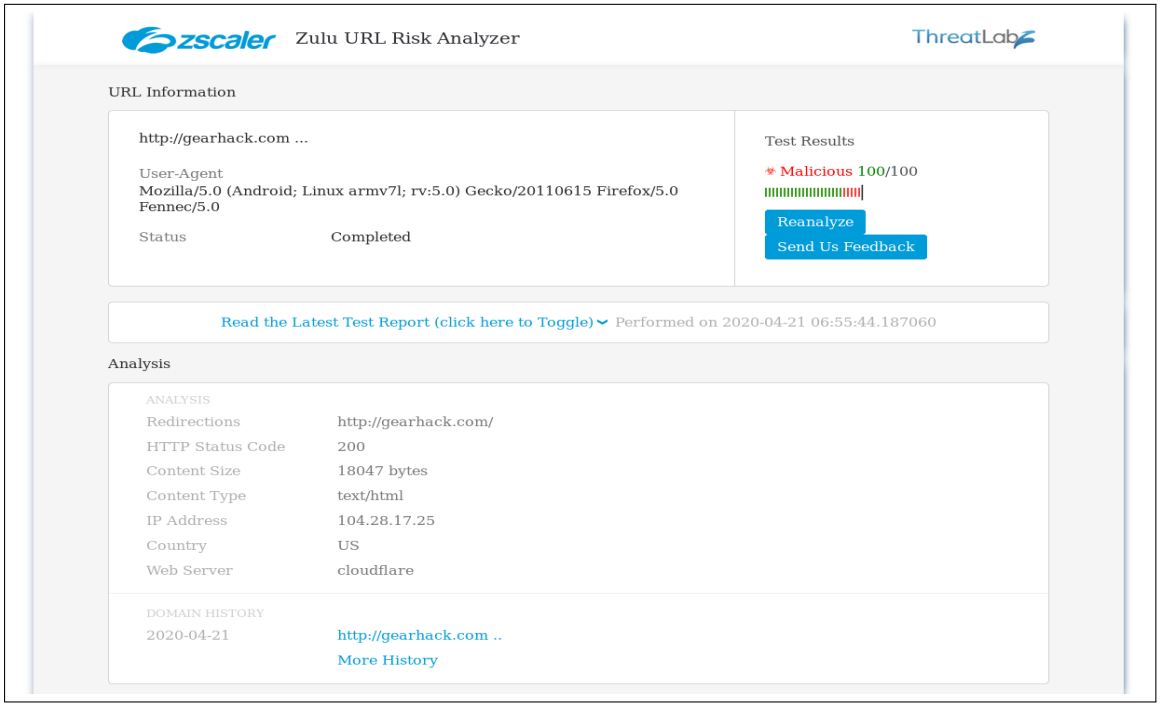


Fig. 7.3: Consulta en Zulu URL Risk Analyzer de una página web (Web: gearhack.com).

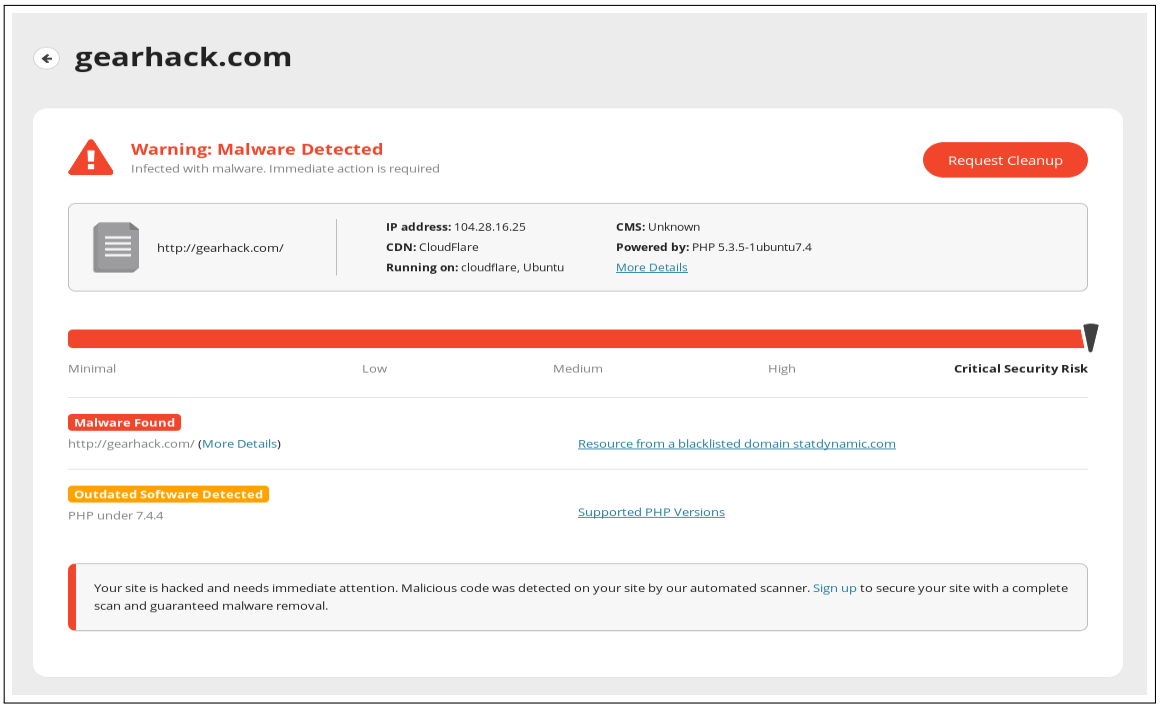


Fig. 7.4: Consulta en Sucuri Sitecheck de una página web (Web: gearhack.com).

7.2.6. Siteguarding Sitecheck

La solución Siteguarding Sitecheck [137] nos permite, al igual que el resto analizar una URL desde su sitio web. Todos las herramientas comentadas tienen análisis muy similares, y en este caso este también. Por ejemplo, hace uso también de listas negras o comprueba las versiones del software que está corriendo el sitio web. En la imagen 7.5 se puede examinar un análisis de esta herramienta.

Fig. 7.5: Consulta en Siteguarding Sitecheck de una página web (Web: gearhack.com).

Bibliografía

[1] Wget. http://www.gnu.org/software/wget/, (consultado el 23/04/20).

[2] Curl. https://curl.haxx.se/, (consultado el 23/04/20).

[3] Selenium. https://www.selenium.dev/, (consultado el 23/04/20).

[4] PhantomJS. https://phantomjs.org/, (consultado el 23/04/20).

[5] VV8. https://kapravelos.com/projects/vv8, (consultado el 23/04/20).

[6] SlimerJS. https://slimerjs.org/, (consultado el 23/04/20).

[7] Puppeter. https://github.com/puppeteer/puppeteer, (consultado el 23/04/20).

[8] Chrome Remote Interface. https://github.com/cyrus-and/chrome-remote-interface, (consultado el 23/04/20).

[9] OpenWPM Framework. https://github.com/mozilla/OpenWPM, (consultado el 29/05/20).

[10] Selenium foto. https://www.pngocean.com/gratis-png-clipart-dmaxf, (consultado el 25/04/20).

[11] PhantomJS foto. https://commons.wikimedia.org/wiki/File:Phantomjs-logo.png, (consultado el 25/04/20).

[12] VUSec Minesweeper Paper. https://dl.acm.org/doi/pdf/10.1145/3243734.3243858, (consultado el 16/05/20).

[13] How You Get Shot in the Back: A Systematical Study aboutCryptojacking in the Real World. https://dl.acm.org/doi/pdf/10.1145/3243734.3243840, (consultado el 18/05/20).

[14] Octopus Github. https://github.com/cyrus-and/chrome-remote-interface, (consultado el 16/05/20).

[15] A first look at browser-based cryptojacking. https://arxiv.org/pdf/1803.02887.pdf, (consultado el 30/05/20).

[16] VUSec Minesweeper Github. https://github.com/vusec/minesweeper, (consultado el 16/05/20).

[17] Heritrix. https://github.com/internetarchive/heritrix3, (consultado el 13/04/20).

[18] Internet Archive. https://archive.org/, (consultado el 13/04/20).

[19] Photon. https://github.com/s0md3v/Photon, (consultado el 13/04/20).

[20] OSINT. https://www.incibe-cert.es/blog/osint-la-informacion-es-poder, (consultado el 15/06/20).

[21] Tracking Inspector. http://paginaspersonales.deusto.es/isantos/papers/2018/2018-sanchez-rola-dimva-knocking.pdf, (consultado el 21/04/20).

[22] Webs Fraudulentas COVID-19. https://www.lavanguardia.com/politica/20200327/48113002773/policia-nacional-fraude-web-12000-coronavirus-vacunas-milagrosas.html, (consultado el 21/04/20).

[23] Detect Headless Browsing. https://github.com/infosimples/detect-headless, (consultado el 28/05/20).

[24] ACM Internet Measurement Conference 2019. https://conferences.sigcomm.org/imc/2019/, (consultado el 28/05/20).

[25] VisibleV8: In-browser Monitoring of JavaScript in the Wild (Documento). https://kapravelos.com/publications/vv8-imc19.pdf, (consultado el 28/05/20).

[26] VisibleV8: In-browser Monitoring of JavaScript in the Wild (Vídeo). https://vimeo.com/showcase/6531379/video/369121825#t=3500s, (consultado el 28/05/20).

[27] VisibleV8: In-browser Monitoring of JavaScript in the Wild (Transparencias). https://kapravelos.com/presentations/vv8-imc19.pdf, (consultado el 28/05/20).

[28] VisibleV8: In-browser Monitoring of JavaScript in the Wild (GitHub). https://github.com/wspr-ncsu/visiblev8, (consultado el 28/05/20).

[29] V8 Engine. https://v8.dev/, (consultado el 28/05/20).

[30] Chrome DevTools Protocol. https://chromedevtools.github.io/devtools-protocol/, (consultado el 29/05/20).

[31] Puppeter Demo. https://try-puppeteer.appspot.com/, (consultado el 29/05/20).

[32] NPM Chrome Remote Interface. https://www.npmjs.com/package/chrome-remote-interface, (consultado el 29/05/20).

[33] Chrome Remote Interface demo. https://github.com/AlbertoFDR/basic-chrome-crawler, (consultado el 29/05/20).

[34] University of Princeton. https://www.princeton.edu/, (consultado el 29/05/20).

[35] Mozilla Webpage. https://www.mozilla.org/, (consultado el 29/05/20).

[36] OpenWPM : An automated platform for web privacy measurement. https://senglehardt.com/papers/openwpm_03-2015.pdf, (consultado el 29/05/20).

[37] Online Tracking:A 1-million-site Measurement and Analysis. https://www.cs.princeton.edu/ arvindn/publications/OpenWPM_1_million_site_tracking_measurement.pdf, (consultado el 29/05/20).

[38] OpenWPM Demo. https://github.com/mozilla/OpenWPM/wiki/Platform-Demo, (consultado el 29/05/20).

[39] Wikipedia Internet Fraud. https://en.wikipedia.org/wiki/Internet_fraud, (consultado el 29/05/20).

[40] Noticia de 20minutos sobre estafadas en tiempos del coronavirus. https://www.20minutos.es/noticia/4206878/0/enganos-estafas-perpetradas-crisis-coronavirus/, (consultado el 29/05/20).

[41] Europol reportar Scam. https://www.europol.europa.eu/report-a-crime/report-cybercrime-online, (consultado el 29/05/20).

[42] Australia Report Scam. https://www.scamwatch.gov.au/, (consultado el 29/05/20).

[43] USA Report Scam. https://www.usa.gov/stop-scams-frauds, (consultado el 29/05/20).

[44] Consumer Report Scam. https://www.econsumer.gov/#crnt, (consultado el 29/05/20).

[45] Tipos de scam. https://www.scamwatch.gov.au/types-of-scams, (consultado el 29/05/20).

[46] Botnet Spam Campaigns Can Be Long Lasting: Evidence, Implications, and Analysis. https://web.eecs.umich.edu/ zmao/Papers/sigm09_spamcamp.pdf, (consultado el 29/05/20).

[47] Spamscatter: Characterizing Internet Scam Hosting Infrastructure. https://www.usenix.org/legacy/events/sec07/tech/full_papers/anderson/anderson.pdf, (consultado el 29/05/20).

[48] Wikipedia Estafa Nigeriana. https://es.wikipedia.org/wiki/Estafa_nigeriana, (consultado el 29/05/20).

[49] Spamscatter: Characterizing Internet Scam Hosting Infrastructure. https://www.cs.umd.edu/ elaine/docs/scambaiter.pdf, (consultado el 29/05/20).

[50] Unicef Cryptojacking. https://www.thehopepage.org/, (consultado el 30/05/20).

[51] Crypto Me0wing Attacks: Kitty Cashes in on Monero. https://www.imperva.com/blog/crypto-me0wing-attacks-kitty-cashes-in-on-monero/, (consultado el 30/05/20).

[52] Showtime websites secretly mined user CPU for cryptocurrency. https://www.theverge.com/2017/9/26/16367620/showtime-cpu-cryptocurrency-monero-coinhive/, (consultado el 30/05/20).

[53] Cryptojacking Campaign Exploits Drupal Bug, Over 400 Websites Attacked. https://threatpost.com/cryptojacking-campaign-exploits-drupal-bug-over-400-websites-attacked/131733/, (consultado el 30/05/20).

[54] Hacked Websites Mine Cryptocurrencies. https://blog.sucuri.net/2017/09/hacked-websites-mine-crypocurrencies.html, (consultado el 30/05/20).

[55] Pulitzer-winning website Politifact hacked to mine crypto-coins in browsers. https://www.theregister.com/2017/10/13/politifact_mining_cryptocurrency/, (consultado el 30/05/20).

[56] Cryptojacking Attack Found on Los Angeles Times Website. https://threatpost.com/cryptojacking-attack-found-on-los-angeles-times-website/130041/, (consultado el 30/05/20).

[57] Now even YouTube serves ads with CPU-draining cryptocurrency miners. https://arstechnica.com/information-technology/2018/01/now-even-youtube-serves-ads-with-cpu-draining-cryptocurrency-miners/, (consultado el 30/05/20).

[58] Cryptocurrency Web Miner Script Injected into AOL Advertising Platform. https://blog.trendmicro.com/trendlabs-security-intelligence/cryptocurrency-web-miner-script-injected-into-aol-advertising-platform/, (consultado el 30/05/20).

[59] UK ICO, USCourts.gov... Thousands of websites hijacked by hidden crypto-mining code after popular plugin pwned. https://www.theregister.com/2018/02/11/browsealoud_compromised_coinhive/, (consultado el 30/05/20).

[60] Mozilla asm.js. http://asmjs.org/spec/latest/, (consultado el 30/05/20).

[61] About HTML5 WebSocket. http://www.websocket.org/aboutwebsocket.html, (consultado el 30/05/20).

[62] Stratum Mining Protocol. https://slushpool.com/help/topic/stratum-protocol/, (consultado el 30/05/20).

[63] Hardening Stratum, the Bitcoin Pool Mining Protocol. https://content.sciendo.com/view/journals/popets/2017/3/article-p57.xml, (consultado el 30/05/20).

[64] Stratum Protocol Readme. https://github.com/aeternity/protocol/blob/master/STRATUM.md, (consultado el 30/05/20).

[65] Stratum mining protocol. https://en.bitcoin.it/wiki/Stratum_mining_protocol, (consultado el 30/05/20).

[66] How You Get Shot in the Back: A Systematical Study aboutCryptojacking in the Real World, Github. https://github.com/deluser8/cmtracker, (consultado el 18/05/20).

[67] WebAssembly Binary Toolkit (WABT). https://github.com/WebAssembly/wabt, (consultado el 16/05/20).

[68] The Web is Watching You: A Comprehensive Review of Web Tracking Techniques and Countermeasures. https://pdfs.semanticscholar.org/3648/2a90e109af6ea4ea9e2f08b013ecc47e84cb.pdf, (consultado el 30/05/20).

[69] Do Not Track? Advertisers Say 'Don't Tread on Us'. https://www.nytimes.com/2012/10/14/technology/do-not-track-movement-is-drawing-advertisers-fire.html, (consultado el 30/05/20).

[70] FPDetective: Dusting the Web for Fingerprinters. https://www.esat.kuleuven.be/cosic/publications/article-2334.pdf, (consultado el 30/05/20).

[71] Third-Party Web Tracking: Policy and Technology. https://jonathanmayer.org/publications/trackingsurvey12 (consultado el 30/05/20).

[72] Detecting and Defending Against Third-Party Tracking on the Web. https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final17.pdf, (consultado el 30/05/20).

[73] Cookieless Monster:Exploring the Ecosystem of Web-based Device Fingerprinting. https://sites.cs.ucsb.edu/ chris/research/doc/oakland13_cookieless.pdf, (consultado el 30/05/20).

[74] The Web Never Forgets:Persistent Tracking Mechanisms in the Wild. https://securehomes.esat.kuleuven.be/ gacar/persistent/the_web_never_forgets.pdf, (consultado el 30/05/20).

[75] DoNotTrack (DNT). https://tools.ietf.org/pdf/draft-mayer-do-not-track-00.pdf, (consultado el 30/05/20).

[76] PriVaricator: Deceiving Fingerprinterswith Little White Lies. https://www.doc.ic.ac.uk/ livshits/papers/pdf/www15.pdf, (consultado el 30/05/20).

[77] I Do Not Know What You Visited Last Summer:Protecting Users from Third-party Web Tracking with TrackingFree Browser. https://www.ndss-symposium.org/wp-content/uploads/2017/09/01_5_0.pdf, (consultado el 30/05/20).

[78] Art Python. https://pypi.org/project/art/, (consultado el 27/05/20).

[79] Argparse Python. https://docs.python.org/3/library/argparse.html, (consultado el 27/05/20).

[80] Python WHOIS. https://pypi.org/project/pythonwhois/, (consultado el 21/05/20).

[81] Python Geoip2. https://pypi.org/project/geoip2/, (consultado el 22/05/20).

[82] Beautiful Soup. https://pypi.org/project/beautifulsoup4/, (consultado el 20/05/20).

[83] JS Beutifier. https://pypi.org/project/jsbeautifier/, (consultado el 20/05/20).

[84] Re Python. https://docs.python.org/3/library/re.html, (consultado el 27/05/20).

[85] Graphviz Python. https://graphviz.readthedocs.io/en/stable/index.html, (consultado el 17/05/20).

[86] SQLAlchemy. https://www.sqlalchemy.org/, (consultado el 25/05/20).

[87] Psycopg2 Python. https://pypi.org/project/psycopg2/, (consultado el 27/05/20).

[88] Python Jinja2. https://jinja.palletsprojects.com/en/2.11.x/, (consultado el 25/05/20).

[89] Safety Python. https://github.com/pyupio/safety, (consultado el 27/05/20).

[90] Scancode-toolkit Python. https://scancode-toolkit.readthedocs.io/, (consultado el 27/05/20).

[91] Wikipedia: NodeJS. https://es.wikipedia.org/wiki/Node.js, (consultado el 30/05/20).

[92] Heroku. https://dashboard.heroku.com/, (consultado el 23/05/20).

[93] Chrome Debuggin Protocol Resource Types. https://chromedevtools.github.io/devtools-protocol/tot/Network/#type-ResourceType, (consultado el 16/05/20).

[94] Wikipedia WebSockets. https://es.wikipedia.org/wiki/WebSocket, (consultado el 22/05/20).

[95] Chrome Debuggin Protocol WebSockets Methods. https://chromedevtools.github.io/devtools-protocol/tot/Network/, (consultado el 22/05/20).

[96] Chrome Debuggin Protocol Profiler Methods. https://chromedevtools.github.io/devtools-protocol/tot/Profiler/, (consultado el 22/05/20).

[97] Wikipedia WHOIS. https://es.wikipedia.org/wiki/WHOIS, (consultado el 21/05/20).

[98] Alexa About. https://www.alexa.com/about, (consultado el 22/05/20).

[99] Alexa Static Ranking List. http://s3.amazonaws.com/alexa-static/top-1m.csv.zip, (consultado el 21/05/20).

[100] SimilarWeb About. https://data.similarweb.com/api/v1/data?domain=, (consultado el 22/05/20).

[101] Symantec Categorización. https://sitereview.bluecoat.com/, (consultado el 22/05/20).

[102] Virus Total API. https://developers.virustotal.com/reference, (consultado el 22/05/20).

[103] Python Geoip2 Data. https://dev.maxmind.com/geoip/geoip2/geolite2/, (consultado el 22/05/20).

[104] Webutation. http://www.webutation.net/, (consultado el 22/05/20).

[105] Google Safebrowsing. https://safebrowsing.google.com/, (consultado el 22/05/20).

[106] Web Of Trust (WOT). https://www.mywot.com/, (consultado el 22/05/20).

[107] Norton Safe Web. https://safeweb.norton.com/, (consultado el 22/05/20).

[108] Webutation About. http://www.webutation.net/go/about, (consultado el 22/05/20).

[109] Mozilla Observatory. https://observatory.mozilla.org/, (consultado el 22/05/20).

[110] Mozilla Observatory API Docs. https://github.com/mozilla/http-observatory/blob/master/httpobs/docs/api.md, (consultado el 22/05/20).

[111] Web JSFuck. http://www.jsfuck.com/, (consultado el 19/05/20).

[112] Web RegExpr. https://regexr.com/, (consultado el 19/05/20).

[113] Proyecto AntiJSFuck. https://github.com/eagleoflqj/antiJSFuck, (consultado el 19/05/20).

[114] JS Obfuscator Web. https://www.daftlogic.com/projects-online-javascript-obfuscator.htm, (consultado el 20/05/20).

[115] JS Link Finder. https://github.com/GerbenJavado/LinkFinder, (consultado el 21/05/20).

[116] Open Cookie Database Github. https://github.com/jkwakman/Open-Cookie-Database, (consultado el 19/05/20).

[117] Cookie Database. https://cookiedatabase.org/, (consultado el 19/05/20).

[118] ACM CCS Conference (2018). https://www.sigsac.org/ccs/CCS2018/index.html, (consultado el 18/05/20).

[119] End-to-End Analysis of In-Browser Cryptojacking. https://arxiv.org/pdf/1809.02152.pdf, (consultado el 18/05/20).

[120] Patrick Ventuzelo (Octopus) Talk. https://patrickventuzelo.com/2019/06/talk-first-2019-analyze-detect-webassembly-cryptominer/, (consultado el 16/05/20).

[121] Python Libreria Subprocess. https://docs.python.org/3/library/subprocess.html, (consultado el 16/05/20).

[122] Outguard: Detecting In-Browser Covert Cryptocurrency Miningin the Wild. https://dl.acm.org/doi/pdf/10.1145/3308558.3313665, (consultado el 21/05/20).

[123] JQ Linux Tool. https://stedolan.github.io/jq/, (consultado el 25/05/20).

[124] Wikipedia Grep Command. https://es.wikipedia.org/wiki/Grep, (consultado el 25/05/20).

[125] Wikipedia Desarrollo en Espiral. https://es.wikipedia.org/wiki/Desarrollo_en_espiral, (consultado el 23/05/20).

[126] Kanban Información. https://kanbanize.com/es/recursos-de-kanban/primeros-pasos/que-es-kanban, (consultado el 24/05/20).

[127] Github Projects. https://help.github.com/es/github/managing-your-work-on-github/about-project-boards, (consultado el 24/05/20).

[128] Lucidchart. https://app.lucidchart.com/, (consultado el 25/05/20).

[129] DBeaver. https://dbeaver.io/, (consultado el 26/05/20).

[130] Gantter Web. https://www.gantter.com/, (consultado el 24/05/20).

[131] Vagrant. https://www.vagrantup.com/, (consultado el 30/05/20).

[132] VirusTotal Url. https://www.virustotal.com/gui/home/url, (consultado el 28/05/20).

[133] URLVoid. https://www.urlvoid.com/, (consultado el 28/05/20).

[134] NoVirusThanks Company. https://www.novirusthanks.org/, (consultado el 28/05/20).

[135] Zulu URL Risk Analyzer. https://zulu.zscaler.com/, (consultado el 28/05/20).

[136] Sucuri Sitecheck. https://sitecheck.sucuri.net/, (consultado el 28/05/20).

[137] Siteguarding Sitecheck. https://www.siteguarding.com/en/sitecheck, (consultado el 28/05/20).

Agradecimientos

En primer lugar, me gustaría agradecer a mi tutor, David, por haberme ayudado a concretar el proyecto y por su gran paciencia a la hora de responder mis mensajes de correo electrónico. También, gracias a la Universidad de Deusto y a los profesores que durante el grado me han impartido clase, por haberme facilitado adquirir una gran cantidad de conocimientos, no solo técnicos, durante estos cuatro años.

Una vez dadas las gracias a mi Universidad y tutor me gustaría agradecer su colaboración a mis compañeros de DeustoTech, los cuales me han ayudado a resolver dudas coyunturales que han surgido durante el transcurso del proyecto, además de ayudarme a enfocarlo correctamente.

Por último, pero no por ello menos importante, gracias a mi familia, que ha supuesto en todo momento un apoyo moral muy importante, sobretodo dadas las circunstancias (SARS-CoV-2, confinamiento...) en las que se ha desarrollado el mismo. También deseo agradecer de manera especial a mis amigos que hayan sido un gran apoyo durante el periodo del proyecto fin de grado.

Sin todos ellos, este trabajo no hubiera sido posible. Muchas gracias.